Zilog

PRELIMINARY
Technical Manual

July 1987

Z280™ MPU
Microprocessor Unit

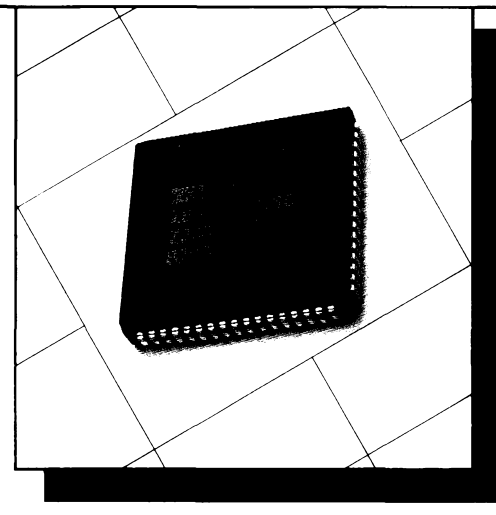


Table of Contents

Table of Contents (Continued)

5

Chapter 6. Interrupts and Traps

6

Table of Contents (Continued)

Table of Contents (Continued)

LIST OF ILLUSTRATIONS AND TABLES

Table of Contents (Continued)

Table of Contents (Continued)

Chapter 1.
Z280 Architectural Overview

1.1 INTRODUCTION

The Z280™ microprocessor unit (MPU) features an advanced 16-bit CPU that is object-code compatible with the Z80® CPU. The Z280 microprocessor unit includes memory management, peripherals, memory refresh logic, cache memory, wait state generators, and a clock oscillator on the same integrated circuit as the CPU. The on-chip peripheral devices include 4 DMA (Direct Memory Access) channels, 3 counter/timers, and a UART (Universal Asynchronous Receiver/Transmitter). A block diagram of the Z280 MPU is shown in Figure 1-1. This chapter presents some of the features of the Z280 MPU family, with detailed descriptions of the various aspects of the processor provided in succeeding chapters.

The Z280 MPU has a multiplexed address/data bus for communication with external memory and peripheral devices. Two different bus structures are supported by the Z280: an 8-bit data bus that uses Z80 Bus control signals, and a 16-bit data bus that uses Z-BUS® bus control signals. Zilog's Z80 and Z8500 families of peripherals are easily interfaced to the Z80 Bus; Zilog's Z8000® family of peripherals are easily interfaced to the Z-BUS.

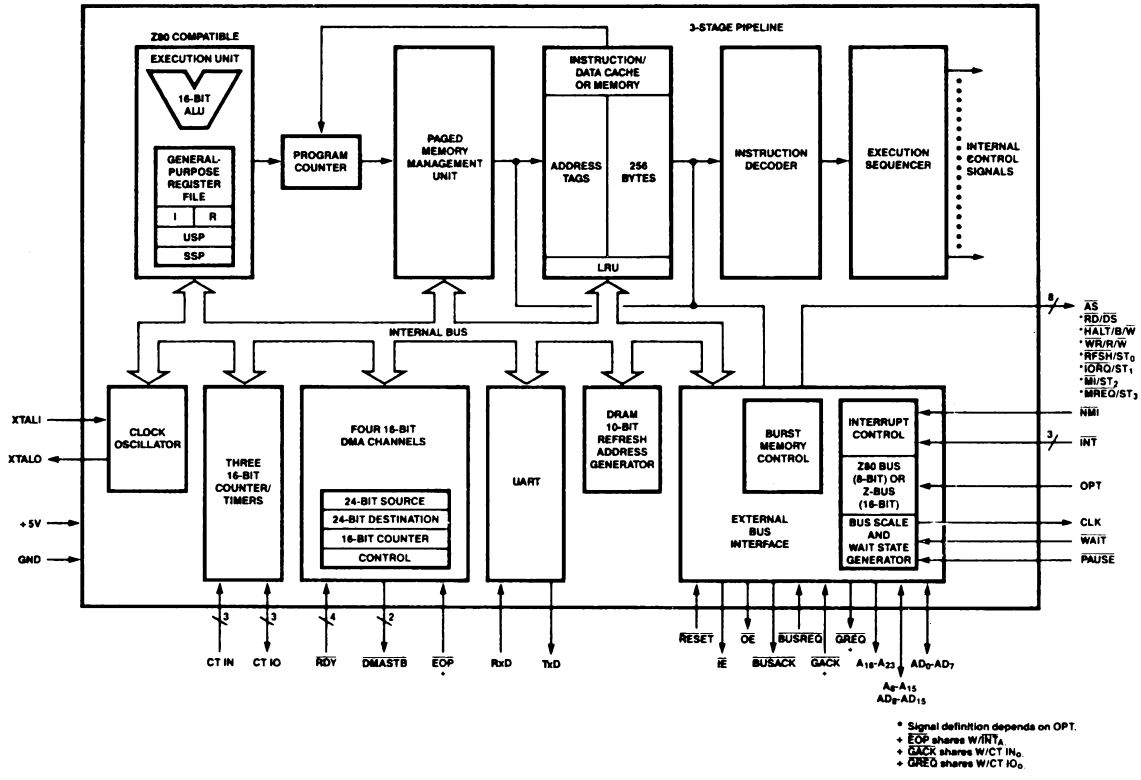


Figure 1-1. Block Diagram

1.2 MPU ARCHITECTURAL FEATURES

The central processing unit of the Z280 MPU is a binary-compatible extension of the Z80 CPU architecture. High throughput rates for the Z280 CPU are achieved by a high clock rate, instruction pipelining, and the use of on-chip cache memory. The internal CPU clock can be scaled down to provide for slower speed bus transaction timing. A programmable refresh mechanism for dynamic RAMs and a clock oscillator are provided on-chip.

1.2.1 System and User Modes

Two modes of CPU operation, system and user, are provided to facilitate operating system design. In system mode, all of the instructions can be executed and all of the CPU registers can be accessed. This mode is intended for use by programs performing operating system functions. In user mode, certain instructions that affect the state of the machine cannot be executed and the control registers in the CPU are inaccessible. In general, user mode is intended for use by applications programs. This separation of CPU resources promotes the integrity of the system, since programs executing in user mode cannot access those aspects of the CPU that deal with time-dependent or system-interface events.

The register structure has been extended to include separate Stack Pointer registers, one for a system-mode stack and one for a user-mode stack. The system-mode stack is used for saving program status on the occurrence of an interrupt or trap condition, thereby ensuring that the user stack is free of system information. The isolation of the system stack from user-mode programs further promotes system integrity.

1.2.2 Address Spaces

Addressing spaces in the Z280 CPU include the CPU register space, the CPU control register space, the memory address space, and the I/O address space. The CPU register file is identical to the Z80 register set, with the exception of the separate system- and user-mode Stack Pointers. The A register acts as an 8-bit accumulator; the HL register is the 16-bit accumulator. These are supplemented by four other 8-bit registers (B, C, D, E) and two other 16-bit registers (IX, IY); the 8-bit registers can be paired for 16-bit operation, and each 16-bit register can be treated as two 8-bit registers. The Flag register (F) contains information about the result of the last operation. The A, F, B, C, D, E, H, and L registers are replicated in an auxiliary bank of registers. These auxiliary registers can be exchanged with the primary register bank for fast context switching.

Several CPU control registers determine the operation of the Z280 MPU. For example, the contents of control registers determine the CPU operating mode, which interrupts are enabled, and the bus transaction timing. The control registers are accessible in system-mode operation only.

The Z280 CPU's logical memory address space is the same as that of the Z80 CPU: 16-bit addresses are used to reference up to 64K bytes of memory. However, the on-chip Memory Management Unit (MMU) extends the 16-bit logical memory address to a 24-bit physical memory address. Two separate logical address spaces, one for system mode and one for user mode, are supported by the CPU and MMU. Optionally, the MMU can be programmed to distinguish between instruction fetches and data accesses; thus, the Z280 CPU can have up to four memory address spaces: system-mode program, system-mode data, user-mode program, and user-mode data. The logical address space is divided into pages to facilitate controlled sharing of program or data among separate processes.

The Z280 CPU architecture also distinguishes between the memory and I/O address spaces and, therefore, requires specific I/O instructions. I/O addresses in the Z280 CPU are 24 bits long, with the upper 8 bits provided by an I/O page register in the CPU.

1.2.3 Data Types

Many data types are supported by the Z280 CPU architecture. The basic data type is the 8-bit byte, which is also the basic addressable memory element. The architecture also supports operations on bits, BCD digits, 2-byte words, and byte strings.

1.2.4 Addressing Modes

The operand addressing mode is the method by which a data operand's location is specified. The Z280 CPU supports nine addressing modes, including the five modes available on the Z80 CPU. The addressing modes of the Z280 CPU are:

- Register
- Immediate
- Indirect Register
- Direct Address
- Indexed (with a 16-bit displacement)
- Short Index (with an 8-bit displacement)
- Program Counter (PC) Relative
- Stack Pointer (SP) Relative
- Base Index

All addressing modes are available on the 8-bit load, arithmetic, and logical instructions; the 8-bit shift, rotate, and bit manipulation instructions are limited to the Register, Indirect Register, and Short Index addressing modes. The 16-bit loads on the addressing registers support all addressing modes except Short Index, while other 16-bit operations are limited to the Register, Immediate, Indirect Register, Index, Direct Address, and PC Relative addressing modes.

1.2.5 Instruction Set

The Z280 CPU instruction set is an expansion of the Z80 instruction set; the enhancements include support for additional addressing modes for the Z80 instructions as well as the addition of new instructions. The Z280 CPU instruction set provides a full complement of 8- and 16-bit arithmetic operations, including signed and unsigned multiplication and division. Additional 8-bit computational instructions support logical and decimal operations. Bit manipulation, rotate, and shift instructions round out the data manipulation capabilities of the Z280 CPU. The Jump, Call, and Return instructions have both conditional and unconditional versions; Relative addressing is provided for the Jump and Call instructions to support position-independent programs. Block move, search, and I/O instructions provide powerful data movement capabilities. In addition, special instructions have been included to facilitate multitasking, multiple processor configurations, and typical high-level language and operating system functions.

1.2.6 Exception Conditions

The Z280 MPU supports three types of exceptions (conditions that alter the normal flow of program execution): interrupts, traps, and resets.

Interrupts are asynchronous events typically triggered by peripherals requiring attention. The Z280 MPU interrupt structure has been significantly enhanced by increasing the number of interrupt request lines and by adding an efficient means for handling nested interrupts. There are four modes for handling interrupts:

- 8080 compatible, in which the interrupting device provides the first instruction of the interrupt routine.
- Dedicated interrupts, in which the CPU jumps to a dedicated address when an interrupt occurs.
- Vectored interrupt mode, in which the interrupting peripheral provides a vector into a table of jump addresses.
- Enhanced vectored interrupt mode, wherein the CPU handles traps and multiple interrupt sources, saving control information as well as the Program Counter when an interrupt occurs.

The first three modes are compatible with the Z80 CPU interrupt modes; the fourth mode provides more flexibility, with support for nested interrupts and a sophisticated vectoring scheme.

Traps are synchronous events that trigger a special CPU response when certain conditions occur during instruction execution. The Z280 CPU supports a sophisticated complement of traps including Division Exception, System Call, Privileged Instruction, Extended Instruction, Single-Step, Breakpoint-on-Halt, Memory Access Violation, and System Stack Overflow Warning traps.

Hardware resets occur when the $\overline{\text{RESET}}$ line is activated and override all other conditions. A reset causes certain CPU control registers to be initialized.

1.2.7 Memory Management

Memory management consists primarily of dynamic relocation, protection, and sharing of memory.

Proper memory management can provide a logical structure to the memory space that is independent of the actual physical location of data, protect the user from inadvertent mistakes (such as trying to execute data), prevent unauthorized accesses to memory, and protect the operating system from disruption by users.

The 16-bit addresses manipulated by the programmer, used by instructions, and output by the CPU are called logical addresses. The on-chip Memory Management Unit (MMU) transforms the logical addresses into the corresponding 24-bit physical addresses required for accessing memory. This address transformation process is called relocation, and makes user software independent of physical memory. Thus, the user is freed from specifying where information is actually located in physical memory.

Status information generated by the CPU allows the MMU to monitor the intended use of each memory access. Illegal types of accesses, such as writes to read-only memory, can be suppressed; thus, areas of memory can be protected from unintended or unwanted modes of use. Also, the MMU records which memory areas have been modified and can inhibit copies of data from being retained in the on-chip cache.

When a memory access violation is detected by the MMU, a trap condition is generated in the CPU and execution of the current instruction is automatically aborted. This mechanism facilitates the easy implementation of virtual memory systems based on the Z280 MPU.

1.2.8 Cache Memory

Cache memories are small high-speed buffers situated between the processor and main memory. For each memory access, control logic checks to see if the data at that memory location is currently stored in the cache. If so, the access is made to the high-speed cache; if not, the access is made to main memory, and the cache itself might be updated. Thus, use of a cache leads to increased performance with fewer memory transactions on the system bus.

The Z280 MPU includes on-chip memory that can be used as a cache for programs, data, or both. Cache operations, including updating, are performed automatically and are completely transparent to the user. Optionally, this on-chip memory can be dedicated to a set of memory locations that are specified under program control, instead of being used as a cache.

1.2.9 Refresh

The Z280 MPU has an internal mechanism for refreshing dynamic memory. This mechanism can be enabled or disabled under program control. If enabled, memory refresh operations are performed periodically at a rate determined by the contents of a refresh rate register. A 10-bit refresh address is generated for each refresh operation.

1.2.10 On-Chip Peripherals

Several programmable peripheral devices are included on-chip in the Z280 MPUs: four DMA channels, three 16-bit counter/timers, and a UART. Optionally, one of the DMA channels can be used with the UART as a bootstrap loader for the Z280 MPU's memory after a reset.

1.2.11 Multiprocessor Mode

A special mode of operation allows the Z280 MPU to operate in environments that have a global bus, wherein the Z280 MPU is not the bus master of the global bus. A set of memory addresses (determined under program control) is dedicated to a local bus, which is controlled by the Z280 MPU, and another set of addresses is used for the global bus. The Z280 MPU is required to make a bus request and receive an acknowledgement before making a memory access to an address on the global bus. This mode of operation facilitates use of the Z280 MPU in multiple-processor configurations. For example, a Z280 MPU could be used as an I/O processor in a Z80000-, Z8000-, or Z280-based system.

1.2.12 Extended Instruction Facility

The Z280 MPU architecture has a mechanism for extending the basic instruction set through the use of external devices called Extended Processing Units (EPUs). Special opcodes have been set aside to implement this feature. When the Z280 MPU encounters an instruction with one of these opcodes, it performs any indicated address calculations and data transfers; otherwise, it treats the "extended instruction" as if it were executed by the EPU.

If an EPU is not present, the Z280 MPU can be programmed to trap when an extended instruction is encountered so that system software can emulate the EPU's activity.

1.3 BENEFITS OF THE ARCHITECTURE

The features of the Z280 MPU architecture provide several significant benefits, including increased program throughput, increased integration of system functions, support for operating systems, and improvements in compiler efficiency and code density.

1.3.1 High Throughput

Very high throughput rates can be achieved with the Z280 MPU, due to the cache memory, instruction pipelining, and high clock rates achievable with this processor. The CPU clock rate can be scaled down to provide the bus clock rate, allowing the designer to use slower, less-expensive memory and I/O devices. Use of the on-chip cache memory further increases throughput by minimizing the number of accesses to the slower, off-chip memory devices. The high code density achievable with the Z280 CPU's expanded instruction set also contributes to program throughput, since fewer instructions are needed to accomplish a given task.

1.3.2 Integration of System Functions

Besides a powerful CPU, the Z280 MPU includes many on-chip devices that previously had to be implemented in logic external to the microprocessor chip. These devices include a clock oscillator, memory refresh logic, wait state generators, the MMU, cache memory, DMA channels, counter/timers, and a UART. Integration of all these functions onto a single chip results in a reduced parts count in a system design, accompanied by a resulting reduction in design and debug time, power requirements, and printed circuit board space. This increased level of integration also contributes to system throughput, since the on-chip devices can be accessed quickly without the need of an external bus transaction.

1.3.3 Operating System Support

Several of the Z280 MPU's architectural features facilitate the implementation of multitasking operating systems for Z280-based systems.

The inclusion of user and system operating modes improves operating system organization. User-mode programs are automatically inhibited from performing operating-system type functions. System-mode memory can be separated from user-mode memory and separate stacks can be maintained for system-mode and user-mode operations. The System Call instruction and the trap mechanism provide a controlled means of accessing operating system functions during user-mode execution.

The interrupt- and trap-handling mechanisms are well suited for operating system implementations. Several levels of interrupts are provided, allowing for separate control of various peripheral devices (both on and off the chip). A new interrupt mode is provided, wherein status information about the currently executing task is saved on the stack and new program status information for the service routine is automatically loaded from a special memory area. Traps result in the same type of program status saving. In both cases, status is always saved on the system stack, leaving the user stack undisturbed.

Allocation of resources within the operating system can be accomplished using a special Test and Set instruction. Other instructions, such as the Purge Cache instruction, are provided to aid in task switching and other operating system chores.

The on-chip MMU supports a multitasking environment by providing both a means of quickly allocating physical memory to tasks as they are executed on the system and protection mechanisms to enforce proper memory usage.

1.3.4 Code Density

Code density affects both processor speed and memory utilization. Code compaction saves memory space and improves processor speed by reducing the number of instructions that must be fetched and decoded. The largest reduction in program size results from the powerful instruction set, where instructions such as Multiply and Divide help substantially reduce the number of instructions required to complete a task.

The efficiency of the instruction set is enhanced by the addition of new addressing modes. For example, all nine addressing modes are available for all the 8-bit load, arithmetic, and logical instructions.

1.3.5 Compiler Efficiency

For microprocessor users, the transition from assembly language to high-level languages allows greater freedom from architectural dependency and improves ease of programming. For the Z280 MPUs, high-level language support is provided through the inclusion of features designed to minimize typical compilation and code-generation problems.

Among these features is the variety and the power of the Z280 instruction set, allowing the Z280 CPU to easily handle a large amount and variety of data types. The Z280 CPU's ability to manipulate many different data types aids in compiler efficiency; since data structures are high-level constructs frequently used in programming, processing performance is enhanced by providing efficient mechanisms for manipulating them.

Examples of commonly used data structures include arrays, strings, and stacks. Arrays are supported in the Z280 CPU by the Indirect Register, Index, and Base Index addressing modes. Strings are supported by those same addressing modes and the Block Move and Compare instructions; since compilers and assemblers often must manipulate character strings, the Block Move and Block Compare instructions can result in dramatic speed improvements over software simulations of those tasks. Numeric strings of BCD data can be manipulated using the Decimal Adjust and Rotate Digit instructions. Stacks are supported by the Push and Pop instructions and the Stack Pointer Relative, Index, and Base Index addressing modes; the Stack Pointer Relative addressing mode is especially useful for accessing parameters and local variables stored on the stack.

1.4 SUMMARY

The Z280 MPU is a high-performance 16-bit microprocessor, available with 8- and 16-bit external bus interfaces. Code-compatible with the Z80 CPU, the Z280 MPU architecture has been expanded to include features such as multiple memory address spaces, efficient handling of nested interrupts, system and user operating modes, and support for multiprocessor configurations. Additional functions such as memory management, clock generation, wait state generation, and cache memory are included on-chip, as well as a number of peripheral devices. The benefits of this architecture--including high throughput rates, a high level of system integration, operating system support, code density, and compiler efficiency--greatly enhance the power and versatility of the Z280 MPU. Thus, the Z280 MPU provides both a growth path for existing Z80-based designs and a high-performance processor for future applications.

Chapter 2.
Address Spaces

2.1 INTRODUCTION

The Z280 MPU supports four address spaces corresponding to the different types of locations that can be addressed, the method by which the logical addresses are formed, and the translation mechanisms used to map the logical address into physical locations. These four address spaces are:

- **CPU register space.** This consists of the addresses of all registers in the CPU register file.
- **CPU control register space.** This consists of the addresses of all registers in the CPU control register file.
- **Memory address space.** This consists of the addresses of all locations in the main memory.
- **I/O address space.** This consists of the addresses of all I/O ports through which peripheral devices are accessed, including on-chip peripherals and MMU registers.

2.2 CPU REGISTER SPACE

The Z280 CPU register file is illustrated in Figure 2-1. The primary register file, consisting of the A, F, B, C, D, E, H, and L registers, is augmented by an auxiliary file containing duplicates of those registers. Only one set (either the primary or auxiliary file) can be used at any one time. Special exchange instructions are provided for switching between the primary and auxiliary registers.

The CPU register file is divided into five groups of registers (an apostrophe indicates a register in the auxiliary file):

- Flag and accumulator registers (F, A, F', A')
- Byte/word registers (B, C, D, E, H, L, B', C', D', E', H', L')
- Index registers (IX, IY)
- Stack Pointers (SSP, USP)
- Program Counter, Interrupt register, and Refresh register (PC, I, R)

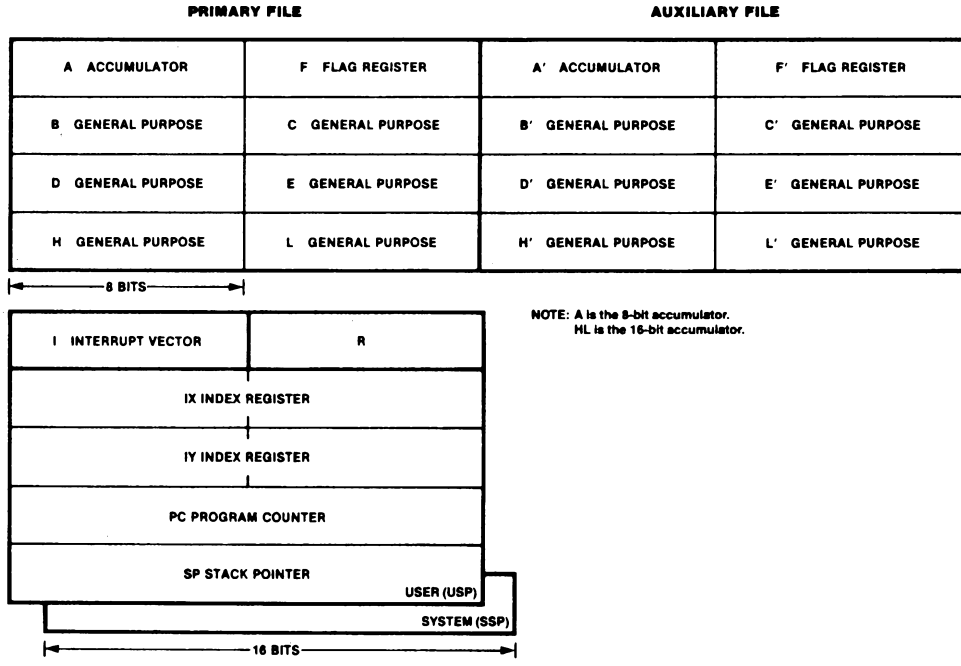


Figure 2-1. Register File Organization

Register addresses are either specified explicitly in the instruction or are implied by the semantics of the instruction.

The flag registers (F, F') contain eight status flags. Four can be individually used for control of program branching, two are used to support decimal arithmetic, and two are reserved (see section 5-2). The accumulator (A) is the implied destination (i.e., where the result is stored) for the 8-bit arithmetic and logical instructions. Two sets of flag and accumulator registers exist in the Z280 CPU, with only one set accessible as the flag register and the accumulator at any one time. An exchange instruction allows switching to the alternate flag register and accumulator.

The byte/word registers can be accessed either as 8-bit byte registers or 16-bit word registers. Bits within these registers can also be accessed individually. For 16-bit accesses, the registers are paired B with C, D with E, and H with L. Two sets of byte/word registers exist in the Z280 CPU, although only one set is used as the current byte/word registers; the other set is accessible as the alternate group of byte/word registers via an exchange instruction.

The index registers IX and IY can be accessed as 16-bit registers or their upper and lower bytes (IXH, IXL, IYH, and IYL) can be individually accessed.

The Z280 CPU has two hardware Stack Pointers, one dedicated to system mode operation and one to user mode operation. The System Stack Pointer (SSP) is used for saving information when an interrupt or trap occurs and for supporting subroutine calls and returns in system mode. The User Stack Pointer (USP) is used for supporting subroutine calls and returns in user mode.

The Program Counter is used to sequence through instructions in the currently executing program and for generating relative addresses. The Interrupt register is used in interrupt mode 2 to generate a 16-bit logical address from an 8-bit vector returned by a peripheral during an interrupt acknowledge. The Refresh register is used by the Z80 CPU to indicate the current refresh address, but does not perform this function in the Z280 CPU; instead, it is another 8-bit register available for the programmer.

The explicit or implicit register specified by an instruction is mapped into the CPU register file based on the state of three control bits. One of the three control bits is used to map the flag and accumulator registers, selecting either F, A or F', A' whenever the instruction specifies the flag register or the accumulator. Another control bit is used to map the byte/word registers, selecting the B, C, D, E, H, L registers or the B', C', D', E', H', L' registers. These two control bits are changed by the Exchange Flag and Accumulator and the Exchange Byte/Word Registers instructions, respectively. At any time the program can sense the state of these control bits by special jump instructions. The third control bit, the User/System control bit in the Master Status register, specifies whether the System Stack Pointer register or the User Stack Pointer register is selected whenever an instruction specifies the Stack Pointer register. In addition, the User Stack Pointer register also has an address in the CPU control register space via a special Load Control instruction.

2.3 CPU CONTROL REGISTER SPACE

The Z280 CPU status and control registers govern the operation of the CPU. They are accessible only by the privileged Load Control (LDCTL) instruction.

Control register addresses are specified by the contents of the C register. No translation is performed in mapping this 8-bit logical address into the control register file location.

The Z280 CPU control registers are the Bus Timing and Initialization register, the Bus Timing and Control register, the Master Status register, the Interrupt/Trap Vector Table Pointer, the I/O Page register, the System Stack Limit register, the Trap Control register, the Interrupt Status register, the Cache Control register, and the Local Address register (Figure 2-2). The CPU control registers are described in detail in Chapter 3.

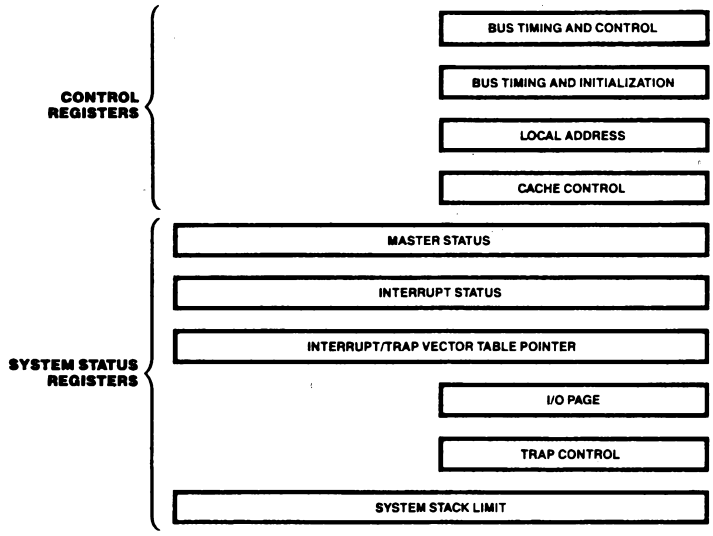


Figure 2-2. CPU Control Registers

2.4 MEMORY ADDRESS SPACES

Two memory address spaces, one for system and one for user mode operation, are supported by the Z280 MPU. They are selected by the User/System mode control bit in the Master Status register, which governs the selection of page descriptor registers in the MMU during address translation.

Each address space can be viewed as a string of 64K bytes numbered consecutively in ascending order. The 8-bit byte is the basic addressable element in the Z280 MPU memory address spaces. However, there are other addressable data elements: bits, 2-byte words, byte strings, and multiple-byte EPU operands.

The size of the data element being addressed depends on the instruction being executed. A bit can be addressed by specifying a byte and a bit within that byte. Bits are numbered from right to left, with the least significant bit being bit 0, as illustrated in Figure 2-3.

Figure 2-3. Numbering of Bits within a Byte

The address of a multiple-byte entity is the same as the address of the byte with the lowest memory address within the entity. Multiple-byte entities can be stored beginning with either even or odd memory addresses. A word (2-byte entity) is aligned if its address is even; otherwise it is unaligned. Multiple bus transactions, which may be required to access multiple-byte entities, can be minimized if alignment is maintained.

The formats of multiple byte data types in memory are given in Figure 2-4.

Note that when a word is stored in memory, the least significant byte precedes the most significant byte of the word, as in the Z80 CPU architecture.

The 16-bit logical addresses generated by a program can be translated into 24-bit physical addresses by the on-chip MMU. When the translation mechanism is disabled, the 24-bit physical address consists of the logical address for bits A_0-A_{15} and zeros for A_{16}-A_{23}.

60-bit floating-point (EPU instruction only) at address n:

sign,E10-4	address n
E3-0, F51-48	address n+1
F47-40	address n+2
F39-32	address n+3
F31-24	address n+4
F23-16	address n+5
F15-8	address n+6
F7-0	address n+7
<-- 1 byte -->	

80-bit floating-point (EPU instructions only) at address n:

sign,E14-8	address n
E7-0	address n+1
F63-56	address n+2
F55-48	address n+3
F47-40	address n+4
F39-32	address n+5
F31-24	address n+6
F23-16	address n+7
F15-8	address n+8
F7-0	address n+9

BCD digit strings (EPU instruction only) at address n:
(up to 10 bytes in length; the illustration is for the maximum length string)

sign,D18	address n
D17,D16	address n+1
D15,D14	address n+2
D13,D12	address n+3
D11,D10	address n+4
D9,D8	address n+5
D7,D6	address n+6
D5,D4	address n+7
D3,D2	address n+8
D1,D0	address n+9

16-bit word at address n:

least significant byte	address n
most significant byte	address n+1
<------- 1 byte ------->	

32-bit integer (EPU instruction only) at address n:

B31-24 (most significant byte)	address n
B23-16	address n+1
B15-8	address n+2
B7-0 (least significant byte)	address n+3
<------------- 1 byte ------------->	

64-bit integer (EPU instruction only) at address n:

B63-56 (most significant byte)	address n
B55-48	address n+1
B47-40	address n+2
B39-32	address n+3
B31-24	address n+4
B23-16	address n+5
B15-8	address n+6
B7-0 (least significant byte)	address n+7
<------------- 1 byte ------------->	

32-bit floating-point (EPU instruction only) at address n:

sign,E7-1	address n
E0,F22-16	address n+1
F15-8	address n+2
F7-0	address n+3
<-- 1 byte -->	

Figure 2-4. Formats of Multiple-Byte Data Elements in Memory

2.5 I/O ADDRESS SPACE

I/O addresses are generated only by I/O instructions. The 8-bit logical port address specified in the instruction appears on AD_0-AD_7; this is concatenated with the contents of the A register on lines A_8-A_{15} for Direct addressing mode, or by the contents of the B register for Indirect Register addressing mode or block I/O instructions. The contents of the I/O Page register are appended to this address on lines A_{16}-A_{23}. Thus, the 24-bit I/O port address consists of the 8-bit address specified in the instruction, the contents of the A or B register, and the contents of the I/O Page register.

An I/O read or write is always one transaction, regardless of the bus size and the type of I/O instruction. On-chip peripherals with word registers are always accessed with word instructions, regardless of the size of the external bus.

Chapter 3.
CPU Control Registers

3.1 INTRODUCTION

Several CPU control and status registers specify the operating mode of the Z280 MPU. There are two types of CPU control registers: system configuration registers and system status registers. The system configuration registers contain information about the physical configuration of the Z280-based system, such as bus timing information. Typically, the system configuration registers are loaded once during system initialization and are not altered during subsequent operations. The system status registers contain information that may change during system operation, such as the current I/O page. Access to the CPU control registers is restricted to system mode operation only, using the privileged Load Control (LDCTL) instruction. Resets initialize the control registers so that a Z80 object program will execute successfully on the Z280 MPU. (Z80 programs do not affect these registers, since the Load Control instruction is not part of the Z80 CPU's instruction set.) Unused bits in these registers should always be loaded with zeros.

3.2 SYSTEM CONFIGURATION REGISTERS

There are four 8-bit system configuration registers: the Bus Timing and Initialization register, the Bus Timing and Control register, the Local Address register, and the Cache Control register.

3.2.1 Bus Timing and Initialization Register

The Bus Timing and Initialization register controls the scaling of the processor clock for bus timing, the duration of bus transactions to the lower half of physical memory, and the enabling of the multiprocessor and bootstrap modes. Figure 3-1 illustrates the bit fields in this register.

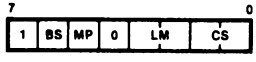


Figure 3-1. Bus Timing and Initialization Register

Clock Scaling (CS) Field. This 2-bit field governs the scaling of the CPU clock for generation of bus timing cycles. The state of the CS field determines the bus clock frequency for all bus transactions, as per Table 3-1. This field is initialized during a reset operation, as described below, and cannot be modified via software.

Table 3-1. CS Field of Bus Timing and Initialization Register

CS Field	Bus Clock Frequency
00	Bus clock frequency equals 1/2 CPU clock frequency (one bus clock cycle for every two CPU clock cycles)
01	Bus clock frequency equals CPU clock frequency (one bus clock cycle for every one CPU clock cycle)
10	Bus clock frequency equals 1/4 CPU clock frequency (one bus clock cycle for every four CPU clock cycles)
11	Reserved

Low Memory Wait Insertion (LM) Field. This 2-bit field specifies the number of automatic wait states to insert in memory transactions to the lower 8 megabytes of physical memory (that is, all memory locations where bit 23 of the physical address is a 0), as per Table 3-2. Additional wait states can still be added to any given memory transaction via control of the $\overline{WAIT}$ input.

Table 3-2. LM Field of Bus Timing and Initialization Register

LM Field	Number of Wait States for Lower 8M Bytes of Memory
00	0
01	1
10	2
11	3

Multiprocessor Configuration Enable (MP) Bit. This 1-bit field enables the multiprocessor mode of operation, wherein the Z280 MPU is connected to both a local and a global bus. Transactions to

addresses on the global bus require a special bus request and acknowledgement before the bus transaction can occur. (See Chapter 10 for details concerning this mode of operation.) Setting this bit to 1 enables the multiprocessor mode, and clearing this bit to 0 disables this mode.

Bootstrap Mode Enable (BS) Bit. This 1-bit field enables the bootstrap mode of operation. If the bootstrap mode is selected during a reset operation, memory is automatically initialized via the UART after the reset; the UART receiver and DMA channel 0 are used to transfer 256 bytes of data into the first 256 memory locations; execution then begins from memory location 0. (See Chapter 9 for further details.) Setting this bit to 1 enables the bootstrap mode and clearing this bit to 0 disables this mode. The BS bit can be set to 1 only during a reset operation, as described below. Writing to this bit via a software command has no effect. This bit is always a 1 when this register is read.

Bits 4 and 7 of the Bus Timing and Initialization register are reserved for special use by Zilog and should always be loaded with a zero when writing to this register. When this register is read, bits 4 and 7 may return a 1.

The Bus Timing and Initialization register can be initialized with either of two methods during a reset operation. If the MPU's $\overline{WAIT}$ input is not asserted during reset, this register is automatically initialized to all zeros, thereby specifying a bus clock frequency of one-half the internal CPU clock, no automatic wait states during transactions to the lower 8M bytes of memory, and disabling of the multiprocessor and bootstrap modes. If the $\overline{WAIT}$ input is asserted during reset, the Bus Timing and Initialization register is set to the contents of the AD_0-AD_7 bus lines, as read during the reset operation (see Chapter 12); this form of initialization is the only way to specify the bootstrap mode. Once the CS field has been loaded during reset, it cannot be modified via software; however, the LM and MP fields can be written using the LDCTL instruction.

3.2.2 Bus Timing and Control Register

The 8-bit Bus Timing and Control register determines the timing of bus transactions to the upper 8M bytes of memory and to all I/O devices, and the timing of interrupt acknowledge transactions. Figure 3-2 indicates the format of this register.

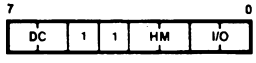


Figure 3-2. Bus Timing and Control Register

I/O Wait Insertion (I/O) Field. This 2-bit field specifies the number of automatic wait states (in addition to the one wait state always present during I/O transactions) to be inserted during each I/O read or write transaction, as per Table 3-3. The specified number of wait states is also added to the vector read portion of an interrupt acknowledge cycle.

Table 3-3. I/O Field of Bus Timing and Control Register

I/O Field	Number of Wait States for I/O
00	0
01	1
10	2
11	3

High Memory Wait Insertion (HM) Field. This 2-bit field specifies the number of automatic wait states to be inserted during memory transactions to the upper 8M bytes of physical memory (locations where address bit 23 of the physical address is a 1), as per Table 3-4.

Table 3-4. HM Field of Bus Timing and Control Register

HM Field	Number of Wait States for Upper 8M Bytes of Memory
00	0
01	1
10	2
11	3

Daisy Chain Timing (DC). This 2-bit field determines the number of automatic wait states to be inserted during interrupt acknowledge transactions while the interrupt acknowledge daisy chain is settling, as per Table 3-5. Normally, 2.5 bus clock cycles elapse between the assertion of Address Strobe and the assertion of Data Strobe during an interrupt acknowledge (for the Z-BUS) or between the assertion of $\overline{MT}$ and the assertion of $\overline{IORQ}$ (for the Z80 Bus). The value of the DC field determines if any additional clocks are to be added between the Address Strobe and Data Strobe (or $\overline{MT}$ and $\overline{IORQ}$) assertions.

Table 3-5. DC Field of Bus Timing and Control Register

DC Field	Number of Wait States for Interrupt Acknowledge
00	0
01	1
10	2
11	3

The contents of the Bus Timing and Control register govern the number of automatic wait states to be inserted during various bus transactions. Additional wait states can be added to any bus transaction via control of the $\overline{WAIT}$ input.

The Bus Timing and Control register is set to 30H by a reset. Bits 4 and 5 should always be written with 0. When this register is read, bits 4 and 5 may return a 1.

3.2.3 Local Address Register

The 8-bit Local Address register is used while in multiprocessor mode to determine which memory addresses are accessed via the local bus and which memory addresses are accessed via the global bus. If the multiprocessor mode is disabled (that is, if there is a 0 in bit 5 of the Bus Timing and Initialization register), the contents of the Local Address register have no effect on MPU operation.

If multiprocessor mode is enabled, the MPU automatically uses the Local Address register during each memory access to determine if the global bus is required. The Local Address register consists of a 4-bit match field and a 4-bit base field that are compared to the upper four bits of the physical memory address during memory transactions. The 4-bit match field specifies which bits of the physical memory address are of interest; for those bit positions specified in the match field, if all the corresponding address bits match the Local Address register's base field bits, then the bus transaction can proceed on the local bus. If there is a mismatch in at least one of the specified bit positions, then the global bus is requested, and the transaction cannot proceed until the global bus acknowledge signal is asserted. (See Chapter 10 for further discussion of the Multiprocessor mode.)

The format of the Local Address register is illustrated in Figure 3-3.

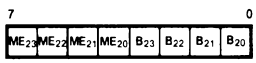


Figure 3-3. Local Address Register

Base bit (B_n): For each ME_n that is set to 1, the corresponding value of B_n must match the value of address bit A_n in order for the local bus to be used; otherwise, the transaction requires the use of the global bus.

Match Enable bit (ME_n): If ME_n is set to 1, then the corresponding physical address bit A_n is compared to base bit B_n to determine if the address requires the use of the global bus. If ME_n is a zero, then any values for A_n and B_n produce a match, signifying a local bus access. If every ME_n is cleared to 0, then all memory transactions are performed on the local bus.

The Local Address register is cleared to all zeros by a reset.

3.2.4 Cache Control Register

The 8-bit Cache Control register controls the operation of the on-chip memory. The contents of the Cache Control register determine if the on-chip memory is to be used as a cache or as fixed memory locations; if used as a cache, the cache can be enabled for instruction fetches only, for data fetches only, or for both instruction and data fetches. This register is also used to determine if burst-mode memory transactions are supported. (See Chapter 8 for further discussion of the on-chip memory and Chapter 13 for a description of the burst mode memory transaction.)

The Cache Control register contains five control bits, as described below. The format for this register is shown in Figure 3-4.

Figure 3-4. Cache Control Register

Memory/Cache (M/$\overline{C}$) Bit. While this bit is set to 1, the on-chip memory is accessed as physical memory with fixed memory addresses; the user can programmably select the ranges of memory addresses for which the on-chip memory will respond. While this bit is cleared to 0, the on-chip memory is accessed associatively as a cache.

Cache Instruction Disable (I) Bit. While this bit and the M/C bit are cleared to 0, the on-chip memory is used as a cache during instruction fetches. While this bit is set to 1, instruction fetches do not use the cache. If the M/C bit is a 1, the state of this bit is ignored.

Cache Data Disable (D) Bit. While this bit and the M/C bit are cleared to 0, the on-chip memory is used as a cache during data fetches. While this bit is set to 1, data fetches do not use the cache. (The cache can be enabled for both

instruction and data fetches by clearing both the I and D bits.) If the M/C bit is a 1, the state of this bit is ignored.

Low Memory Burst Capability (LMB) Bit. This 1-bit field specifies whether burst-mode memory transactions will occur during memory transactions to the lower 8M bytes of physical memory (locations where address bit 23 of the physical address is a 0). Setting this bit to 1 enables burst-mode transactions; clearing this bit to 0 disables burst mode transactions.

High Memory Burst Capability (HMB) Bit. This 1-bit field specifies whether burst-mode memory transactions will occur during memory transactions to the upper 8M bytes of physical memory (locations where address bit 23 of the physical address is a 1). Setting this bit to 1 enables burst-mode transactions; clearing this bit to 0 disables burst-mode transactions.

The Cache Control register is set to a 20_H (hexadecimal) by a reset, enabling the on-chip memory for use as a cache for instruction fetches only and disabling burst mode transactions. Bits 0, 1, and 2 of this register are not used.

3.3 SYSTEM STATUS REGISTERS

There are six system status registers in the Z280 CPU: the Master Status register, Interrupt Status register, Interrupt/Trap Vector Table Pointer, I/O Page register, Trap Control register, and System Stack Limit register.

3.3.1 Master Status Register

The 16-bit Master Status register (MSR) contains status information about the currently executing program. Typically, the MSR changes when a new programming task is dispatched; it changes automatically when an interrupt or trap occurs. For all traps and for interrupts processed using interrupt mode 3, the old value of the MSR is saved on the system stack and a new MSR is loaded along with the Program Counter to define the service routine. (See Chapter 6 for a detailed discussion of interrupt and trap processing).

The format of the Master Status register is shown in Figure 3-5.

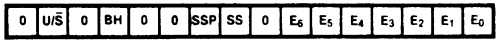


Figure 3-5. Master Status Register

User/System (U/$\overline{S}$) Bit. While this bit is cleared to 0, the Z280 MPU is in the system mode of operation; while set to 1, the MPU is in the user mode of operation. The current operating mode determines which Stack Pointer is in use and which instructions can be executed; privileged instructions can be executed only while in system mode.

Breakpoint-on-Halt Enable (BH) Bit. While this bit is set to 1, the CPU generates a breakpoint trap whenever a Halt instruction is encountered; while cleared to 0, the Halt instruction is executed normally.

Single-Step Pending (SSP) Bit. The CPU checks this bit prior to the start of an instruction execution and generates a Single-Step trap if this bit is set to 1. The Single-Step bit is automatically copied into this field at the completion of an instruction. This bit is automatically cleared when a Single-Step, Division Exception, Access Violation, Privileged Instruction, or Breakpoint-on-Halt trap is executed, so that the saved MSR has a 0 in this bit position. (For these traps, the PC address of the trapped instruction is saved for possible re-execution.)

Single-Step (SS) Bit. This bit is the enable for the single-step operating mode. While this bit is set to 1, the CPU is in a single-step mode wherein a Single-Step trap is generated for each instruction; if cleared to 0, single-step mode is disabled.

Interrupt Request Enable (E_n) Bit. There are seven interrupt enable bits in the MSR, one for each type of maskable interrupt source. The Z280 MPU's interrupt sources, including both the external interrupt requests and the on-chip peripherals, are grouped into seven levels of interrupt requests. While bit E_n is set to 1, interrupt requests from sources at level n are accepted by the CPU; while E_n is cleared to 0, interrupt requests from sources at level n are not accepted.

The Master Status register is loaded with all zeros by a reset. Bits 7, 10, 11, 13, and 15 of the MSR always should be written with zeros.

3.3.2 Interrupt Status Register

The 16-bit Interrupt Status register indicates which interrupt mode is in effect, which interrupt requests are pending, and which interrupt requests are to be vectored. Only the interrupt vector

enable bits are writeable; all other bits in this register are read-only status bits. The fields in the Interrupt Status register are shown in Figure 3-6.

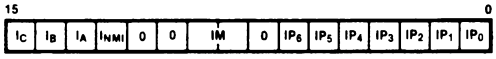


Figure 3-6. Interrupt Status Register

Interrupt Vector Enable (I_n) Bits. These four bits indicate which of the four external interrupt inputs are to be vectored. While I_n is set to 1, interrupts on the Interrupt n line are vectored when the CPU is in interrupt mode 3; while I_n is cleared to 0, that interrupt is not vectored. These bits are ignored when not in interrupt mode 3.

Interrupt Mode (IM) Field. This 2-bit field indicates the current interrupt mode in effect, with a value n in this field denoting interrupt mode n. This field can be changed by executing the IM instruction.

Interrupt Request Pending (IP_n) Bits. When bit IP_n is a 1, an interrupt request from a source at level n is pending.

On reset, the Interrupt Vector Enable bits are cleared to all zeros, interrupt mode 0 is in effect, and the Interrupt Pending bits reflect the state of the interrupt requests. Bits 7, 10, and 11 of this register are not used.

3.3.3 Interrupt/Trap Vector Table Pointer

The 16-bit Interrupt/Trap Vector Table Pointer contains the twelve most significant bits of the physical memory address of the start of the Interrupt/Trap Vector Table. The Interrupt/Trap Vector Table is a memory area that holds the values that are loaded into the Master Status register and Program Counter during trap and interrupt processing under interrupt mode 3, as described in Chapter 6. The twelve low-order bits of the 24-bit physical address are assumed to be all zeros: thus, the Interrupt/Trap Vector Table must start on a 4K byte boundary in physical memory. The low-order four bits of the Interrupt/Trap Vector Table Pointer must be all zeros (Figure 3-7).

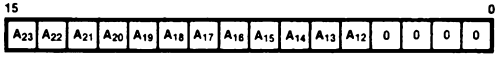


Figure 3-7. Interrupt/Trap Vector Table Pointer

The contents of the Interrupt/Trap Vector Table Pointer are unaffected by a reset and are undefined after power-up. When this register is read, bits 3,2,1 and 0 may return a 1.

3.3.4 I/O Page Register

The 8-bit I/O Page register determines the upper eight bits of the 24-bit peripheral address output during execution of an I/O transaction (Figure 3-8). I/O pages FEH and FFH are reserved for on-chip peripheral addresses.

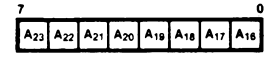


Figure 3-8. I/O Page Register

The contents of the I/O Page register are cleared to all zeros by a reset.

3.3.5 Trap Control Register

The 8-bit Trap Control register contains the enables for the maskable traps. Figure 3-9 illustrates the format of this register.

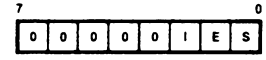


Figure 3-9. Trap Control Register

Inhibit User I/O (I) Bit. This bit determines whether or not I/O instructions are privileged instructions. While this bit is set to 1, all I/O instructions are treated as privileged instructions, and an attempt to execute an I/O instruction while in user mode results in a Privileged Instruction trap. While this bit is cleared to 0, I/O instructions can be successfully executed in user mode. I/O instructions can always be executed in system mode, regardless of the state of this bit.

EPU Enable (E) Bit. This bit indicates whether or not an Extended Processor Unit (EPU) is available in the system for execution of extended instructions. If this bit is cleared to 0, indicating that no EPUs are present, the CPU generates an Extended Instruction trap whenever an extended instruction is encountered. If this bit is set to 1, the CPU performs whatever data transfers are indicated by the extended instruction opcode, and assumes that the EPU is present to execute the instruction.

System Stack Overflow Warning (S) Bit. This is the enable bit for the System Stack Overflow Warning trap. While it is set to 1, Stack Overflow Warning traps can occur during a stack access while in system mode, as determined by the contents of the Stack Limit register. While this bit is cleared to 0, Stack Overflow Warning traps are disabled. This bit is automatically cleared when a System Stack Overflow Warning trap is generated.

The Trap Control register is cleared to all zeros by a reset, indicating that I/O instructions are not privileged, EPUs are not present in the system, and Stack Overflow Warning traps are disabled. Bits 3 through 7 of this register are not used.

3.3.6 System Stack Limit Register

The 16-bit System Stack Limit register determines when a System Stack Overflow Warning trap is to be generated. Pushes onto the system-mode stack cause the 12 most significant bits of the logical address of the System Stack Pointer to be compared to the 12 most significant bits of this register; a System Stack Overflow Warning trap is generated if they match. The low-order four bits of this register must be zeros (Figure 3-10). This register has no effect on MPU operation if the System Stack Overflow Warning enable bit in the Trap Control register is cleared to 0.

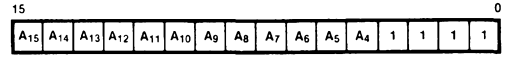


Figure 3-10. System Stack Limit Register

The contents of the System Stack Limit register are cleared to zeros by a reset.

Chapter 4.
Addressing Modes and Data Types

4.1 INTRODUCTION

An instruction is a consecutive list of one or more bytes in memory. Most instructions act upon some data; the term operand refers to the data to be operated upon. For Z280 CPU instructions, operands can reside in CPU registers, memory locations, or I/O ports. The methods used to designate the location of the operands for an instruction are called addressing modes. The Z280 CPU supports nine addressing modes: Register, Immediate, Indirect Register, Direct Address, Indexed, Short Index, Program Counter Relative Address, Stack Pointer Relative, and Base Index. A wide variety of data types can be accessed using these addressing modes.

4.2 ADDRESSING MODE DESCRIPTIONS

The following pages contain descriptions of the addressing modes for the Z280 CPU. Each description explains how the operand's location is calculated, indicates which address spaces can be accessed with that particular addressing mode, and gives an example of an instruction using that mode, illustrating the assembly language format for the addressing mode. The examples using memory addresses use logical memory addresses; if the MMU is enabled, these logical addresses can be translated to physical addresses before the physical memory is accessed, but this process is not discussed or illustrated here.

4.2.1 Register (R, RX)

When this addressing mode is used, the instruction processes data taken from one of the 8-bit registers A, B, C, D, E, H, L, IXH, IXL, IYH, IYL, or one of the 16-bit registers BC, DE, HL, IX, IY, SP, or one of the special byte registers I or R.

Storing data in a register allows shorter instructions and faster execution than occur with instructions that access memory.

THE OPERAND VALUE IS THE CONTENTS OF THE REGISTER.

The operand is always in the register address space. The register length (byte or word) is specified by the instruction opcode.

Example of R mode:

```
LD BC,HL          ;load the contents of HL into BC
```

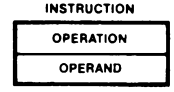
Before instruction execution: *After instruction execution:*

BC:	A	6	B	8	BC:	9	A	2	0
HL:	9	A	2	0	HL:	9	A	2	0

4.2.2 Immediate (IM)

When the Immediate addressing mode is used, the data processed is in the instruction.

The Immediate addressing mode is the only mode that does not indicate a register or memory address as the source operand.



THE OPERAND VALUE IS IN THE INSTRUCTION.

Because an immediate operand is part of the instruction, it is always located in the program memory address space. Immediate mode is often used to initialize registers.

Example of IM mode:

```
LD A,55H          ;load hex 55 into the accumulator
```

Before instruction execution: *After instruction execution:*

A:	6	7			A:	5	5		

4.2.3 Indirect Register (IR)

In the Indirect Register addressing mode, the register specified in the instruction holds the address of the operand. The data to be processed is at the location specified by the HL register for memory accesses or the C register for I/O and control register space accesses. For the Load Byte instruction, BC and DE can also be used in addition to HL.

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS IN THE REGISTER.

Depending on the instruction, the operand specified by IR mode is located in either the I/O address space (I/O instructions), control register space (Load Control instruction), or data memory address space (all other instructions).

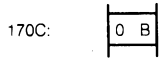
The Indirect Register mode can save space and reduce execution time when consecutive locations are referenced or one location is repeatedly accessed. This mode can also be used to simulate more complex addressing modes, since addresses can be computed before the data is accessed.

Example of IR mode:

```
LD A,(HL)       ;load the accumulator with the data
                ;addressed by the contents of HL
```

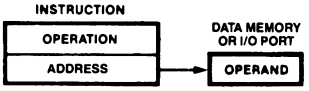
Before instruction execution: *After instruction execution:*

Data memory:



4.2.4 Direct Address (DA)

When the Direct Address addressing mode is used, the data processed is at the location whose memory or I/O port address is in the instruction.



THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS IN THE INSTRUCTION.

Depending on the instruction, the operand specified by DA mode is either in the I/O address space (I/O instructions) or in the data memory address space (all other instructions).

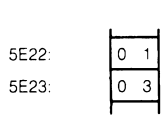
This mode is also used by Jump and Call instructions to specify the address of the next instruction to be executed. (Actually, the address serves as an immediate value that is loaded into the Program Counter.)

Example of DA mode:

```
LD BC,(5E22H)       ;load BC with the data in
                    ;address 5E22
```

Before instruction execution: *After instruction execution:*

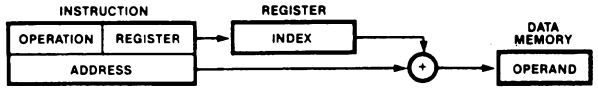
Data memory:



4.2.5 Indexed (X)

For this addressing mode, the data processed is at the location whose address is the address in the instruction offset by the contents of HL, IX, or IY.

The indexed address is computed by adding the address specified in the instruction to a twos-complement "index" contained in the HL, IX or IY register, also specified by the instruction. Indexed addressing allows random access to tables or other complex data structures where the address of the base of the table is known, but the particular element index must be computed by the program.



THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE ADDRESS IN THE INSTRUCTION PLUS THE CONTENTS OF THE REGISTER.

Operands specified by X mode are always in the data memory address space.

Example of X mode:

```
LD A,(IX + 231AH)     ;load into the accumulator
                      ;the contents of the memory
                      ;location whose address
                      ;is 231AH + the value in IX
```

Address calculation:

```
  231A
+01FE
  2518
```

Before instruction execution:

A: 2 3
IX: 0 1 F E

After instruction execution:

A: 3 D
IX: 0 1 F E

Data memory:

2518: 3 D

4.2.6 Short Index (SX)

When the Short Index addressing mode is used, the data processed is at the location whose address is the contents of IX or IY offset by an 8-bit signed displacement in the instruction. (Note that this addressing mode was called "Indexed" in the Z80 CPU literature.)

The short indexed address is computed by adding the 8-bit twos-complement signed displacement specified in the instruction to the contents of the IX or IY register, also specified by the instruction. Short Index addressing allows random access to tables or other complex data structures where the address of the base of the table is known, but the particular element index must be computed by the program.

THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE ADDRESS IN THE INSTRUCTION, OFFSET BY THE CONTENTS OF THE REGISTER.

Operands specified by SX mode are always in the data memory address space.

Example of SX mode:

```
LD A,(IX - 1)     ;load into the accumulator the
                  ;contents of the memory location
                  ;whose address is one less than
                  ;the contents of IX
```

Before instruction execution:

A: 0 1
IX: 2 0 3 A

After instruction execution:

A: 3 D
IX: 2 0 3 A

Data memory:

2039: 3 D

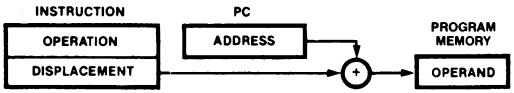
Address calculation: FF encoding in the instruction is sign-extended before the address calculation.

```
  203A
+FFFF
  2039
```

4.2.7 Program Counter (PC) Relative Address (RA)

For Program Counter Relative Addressing mode, the data processed is at the location whose address is the contents of the Program Counter offset by an 8- or 16-bit displacement given in the instruction.

The instruction specifies a twos-complement signed displacement that is added to the Program Counter to form the target address. Except for extended instructions, the Program Counter value used is the address of the first instruction following the currently executing instruction. For extended instructions, the address used to calculate the displacement is the address of the template.



THE OPERAND VALUE IS THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS THE CONTENTS OF PC OFFSET BY THE DISPLACEMENT IN THE INSTRUCTION.

An operand specified by RA mode is always in the program memory address space.

The Program Counter Relative Addressing mode is used by certain program control instructions to specify the address of the next instruction to be executed (specifically, the result of the addition of the Program Counter value and the displacement is loaded into the Program Counter). Relative addressing allows references forward or backward from the current Program Counter value; it is used for program control instructions such as Jumps and for Loads that access constants in the program address space.

Example of RA mode:

```
LD A,<LABEL>      ;load the accumulator with the
                  ;contents of the memory location
                  ;whose address is LABEL
```

This format implies that the assembler will calculate the displacement from the current PC value to the specified label. Alternatively, slightly different syntaxes can be used for the RA mode if the actual displacement from the instruction using this mode is known. Thus, this example can also be written in the following manner:

```
LD A,<$ + 6>      ;load the accumulator with the
                  ;contents of the memory location
                  ;whose address is six more than
                  ;the address of the start of this
                  ;LD instruction
```

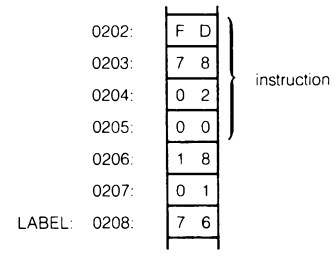
or

```
LD A,(PC + 2)     ;load the accumulator with the
                  ;contents of the memory location
                  ;whose address is two more than
                  ;the current PC, which now points
                  ;to the next instruction
```

Because the Program Counter is advanced to point to the next instruction when the address calculation is performed, the constant that occurs in the instruction is +2.

Before instruction execution: *After instruction execution:*

Program memory:



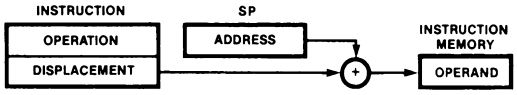
Address calculation:

```
  0206
+    2
  0208
```

4.2.8 Stack Pointer Relative (SR)

For the Stack Pointer Relative addressing mode, the data processed is at the location whose address is the contents of the Stack Pointer offset by a 16-bit displacement in the instruction.

The instruction specifies a twos-complement displacement that is added to the contents of the Stack Pointer register to form the address. An operand specified by SR mode is always in the data memory address space.



The SR addressing mode is used to specify data items to be found in the stack such as parameters passed to subroutines. The System Stack Pointer or User Stack Pointer is selected depending on the state of the User/System bit in the Master Status register.

Example of SR mode:

```
LD A,(SP +2)        ;load into the accumulator
                    ;the contents of the memory
                    ;location whose address is
                    ;two more than the contents
                    ;of SP
```

Before instruction execution:

A: 6 9
SP: 8 2 0 0

After instruction execution:

A: F 3
SP: 8 2 0 0

Data memory:

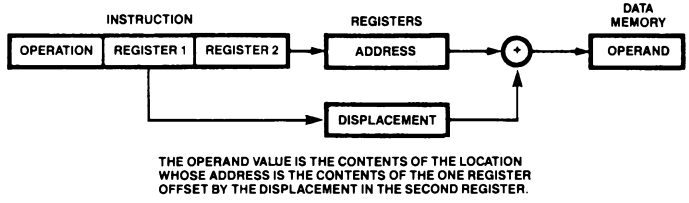
Top of stack	8200:	A B
	8201:	0 1
	8202:	F 3
	8203:	2 8

Address calculation:

```
  8200
+    2
  8202
```

4.2.9 Base Index (BX)

For the Base Index addressing mode, the data processed is at the location whose address is the contents of HL, IX, or IY, offset by the contents of another of these three registers.



This mode allows access to memory locations whose physical addresses are computed at run time and are not fully known at assembly time. An operand specified by BX mode is always in the data memory address space.

Example of BX mode:

```
LD A,(HL + IX)      ;load into the accumulator the
                    ;contents of the memory location
                    ;whose address is the sum of the
                    ;contents of the HL and IX
                    ;register
```

Before instruction execution:

A: B C
HL: 1 5 0 2
IX: F F F E

After instruction execution:

A: A 2
HL: 1 5 0 2
IX: F F F F

Data memory:

1500: A 2

Address calculation:

```
  1502
+FFFE
  1500
```

4.3 DATA TYPES

Many data types are supported by the Z280 MPU architecture; that is, many data types have a hardware representation in a Z280 MPU system and instructions that directly apply to them. The Z280 MPU supports operations on bytes, words, bits, BCD digits, and byte strings.

The basic data type is a byte, which is also the basic addressable element in the register, memory, and I/O address spaces. The 8-bit load, arithmetic, logical, shift, and rotate instructions operate on bytes in registers or memory. Bytes can be treated as logical, signed numeric, or unsigned numeric values.

Operations on two-byte words are also supported. Sixteen-bit load and arithmetic instructions operate on words in registers or memory; words can be treated as signed or unsigned numeric values. I/O reads and writes can be 8-bit or 16-bit operations. Sixteen-bit logical memory addresses can be held and manipulated in 16-bit registers.

Bits are fully supported and addressed by number within a byte (see Figure 2-2). Bits within byte registers or byte memory locations can be tested, set, or cleared.

Operations on binary-coded decimal (BCD) digits are supported by the Decimal Adjust Accumulator and Rotate Digit instructions. BCD digits are stored in byte registers or memory locations, two per byte. The Decimal Adjust Accumulator instruction is used after a binary addition or subtraction of BCD numbers. The Rotate Digit instructions are used to shift BCD digit strings in memory.

Strings of up to 65,536 bytes can be manipulated by the Z280 CPU's block move, block search, and block I/O instructions. The block move instructions allow strings of bytes in memory to be moved from one location to another. Block search instructions provide for scanning strings of bytes in memory to locate a particular value. The block I/O instructions allow strings of bytes or words to be transferred between memory and a peripheral device.

Arrays are supported by the Indexed, Short Index, and Base Index addressing modes. Stacks are supported by those same modes and the Stack Pointer Relative addressing mode, and by special instructions such as Call, Return, Push, and Pop. A special stack write warning feature aids in the allocation of system stack memory space.

Strings of up to 16 bytes can be transferred between memory and an Extended Processing Unit (EPU) during execution of an extended instruction.

Chapter 5.
Instruction Set

5.1 INTRODUCTION

The Z280 CPU's instruction set is a superset of the Z80's; the Z280 CPU is opcode compatible with the Z80 CPU. Thus, a Z80 program can be executed on a Z280 MPU without modification. The instruction set is divided into ten groups by function:

- 8-bit load
- 16-bit load and exchange
- Block transfer and search
- 8-bit arithmetic and logical
- 16-bit arithmetic
- Rotate, shift, and bit manipulation
- Program control
- Input/Output
- CPU control
- Extended instructions

This chapter describes the instruction set of the Z280 CPUs. First, flags and condition codes are discussed in relation to the instruction set. Then, interruptibility of instructions is discussed and traps are described. The last part of this chapter is a detailed description of each instruction, listed in alphabetic order by mnemonic. This section is intended to be used as a reference for Z280 MPU programmers. The entry for each instruction contains a complete description of the instruction, including addressing modes, assembly language mnemonics, instruction opcode formats, and simple examples illustrating the use of the instruction.

5.2 PROCESSOR FLAGS

The Flag register contains six bits of status information that are set or cleared by CPU operations (Figure 5-1). Four of these bits are testable (C, P/V, Z, and S) for use with conditional jump, call, or return instructions. Two flags are not testable (H, N) and are used for binary-coded decimal (BCD) arithmetic.

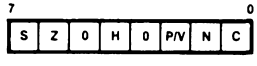


Figure 5-1. Flag Register

The flags provide a link between sequentially executed instructions, in that the result of executing one instruction may alter the flags, and the resulting value of the flags can be used to determine the operation of a subsequent instruction. The program control instructions whose operation depends on the state of the flags are the Jump, Jump Relative, subroutine Call, and subroutine Return instructions; these instructions are referred to as conditional instructions.

5.2.1 Carry Flag (C)

The Carry flag is set or cleared depending on the operation being performed. For add instructions that generate a carry and subtract instructions that generate a borrow, the Carry flag is set to 1. The Carry flag is cleared to 0 by an add that does not generate a carry or a subtract that generates no borrow. This saved carry facilitates software routines for extended precision arithmetic. The multiply and divide instructions use the Carry flag to signal information about the precision of the result. Also, the Decimal Adjust Accumulator instruction leaves the Carry flag set to 1 if a carry occurs when adding BCD quantities.

For the rotate instructions, the Carry flag is used as a link between the least significant and most significant bits for any register or memory location. During shift instructions, the Carry flag contains the last value shifted out of any register or memory location. For logical instructions the Carry flag is cleared. The Carry flag can also be set and complemented with explicit instructions.

5.2.2 Add/Subtract Flag (N)

The Add/Subtract flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag is used to record whether an add or subtract was last executed, allowing a subsequent Decimal Adjust Accumulator instruction to perform correctly. See the discussion of the DAA instruction for further information.

5.2.3 Parity/Overflow Flag (P/V)

This flag is set to a particular state depending on the operation being performed.

For signed arithmetic, this flag, when set to 1, indicates that the result of an operation on twos-complement numbers has exceeded the largest number, or is less than the smallest number, that can be represented using twos-complement notation. This overflow condition can be determined by examining the sign bits of the operands and the result.

The P/V flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of bits set to 1 in a byte are counted. If the total is odd, odd parity (P = 0) is flagged. If the total is even, even parity is flagged (P = 1).

During block search and block transfer instructions, the P/V flag monitors the state of the byte count register (BC). When decrementing the byte counter results in a zero value, the flag is cleared to 0, otherwise the flag is set to 1.

During the Load Accumulator with I or R register instructions, the P/V flag is loaded with the contents of the Interrupt A enable bit in the Master Status register.

When inputting a byte to a register from an I/O device addressed by the C register, the flag is adjusted to indicate the parity of the data.

5.2.4 Half-Carry Flag (H)

The Half-Carry flag (H) is set to 1 or cleared to 0 depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation and between bits 11 and 12 of a 16-bit arithmetic operation. This flag is used by the Decimal Adjust Accumulator instruction to correct the result of an addition or subtraction operation on packed BCD data.

5.2.5 Zero Flag (Z)

The Zero flag (Z) is set to 1 if the result generated by the execution of certain instructions is a zero.

For arithmetic and logical operations, the Zero flag is set to 1 if the result is zero. If the result is not zero, the Zero flag is cleared to 0.

For the block search instructions, the Zero flag is set to 1 if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Zero flag contains the complemented state of the tested bit (i.e., the Zero flag is set to 1 if the tested bit is a 0, and vice-versa).

For the block I/O instructions, if the result of decrementing B is zero, the Zero flag is set to 1; otherwise, it is cleared to 0. Also for byte inputs to registers from I/O devices addressed by the C register, the Zero flag is set to 1 to indicate a zero byte input.

5.2.6 Sign Flag (S)

The Sign flag (S) stores the state of the most significant bit of the result. When the Z280 CPU performs arithmetic operations on signed numbers, binary twos-complement notation is used to represent and process numeric information. A positive number is identified by a zero in the most significant bit. A negative number is identified by a 1 in the most significant bit.

When inputting a byte from an I/O device addressed by the C register to a CPU register, the Sign flag indicates either positive (S = 0) or negative (S = 1) data.

For the Test and Set instruction, the Sign bit is set to 1 if the tested bit is 1, otherwise it is cleared to 0.

5.2.7 Condition Codes

The Carry, Zero, Sign, and Parity/Overflow flags are used to control the operation of the conditional instructions. The operation of these instructions is a function of the state of one of the flags. Special mnemonics called condition codes are used to specify the flag setting to be tested during execution of a conditional instruction; the condition codes are encoded into a 3-bit field in the instruction opcode itself.

Table 5-1 lists the condition code mnemonic, the flag setting it represents, and the binary encoding for each condition code.

Table 5-1. Condition Codes

Mnemonic	Meaning	Flag Setting	Binary Code
Condition Codes for Jump, Call, and Return Instructions			
NZ	Not Zero	Z = 0	000
Z	Zero	Z = 1	001
NC	No Carry	C = 0	010
C	Carry	C = 1	011
NV	No Overflow	V = 0	100
PO	Parity Odd	V = 0	100
V	Overflow	V = 1	101
PE	Parity Even	V = 1	101
NS	No Sign	S = 0	110
P	Plus	S = 0	110
S	Sign	S = 1	111
M	Minus	S = 1	111
Condition Codes for Jump Relative Instruction			
NZ	Not Zero	Z = 0	100
Z	Zero	Z = 1	101
NC	No Carry	C = 0	110
C	Carry	C = 1	111

5.3 INSTRUCTION EXECUTION AND EXCEPTIONS

Two types of exception conditions, interrupts and traps, can alter the normal flow of program execution. Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Traps are synchronous events generated internally in the CPU by particular conditions that occur during instruction execution. Interrupts and traps are discussed in detail in Chapter 6. This section examines the relationship between instructions and the exception conditions.

5.3.1 Instruction Execution and Interrupts

When the CPU receives an interrupt request, and it is enabled for interrupts of that class, the interrupt is normally processed at the end of the current instruction. However, the block transfer and search instructions are designed to be interruptible so as to minimize the length of time it takes the CPU to respond to an interrupt. If an interrupt request is received during a block move, block search, or block I/O instruction, the instruction is suspended after the current iteration. The address of the instruction itself, rather than the address of the following instruction, is saved on the system stack, so that the same instruction is executed again when the interrupt handler executes an interrupt return instruction. The contents of the repetition counter and the registers that index into the block operands are such that, after each iteration, when the instruction is reissued upon returning from an interrupt, the effect is the same as if the instruction were not interrupted. This assumes, of course, that the interrupt handler preserved the registers.

5.3.2 Instruction Execution and Traps

Traps are synchronous events that result from the execution of an instruction. The action of the CPU in response to a trap condition is similar to the case of an interrupt in interrupt mode 3 (see Chapter 6). All traps except for Extended Instruction, System Stack Overflow Warning, Single Step and Breakpoint-on-Halt are nonmaskable.

The Z280 MPU supports eight kinds of traps:

- Division Exception
- Extended Instruction
- Privileged Instruction
- System Call
- Access Violation (page fault and write protect)
- System Stack Overflow Warning
- Single Step
- Breakpoint-on-Halt

The Division Exception trap occurs when executing a divide instruction if either the divisor is zero or the result cannot be represented in the destination (overflow).

The Extended Instruction trap occurs when an extended instruction is encountered, but the Extended Processor Architecture is disabled, (the EPA bit in the Trap Control register should be cleared to 0 if there is no EPU in the system or if the Z280 MPU is configured with an 8-bit bus). This allows the same software to be run on Z280 MPU system configurations with or without Extended Processing Units (EPUs). For systems without EPUs, the desired extended instructions can be emulated by software that is invoked by the Extended Instruction trap. For systems with an 8-bit data bus that also have an EPU, the software invoked by the Extended Instruction trap can use I/O instructions to access the EPU. The information saved on the system stack during this trap is designed to facilitate the 8-bit I/O interface to an EPU by providing address calculation for the operands and by pushing addresses onto the system stack in the reverse order from which they will be used by an I/O interface trap handler.

The Privileged Instruction trap serves to protect the integrity of a system from erroneous or unauthorized actions of user mode processes. Certain instructions, called privileged instructions, can be executed only in system mode. An attempt to execute one of these instructions in user mode causes a Privileged Instruction trap.

The System Call instruction always causes a trap. This instruction is used to transfer control to system mode software in a controlled way, typically to request operating system services.

The Access Violation trap occurs whenever the Z280 MPU's on-chip MMU detects an illegal memory access. Access Violation traps cause instructions to be aborted. When Access Violation traps occur, the logical address of the instruction is pushed onto the system stack along with the Master Status register; part of the logical address that caused the page fault is latched in the MMU to indicate which page frame caused the fault; and the CPU registers are unmodified, i.e., their contents are the same as just before the instruction execution began. (For block move, block search, or block I/O instructions, the registers are the same as just before the iteration in which the page fault occurred.)

The System Stack Overflow Warning trap arises when pushing information onto the system stack causes the Stack Pointer to reference a specified 16-byte area of memory. Use of this facility protects the system from system stack overflow errors.

The Single Step trap occurs with the execution of each instruction, provided the Single-Step control bit in the Master Status register is set to 1. This facilitates software debugging of programs.

The Breakpoint-on-Halt trap occurs whenever the Halt instruction is encountered and the Breakpoint-on-Halt control bit in the Master Status register is set to 1. This facilitates software debugging of programs.

5.4 INSTRUCTION SET FUNCTIONAL GROUPS

This section presents an overview of the Z280 instruction set, arranged by functional groups. (See Section 5.5 for an explanation of the notation used in Tables 5-2 through 5-11.)

5.4.1 8-Bit Load Group

This group of instructions (Table 5-2) includes load instructions for transferring data between byte registers, transferring data between a byte register and memory, and loading immediate data into byte registers or memory. All addressing modes are supported for loading between the accumulator and memory or for loading immediate values into memory. Loads between other registers and memory use the IR and SX addressing modes. An exchange instruction is available for swapping the contents of the accumulator with another register or with memory.

The LDUD and LDUP instructions are available for loading to or from the user-mode memory address space while executing in system mode. The CPU flags are used to indicate if the transfer was successfully completed. LDUD and LDUP are privileged instructions. The other instructions in this group do not affect the flags, nor can their execution cause exception conditions.

Table 5-2. 8-Bit Load Group Instructions

		Addressing Modes Available									
Instruction Name	**Format**	**R**	**RX**	**IM**	**IR**	**DA**	**X**	**SX**	**RA**	**SR**	**BX**
Exchange Accumulator	EX A,src	•	•		•	•	•	•	•	•	•
Exchange H,L	EX H,L										
Load Accumulator	LD A,src	•	•	•	•	•	•	•	•	•	•
	LD dst,A	•	•		•	•	•	•	•	•	•
Load Immediate	LD dst,n	•	•		•	•	•	•	•	•	•
Load Register (Byte)	LD R,src	•	•	•	•			•			
	LD dst,R	•	•		•			•			
Load in User Data Space	LDUD A,src				•			•			
	LDUD dst,A				•			•			
Load in User Program Space	LDUP A,src				•			•			
	LDUP dst,A				•			•			

5.4.2 16-Bit Load and Exchange Group

This group of load and exchange instructions (Table 5-3) allows words of data (two bytes equal one word) to be transferred between registers and memory. The exchange instructions allow for switching between the primary and alternate register files, exchanging the contents of two 16-bit registers, or exchanging the contents of an addressing register with the top word on the stack. The 16-bit loads include transfers between registers and memory and immediate loads of registers or memory. The Load Address instruction facilitates the loading of the address registers with a calculated address. The Push and Pop stack instructions are also included in this group. None of these instructions affect the CPU flags, except for EX AF, AF'. The Push instruction can cause a System Stack Overflow Warning trap; otherwise, no exceptions can arise from the execution of these instructions.

Table 5-3. 16-Bit Load and Exchange Group Instructions

Instruction Name	Format	Addressing Modes Available								
		R	IM	IR	DA	X	SX	RA	SR	BX
Exchange HL with Addressing Register	EX DE,HL									
	EX XY,HL									
Exchange Addressing Register with Top of Stack	EX (SP),XX									
Exchange Accumulator/Flag with Alternate Bank	EX AF,AF'									
Exchange Byte/Word Registers with Alternate Bank	EXX									
Load Addressing Register	LD XX,src		•		•	•		•	•	•
	LD dst,XX				•	•		•	•	•
Load Register (Word)	LD RR,src		•	•	•		•			
	LD dst,RR			•	•		•			
Load Immediate Word	LD dst,nn	•		•	•			•		
Load Stack Pointer	LD SP,src	*	•	•	•		•			
	LD dst,SP			•	•		•			
Load Address	LDA XX,src				•	•		•	•	•
Pop	POP dst	•		•	•			•		
Push	PUSH src	•	•	•	•			•		

*Restricted to an addressing register (HL, IX, or IY).

5.4.3 Block Transfer and Search Group

This group of instructions (Table 5-4) supports block transfer and string search functions. Using these instructions, a block of up to 65,536 bytes can be moved in memory or a byte string can be searched until a given value is found. All the operations can proceed through the data in either direction. Furthermore, the operations can be repeated automatically while decrementing a length counter until it reaches zero, or they can operate on one storage unit per execution with the length counter decremented by one and the source and destination pointer registers properly adjusted. The latter form is useful for implementing more complex operations in software by adding other instructions within a loop containing the block instructions.

Various Z280 MPU registers are dedicated to specific functions for these instructions: the BC register for a counter, the DE and HL registers for memory pointers, and the accumulator for holding the byte value being sought. The repetitive forms of these instructions are interruptible; this is essential since the repetition count can be as high as 65,536. The instruction can be interrupted after any iteration, in which case the address of the instruction itself, rather than the next one, is saved on the system stack; the contents of the operand pointer registers, as well as the repetition counter, are such that the instruction can simply be reissued after returning from the interrupt without any visible difference in the instruction execution.

Table 5-4. Block Transfer and Search Group

Instruction Name	Format
Compare and Decrement	CPD
Compare, Decrement and Repeat	CPDR
Compare and Increment	CPI
Compare, Increment and Repeat	CPIR
Load and Decrement	LDD
Load, Decrement and Repeat	LDDR
Load and Increment	LDI
Load, Increment,and Repeat	LDIR

5.4.4 8-Bit Arithmetic and Logic Group

This group of instructions (Table 5-5) performs 8-bit arithmetic and logical operations. The Add, Add with Carry, Subtract, Subtract with Carry, And, Or, Exclusive Or, Compare, and signed and unsigned Multiply take one input operand from the accumulator and the other from a register, from immediate data in the instruction itself, or from memory. All memory addressing modes are supported: Indirect Register, Short Index, Direct Address,PC Relative Address, Stack Pointer Relative, Indexed, and Base Index. Except for the multiplies, which return the 16-bit result to the HL register, these instructions return the computed result to the accumulator. Both signed and unsigned division are provided. All memory addressing modes except Indirect Register can be used to specify the divisor.

The Increment and Decrement instructions operate on data in a register or in memory; all memory addressing modes are supported. Three instructions operate only on the accumulator: Decimal Adjust, Complement, and Negate. The final instruction in this group, Extend Sign, takes its 8-bit input from the accumulator and returns its 16-bit result to the HL register.

All these instructions except Extend Sign set the CPU flags according to the computed result. Only the Divide instructions can generate an exception.

Table 5-5. 8-Bit Arithmetic and Logic Group

		Addressing Modes Available									
Instruction Name	**Format**	**R**	**RX**	**IM**	**IR**	**DA**	**X**	**SX**	**RA**	**SR**	**BX**
Add With Carry (Byte)	ADC A,src	•	•	•	•	•	•	•	•	•	•
Add (Byte)	ADD A,src	•	•	•	•	•	•	•	•	•	•
And	AND A,src	•	•	•	•	•	•	•	•	•	•
Compare (Byte)	CP A,src	•	•	•	•	•	•	•	•	•	•
Complement Accumulator	CPL A										
Decimal Adjust Accumulator	DAA A										
Decrement (Byte)	DEC dst	•	•		•	•	•	•	•	•	•
Divide (Byte)	DIV A,src	•	•	•		•	•	•	•	•	•
Divide Unsigned (Byte)	DIVU A,src	•	•	•		•	•	•	•	•	•
Extend Sign (Byte)	EXTS A										
Increment (Byte)	INC dst	•	•		•	•	•	•	•	•	•
Multiply (Byte)	MULT A,src	•	•	•	•	•	•	•	•	•	•
Multiply Unsigned (Byte)	MULTU A,src	•	•	•	•	•	•	•	•	•	•
Negate Accumulator	NEG A										
Or	OR A,src	•	•	•	•	•	•	•	•	•	•
Subtract With Carry (Byte)	SBC A,src	•	•	•	•	•	•	•	•	•	•
Subtract (Byte)	SUB A,src	•	•	•	•	•	•	•	•	•	•
Exclusive OR	XOR A,src	•	•	•	•	•	•	•	•	•	•

5.4.5 16-Bit Arithmetic Operations

This group of instructions (Table 5-6) provides 16-bit arithmetic operations. The Add, Add with Carry, Subtract with Carry, and Compare instructions take one input operand from an addressing register and the other from a 16-bit register or from the instruction itself; the result is returned to the addressing register. The 16-bit Increment and Decrement instructions operate on data found in a register or in memory; the Indirect Register, Direct Address or PC Relative addressing mode can be used to specify the memory operand. The instruction that adds the contents of the accumulator to an addressing register supports the use of signed byte indices into tables or arrays in memory.

The remaining 16-bit instructions provide general arithmetic capability using the HL register as one of the input operands. The word Add, Subtract, Compare, and signed and unsigned Multiply instructions take one input operand from the HL register and the other from a 16-bit register, from the instruction itself, or from memory using Indexed, Direct Address, or Relative addressing mode. The 32-bit result of a multiply is returned to the DE and HL registers, with the DE register containing the most significant bits. The signed and unsigned divide instructions take a 32-bit dividend in the DE and HL registers (the DE register containing the most significant bits) and a 16-bit divisor from a register, from the instruction, or from memory using the Indexed, Direct Address, or Relative addressing mode. The

16-bit quotient is returned to the HL register and the 16-bit remainder is returned to the DE register. The Extend Sign instruction takes the contents of the HL register and delivers the 32-bit result to the DE and HL registers, with the DE register containing the most significant bits of the result. The Negate HL instruction negates the contents of the HL register.

Except for Increment, Decrement, and Extend Sign, all the instructions in this group set the CPU flags to reflect the computed result. The only instructions that can generate exceptions are the Divide instructions.

Table 5-6. 16-Bit Arithmetic Operation Instructions

Instruction Name	Format	Addressing Modes Available					
		R	IM	IR	DA	X	RA
Add With Carry (Word)	ADC XX,src	•					
Add (Word)	ADD XX,src	•					
Add Accumulator to Addressing Register	ADD XX,A						
Add Word	ADDW HL,src	•	•		•	•	•
Compare (Word)	CPW HL,src	•	•		•	•	•
Decrement (Word)	DECW dst	•		•	•	•	•
Divide (Word)	DIV DEHL,src	•	•		•	•	•
Divide Unsigned (Word)	DIVU DEHL,src	•	•		•	•	•
Extend Sign (Word)	EXTS HL						
Increment (Word)	INCW dst	•		•	•	•	•
Multiply (Word)	MULT HL,src	•	•		•	•	•
Multiply Unsigned (Word)	MULTU HL,src	•	•		•	•	•
Negate HL	NEG HL						
Subtract With Carry (Word)	SBC XX,src	•					
Subtract (Word)	SUBW HL,src	•	•		•	•	•

5.4.6 Bit Manipulation, Rotate and Shift Group

Instructions in this group (Table 5-7) test, set, and reset bits within bytes and rotate and shift byte data one bit position. Bits to be manipulated are specified by a field within the instruction. Rotation can optionally concatenate the Carry flag to the byte to be manipulated. Both left and right shifting is supported. Right shifts can either shift 0 into bit 7 (logical shifts) or can replicate the sign in bits 6 and 7 (arithmetic shifts). The Test and Set instruction is useful in multiprogramming and multiprocessing environments for implementing synchronization mechanisms between processes. All these instructions except Set Bit and Reset Bit set the CPU flags according to the calculated result; the operand can be a register or a memory location specified by the Indirect Register or Short Index addressing modes.

The RLD and RRD instructions are provided for manipulating strings of BCD digits; these rotate 4-bit quantities in memory specified by the indirect register. The low-order four bits of the accumulator are used as a link between rotations of successive bytes.

None of these instructions generate exceptions.

5.4.7 Program Control Group

This group (Table 5-8) consists of the instructions that affect the Program Counter (PC) and thereby control program flow. The CPU registers and memory are not altered except for the Stack Pointer and the stack, which play a significant role in procedures and interrupts. (An exception is Decrement and Jump if Non-Zero [DJNZ], which uses a register as a loop counter.) The flags are also preserved except for the two instructions specifically designed to set and complement the Carry flag.

The Jump (JP) and Jump Relative (JR) instructions provide a conditional transfer of control to a new location if the processor flags satisfy the condition specified in the instruction. Jump Relative is a 2-byte instruction that jumps to any instruction within the range -126 to +129 bytes from the location of this instruction. Most conditional jumps in programs are made to locations only a few bytes away; the Jump Relative instruction exploits this fact to improve code compactness and efficiency.

A special Jump instruction tests whether the primary or auxiliary register file is being used and branches if the auxiliary file is in use. In

Table 5-7. Bit Manipulation, Rotate and Shift Group

Instruction Name	Format	Addressing Modes Available R	IR	SX
Bit Test	BIT dst	•	•	•
Reset Bit	RES dst	•	•	•
Rotate Left	RL dst	•	•	•
Rotate Left Accumulator	RLA			
Rotate Left Circular	RLC dst	•	•	•
Rotate Left Circular (Accumulator)	RLCA			
Rotate Left Digit	RLD		•	
Rotate Right	RR dst	•	•	•
Rotate Right Accumulator	RRA			
Rotate Right Circular	RRC dst	•	•	•
Rotate Right Circular (Accumulator)	RRCA			
Rotate Right Digit	RRD		•	
Set Bit	SET dst	•	•	•
Shift Left Arithmetic	SLA dst	•	•	•
Shift Right Arithmetic	SRA dst	•	•	•
Shift Right Logical	SRL dst	•	•	•
Test and Set	TSET dst	•	•	•

systems that reserve the auxiliary register file for interrupt handlers only (via a software convention), this instruction can be used to decide whether registers must be saved.

Call and Restart are used for calling subroutines; the current contents of the PC are pushed onto the processor stack and the effective address indicated by the instruction is loaded into the PC. The use of a procedure address stack in this manner allows straightforward implementation of nested and recursive procedures. Call, Jump, and Jump Relative can be unconditional or based on the setting of a CPU flag.

Jump and Call instructions are available with the Indirect Register and PC Relative Address modes in addition to the Direct Address mode. These can be useful for implementing complex control structures such as dispatch tables. When using Direct Address mode for a Jump or Call, the operand is used as an immediate value that is loaded into the PC to specify the address of the next instruction to be executed.

The conditional Return instruction is a companion to the Call instruction; if the condition specified in the instruction is satisfied, it loads the PC from the stack and pops the stack.

Table 5-8. Program Control Group Instructions

Instruction Name	Format	Addressing Modes Available IR	DA	RA
Call	CALL cc,dst	•	•	•
Complement Carry Flag	CCF			
Decrement and Jump if Non-Zero	DJNZ dst			•
Jump on Auxiliary Accumulator/Flag	JAF dst			•
Jump on Auxiliary Register File in Use	JAR dst			•
Jump	JP cc,dst	•	•	•
Jump Relative	JR cc,dst			•
Return	RET cc			
Restart	RST p			
System Call	SC nn			
Set Carry Flag	SCF			

A special instruction, Decrement and Jump if Non-Zero (DJNZ), implements the control part of the basic Pascal FOR loop in a one-word instruction.

System Call (SC) is used for controlled access to facilities provided by the operating system. It is implemented identically to a trap or interrupt in interrupt mode 3: the current program status is pushed onto the system stack, and a new program status is loaded from a dedicated part of memory.

5.4.8 Input/Output Instruction Group

This group (Table 5-9) consists of instructions for transferring a byte, a word, or a string of bytes or words between peripheral devices and the CPU registers or memory. Byte I/O port addresses transfer bytes on AD_0-AD_7 only. Thus in a 16-bit data bus environment, 8-bit peripherals must be connected to bus lines AD_0-AD_7. In an 8-bit data bus environment, word I/O instructions to external peripherals should not be used; however, on-chip peripherals can still be accessed by word I/O instructions.

The instructions for transferring a single byte (IN, OUT) can transfer data between any 8-bit CPU register or memory address specified in the instruction and the peripheral port specified by the contents of the C register. The IN instruction sets the CPU flags according to the input data; however, special cases of these instructions, restricted to using the CPU accumulator and Direct Address mode, do not affect the CPU flags. Another variant tests an input port specified by the contents of the C register and sets the CPU flags without modifying CPU registers or memory.

The instructions for transferring a single word (INW, OUTW) can transfer data between the HL register and the peripheral port specified by the contents of the C register. For word I/O, the contents of H appear on AD_0-AD_7 and the contents of L appear as AD_8-AD_{15}. These instructions do not affect the CPU flags.

The remaining instructions in this group form a powerful and complete complement of instructions for transferring blocks of data between I/O ports and memory. The operation of these instructions is very similar to that of the block move instructions described earlier, with the exception that one operand is always an I/O port whose address remains unchanged while the address of the other operand (a memory location) is incremented or decremented. Both byte and word forms of these instructions are available. The automatically repeating forms of these instructions are interruptible.

I/O instructions are not privileged if the Inhibit User I/O bit in the Trap Control register is clear; they can be executed in either system or user mode, so that I/O service routines can execute in user mode. The Memory Management Unit and on-chip peripherals' control and status registers are accessed using the I/O instructions. The contents of the I/O Page register are output on AD_{23}-AD_{16} with the I/O port address and can be used by external decoding to select specific devices. Pages FF and FE are reserved for on-chip I/O and no external bus transaction is generated. I/O devices can be protected from unrestricted access by using the I/O Page register to select among I/O peripherals.

Table 5-9. Input/Output Instruction Group Instructions

Instruction Name	Format
Input	IN dst,(C)
Input Accumulator	IN A,(n)
Input HL	INW HL,(C)
Input and Decrement (Byte)	IND
Input and Decrement (Word)	INDW
Input, Decrement and Repeat (Byte)	INDR
Input, Decrement and Repeat (Word)	INDRW
Input and Increment (Byte)	INI
Input and Increment (Word)	INIW
Input, Increment and Repeat (Byte)	INIR
Input, Increment and Repeat (Word)	INIRW
Output	OUT (C),src
Output Accumulator	OUT (n),A
Output HL	OUTW (C),HL
Output and Decrement (Byte)	OUTD
Output and Decrement (Word)	OUTDW
Output, Decrement and Repeat (Byte)	OTDR
Output, Decrement and Repeat (Word)	OTDRW
Output and Increment (Byte)	OUTI
Output and Increment (Word)	OTIRW
Output, Increment and Repeat (Byte)	OTIR
Output, Increment and Repeat (Word)	OTIRW
Test Input	TSTI (C)

5.4.9 CPU Control Group

The instructions in this group (Table 5-10) act upon the CPU control and status registers or perform other functions that do not fit into any of the other instruction groups. There are three instructions used for returning from an interrupt or trap service routine. Return from Nonmaskable Interrupt (RETN) and Return from Interrupt (RETI)

are used in interrupt modes 0, 1, and 2 to pop the Program Counter from the stack and manipulate the Interrupt Mask register, or to signal a reset to Z8400 Family peripherals. The Return from Interrupt Long (RETIL) instruction pops a 4-byte program status from the System stack, and is used in interrupt mode 3 and trap processing.

Two of these instructions are not privileged: No Operation (NOP) and Purge Cache (PCACHE). The remaining instructions are privileged.

Table 5-10. CPU Control Group

Instruction Name	Format
Disable Interrupt	DI mask
Enable Interrupt	EI mask
Halt	HALT
Interrupt Mode Select	IM p
Load Accumulator From I or R Register	LD A,src
Load I or R Register From Accumulator	LD dst,A
Load Control	LDCTL dst,src
No Operation	NOP
Purge Cache	PCACHE
Return From Interrupt	RETI
Return From Interrupt Long	RETIL
Return From Nonmaskable Interrupt	RETN

5.4.10 Extended Instruction Group

The Z280 MPU architecture contains a powerful mechanism for extending the basic instruction set through the use of external co-processors called Extended Processing Units (EPUs). A group of 22 opcodes is dedicated for the implementation of extended instructions using this facility. The extended instructions (Table 5-11) are intended for use on a 16-bit data bus; thus, this facility is available only on the Z-BUS configuration of the Z280 MPU.

There are four types of extended instructions in the Z280 MPU instruction set: EPU internal operations, data transfers from an EPU to memory, data transfers from memory to an EPU, and data transfers between an EPU and the CPU's accumulator. The extended instructions that access memory can use any of the six basic memory addressing modes (Indexed, Base Index, PC Relative, SP Relative, Indirect Register, and Direct Address). Transfers between the EPU and CPU accumulator are useful when the program must branch based on conditions generated by an EPU operation.

A 4-byte long "template" is embedded in each of the extended instruction opcodes. These templates determine the operation to be performed in the EPU itself. The formats of these templates are described in the following pages. The descriptions are from the point of view of the CPU; that is, only CPU activities are described. The operation of the EPU is implied, but the full specification of the instruction template depends on the implementation of the EPU, and is beyond the scope of this manual. Fields in the template that are ignored by the CPU are indicated by asterisks, and would typically contain opcodes that determine any operation to be performed by the EPU in addition to the data transfers specified by the instruction. A 2-bit identification field is included in each template, for use in selecting one of up to four EPUs in a multiple-EPU system.

The action taken by the CPU upon encountering an extended instruction depends upon the EPA control bit in the CPU's Trap Control register. When this bit is set to 1, indicating that EPUs are included in the system, extended instructions are executed. If this bit is cleared to 0, indicating that there are no EPUs in the system, the CPU executes an extended instruction trap whenever an extended instruction is encountered; this allows a trap service routine to emulate the desired operation in software.

Table 5-11. Extended Instructions

Instruction Name	Format
Load EPU From Memory	EPUM src
Load Memory From EPU	MEPU dst
Load Accumulator From EPU	EPUF
EPU Internal Operation	EPUI

5.5 NOTATION AND BINARY ENCODING

The rest of this chapter consists of detailed descriptions of the Z280 MPU instructions, arranged in alphabetical order by mnemonic. This section describes the notational conventions used in the instruction descriptions and the binary encoding for register fields within instruction's operation codes (opcodes).

The description of each instruction begins on a new page. The instruction mnemonic and name is printed in bold letters at the top of each page to enable the reader to easily locate a desired

description. The assembly language syntax is then given in a single generic form that covers all the variants of the instruction, along with a list of applicable addressing modes. This is followed by a description of the operation performed by the instruction, a listing of all the flags that are affected by the instruction, a listing of exception conditions that may be caused by execution of the instruction, illustrations of the opcodes for all variants of the instruction, and a simple example of the use of the instruction.

The following notation is used throughout the descriptions of the instructions:

(addr)	A direct address
<addr>	An address to be encoded using relative addressing
b	A 3-bit field specifying the position of a bit within a byte
BX	Base Index addressing mode
cc	A condition code specifying whether a flag is set to 1 or cleared to 0
d	An 8-bit signed displacement
DA	Direct Address addressing mode
dd	A 16-bit signed displacement
disp	The displacement calculated from the address in relative addressing
dst	Destination location or contents
IM	Immediate addressing mode
IR	Indirect Register addressing mode
MSR	The Master Status register
n	8-bit immediate data
nn	16-bit immediate data
p	An interrupt mode
PC	The Program Counter
PS	The program status registers (the Program Counter and Master Status register)
R	A single 8-bit register of the set (A,B,C,D,E,H,L); also, R1 and R2 are used when two different registers are specified in the same instruction. (Note that the R register itself is accessed by a single instruction and violates this convention.)
R'	The corresponding 8-bit or 16-bit register in the alternate register file, such as A'
RA	PC Relative Address addressing mode
RR	A 16-bit register of the set (BC,DE, HL,SP); also, RRA and RRB are used when two different registers are specified in the same instruction
RX	A single byte in the IX or IY registers; that is, a register in the set (IXH,IXL, IYH,IYL); also, RXA and RXB are used when two different registers are specified in the same instruction
SP	The current Stack Pointer in use
SR	Stack Pointer Relative addressing mode
src	Source location or contents
SX	Short Index addressing mode
USP	The User Stack Pointer
X	Indexed addressing mode
XX	One of the 16-bit addressing registers HL, IX, or IY; also XXA and XXB are used when two different registers are specified in the same instruction
XY	One of the 16-bit index registers IX or IY

In the binary encoding of the instruction, lower case is used for the corresponding encoding of the assembler syntax.

Brackets ([and]) are used in the assembly language syntax to indicate an optional field. For example, the 16-bit addition instruction for adding word data to the HL register is described as:

```
ADDW [HL,]src
```

This format means the instruction can be written as:

```
ADDW HL,src
  or
ADDW src
```

Assignment of a value is indicated by the symbol "<--". For example,

```
dst <-- dst + src
```

indicates that the source data is added to the destination data and the result is stored in the destination location.

The notation "addr(n)" is used to refer to bit "n" of a given location, for example, dst(7).

The register field in the binary encoding of an instruction opcode is encoded as shown in Table 5-12.

Table 5-12. Encoding of 8-Bit Registers in Instruction Opcodes

Register	Encoding
A	111
B	000
C	001
D	010
E	011
H	100
L	101

The remainder of this chapter consists of the individual descriptions of each Z280 MPU instruction.

ADC
Add with Carry (Byte)

ADC [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A + src + C

The source operand together with the Carry flag is added to the accumulator and the sum is stored in the accumulator. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a carry from bit 3 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
- **N:** Cleared
- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	ADC A,R	10 001 r
RX:	ADC A,RX	11 Φ11 101 | 10 001 rx
IM:	ADC A,n	11 001 110 | n
IR:	ADC A,(HL)	10 001 110
DA:	ADC A,(addr)	11 011 101 | 10 001 111 | addr(low) | addr(high)
X:	ADC A,(XX + dd)	11 111 101 | 10 001 xx | d(low) | d(high)
SX:	ADC A,(XY + d)	11 Φ11 101 | 10 001 110 | d
RA:	ADC A,<addr>	11 111 101 | 10 001 000 | disp(low) | disp(high)
SR:	ADC A,(SP + dd)	11 011 101 | 10 001 000 | d(low) | d(high)
BX:	ADC A,(XXA + XXB)	11 011 101 | 10 001 bx

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: ADC A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxvn1**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**6 1**	**00x1x000**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

ADC
Add With Carry (Word)

ADC dst,src

dst = HL
src = BC, DE, HL, SP
or
dst = IX
src = BC, DE, IX, SP
or
dst = IY
src = BC, DE, IY, SP

Operation: dst ← dst + src + C

The source operand together with the Carry flag is added to the destination and the sum is stored in the destination. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a carry from bit 11 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of the same sign and the result is of the opposite sign; cleared otherwise
- **N:** Cleared
- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise.

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	ADC HL,RR	11 | 101 | 101 | 01 | rr | 010
	ADC XY,RR	11 | Φ11 | 101 | 11 | 101 | 101 | 01 | rr | 010

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 001 for BC, 011 for DE, 101 for add register to itself, 111 for SP

Example: ADC HL,BC

Before instruction execution:

F:		**szxhxvn1**
BC:	**2 3**	**0 8**
HL:	**F 0**	**3 8**

After instruction execution:

F:		**00x0x001**
BC:	**2 3**	**0 8**
HL:	**1 3**	**4 1**

ADD

Add Accumulator to Addressing Register

ADD dst,A dst = HL, IX, IY

Operation: dst ← dst + A

The contents of the accumulator are added to the contents of the destination and the result is stored in the destination. The contents of the accumulator are unaffected. The contents of the accumulator are treated as a signed binary integer and are sign-extended to 16 bits; twos-complement addition is performed.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Set if there is a carry from bit 11 of the result; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if the operands are of the same sign and the result is of the opposite sign from the operands; cleared otherwise
N: Cleared
C: Set if there is a carry from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	ADD HL,A	11 101 101 | 01 101 101
	ADD XY,A	11 Φ11 101 | 11 101 101 | 01 101 101

Field Encoding: Φ: 0 for IX, 1 for IY

Example: ADD HL,A

Before instruction execution:

AF:	**E 2**	**szxhxvnc**
HL:	**2 3**	**8 4**

After instruction execution:

AF:	**E 2**	**00x1x001**
HL:	**2 3**	**6 6**

Computation: accumulator is sign-extended.

```
  FFE2
+ 2384
------
  2366
```

ADD
Add (Byte)

ADD [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A + src

The source operand is added to the accumulator and the sum is stored in the accumulator. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a carry from bit 3 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
- **N:** Cleared
- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	ADD A,R	10 000 r
RX:	ADD A,RX	11 Φ11 101 | 10 000 rx
IM:	ADD A,n	11 000 110 | n
IR:	ADD A,(HL)	10 000 110
DA:	ADD A,(addr)	11 011 101 | 10 000 111 | addr(low) | addr(high)
X:	ADD A,(XX + dd)	11 111 101 | 10 000 xx | d(low) | d(high)
SX:	ADD A,(XY + d)	11 Φ11 101 | 10 000 110 | d
RA:	ADD A,<addr>	11 111 101 | 10 000 000 | disp(low) | disp(high)
SR:	ADD A,(SP + dd)	11 011 101 | 10 000 000 | d(low) | d(high)
BX:	ADD A,(XXA + XXB)	11 011 101 | 10 000 bx

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: ADD A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxvnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**6 0**	**00x1x000**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

ADD dst,src

dst = HL
src = BC, DE, HL, SP
or
dst = IX
src = BC, DE, IX, SP
or
dst = IY
src = BC, DE, IY, SP

Operation: dst ← dst + src

The source operand is added to the destination and the sum is stored in the destination. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:
S: Unaffected
Z: Unaffected
H: Set if there is a carry from bit 11 of the result; cleared otherwise
V: Unaffected
N: Cleared
C: Set if there is a carry from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	ADD HL,RR	00 | rr | 001
	ADD XY,RR	11 | Φ11 | 101 | 00 | rr | 001

Field Encodings:
Φ: 0 for IX, 1 for IY
rr: 001 for BC, 011 for DE, 101 for add register to itself, 111 for SP

Example: ADD HL,BC

Before instruction execution:

F:		**szxhxvnc**
BC:	**2 3**	**0 8**
HL:	**F 0**	**3 8**

After instruction execution:

F:		**szx0xv01**
BC:	**2 3**	**0 8**
HL:	**1 3**	**4 0**

ADDW
Add Word

ADDW [HL,]src src = R, IM, DA, X, RA

Operation: HL ← HL + src

The source operand is added to the HL register and the sum is stored in the HL register. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a carry from bit 11 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
- **N:** Cleared
- **C:** Set if there is a carry from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	ADDW HL,RR	11 101 101 | 11 rr 110
	ADDW HL,XY	11 Φ11 101 | 11 101 101 | 11 100 110
IM:	ADDW HL,nn	11 111 101 | 11 101 101 | 11 110 110 | n(low byte) | n(high byte)
DA:	ADDW HL,(addr)	11 011 101 | 11 101 101 | 11 010 110 | addr(low) | addr(high)
X:	ADDW HL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 110 | d(low) | d(high)
RA:	ADDW HL,<addr>	11 011 101 | 11 101 101 | 11 110 110 | disp(low) | disp(high)
IR:	ADDW HL,(HL)	11 011 101 | 11 101 101 | 11 000 110

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 000 for BC, 010 for DE, 100 for HL, 110 for SP
- **xy:** 000 for (IX + dd), 010 for (IY + dd)

Example: ADDW HL,DE

Before instruction execution:

F:		szxhxvnc
DE:	0 0	1 0
HL:	A 1	2 3

After instruction execution:

F:		10x0x000
DE:	0 0	1 0
HL:	A 1	3 3

AND [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A AND src

A logical AND operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever the corresponding bits in the two operands are both 1s; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:
S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if all bits of the result are zero; cleared otherwise
H: Set
P: Set if the parity is even; cleared otherwise
N: Cleared
C: Cleared

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	AND A,R	10 100 r
RX:	AND A,RX	11 Φ11 101 | 10 100 rx
IM:	AND A,n	11 100 110 | n
IR:	AND A,(HL)	10 100 110
DA:	AND A,(addr)	11 011 101 | 10 100 111 | addr(low) | addr(high)
X:	AND A,(XX + dd)	11 111 101 | 10 100 xx | d(low) | d(high)
SX:	AND A,(XY + d)	11 Φ11 101 | 10 100 110 | d
RA:	AND A,<addr>	11 111 101 | 10 100 000 | disp(low) | disp(high)
SR:	AND A,(SP + dd)	11 011 101 | 10 100 000 | d(low) | d(high)
BX:	AND A,(XXA + XXB)	11 011 101 | 10 100 bx

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: AND A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxpnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**0 8**	**00x1x000**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

BIT
Bit Test

BIT b,dst dst = R, IR, SX

Operation: Z ← NOT dst(b)

The specified bit b within the destination operand is tested, and the Zero flag is set to 1 if the specified bit is zero, otherwise the Zero flag is cleared to 0. The contents of the destination are unaffected. The bit to be tested is specified by a 3-bit field in the instruction; this field contains the binary encoding for the bit number to be tested. The bit number must be between 0 and 7.

Flags:
S: Unaffected
Z: Set if the specified bit is zero; cleared otherwise
H: Set
P: Unaffected
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	BIT b,R	11 001 011 | 01 b r
IR:	BIT b,(HL)	11 001 011 | 01 b 110
SX:	BIT b,(XY + d)	11 Φ11 101 | 11 001 011 | d | 01 b 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: BIT 1,A

Before instruction execution:

AF:	00010110	szxhxpnc

After instruction execution:

AF:	00010110	s0x1xp0c

CALL [cc,]dst dst = IR, DA, RA

Operation: If the cc is satisfied then: SP ← SP − 2
(SP) ← PC
PC ← dst

A conditional call transfers program control to the destination address if the setting of a selected flag satisfies the condition code "cc" specified in the instruction; an unconditional call always transfers control to the destination address. The current contents of the Program Counter (PC) are pushed onto the top of the stack; the PC value used is the address of the first instruction byte following the Call instruction. The destination address is then loaded into the PC and points to the first instruction of the called procedure. At the end of a procedure a return instruction (RET) can be used to return to the original program.

Each of the Zero, Carry, Sign, and Overflow flags can be individually tested and a call performed conditionally on the setting of the flag.

When using DA mode with the CALL instruction, the operand is not enclosed in parentheses.

Flags: No flags affected

Exceptions: System Stack Overflow Warning

Addressing Mode	Syntax	Instruction Format				
IR:	CALL cc,(HL)	11 011 101	11 cc 100			
	CALL (HL)	11 011 101	11 001 101			"unconditional call"
DA:	CALL cc,addr	11 cc 100	addr(low)	addr(high)		
	CALL addr	11 001 101	addr(low)	addr(high)		"unconditional call"
RA:	CALL cc,<addr>	11 111 101	11 cc 100	disp(low)	disp(high)	
	CALL <addr>	11 111 101	11 001 101	disp(low)	disp(high)	"unconditional call"

Field Encoding: **cc :** 000 for NZ, 001 for Z, 010 for NC, 011 for C, 100 for PO or NV, 101 for PE or V, 110 for P or NS, 111 for M or S

Example: CALL 2520H

Before instruction execution:

PC:	1 6	3 0
SP:	F F	2 6

Data memory:

FF24:	0 0
FF25:	0 0

After instruction execution:

PC:	2 5	2 0
SP:	F F	2 4

Data memory:

FF24:	3 3
FF25:	1 6

CCF
Complement Carry Flag

CCF

Operation: C ← NOT C

The Carry flag is inverted.

Flags:
S: Unaffected
Z: Unaffected
H: The previous state of the Carry flag
P: Unaffected
N: Cleared
C: Set if the Carry flag was clear before the operation; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CCF	00 111 111

Example: CCF

Before instruction execution:

F: **szxhxvn0**

After instruction execution:

F: **szx0xv01**

CP [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A – src

The source operand is compared with the accumulator and the flags are set accordingly. The contents of the accumulator and the source are unaffected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 4 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	CP A,R	10 111 r
RX:	CP A,RX	11 Φ11 101 | 10 111 rx
IM:	CP A,n	11 111 110 | n
IR:	CP A,(HL)	10 111 110
DA:	CP A,(addr)	11 011 101 | 10 111 111 | addr(low) | addr(high)
X:	CP A,(XX + dd)	11 111 101 | 10 111 xx | d(low) | d(high)
SX:	CP A,(XY + d)	11 Φ11 101 | 10 111 110 | d
RA:	CP A,<addr>	11 111 101 | 10 111 000 | disp(low) | disp(high)
SR:	CP A,(SP + dd)	11 011 101 | 10 111 000 | d(low) | d(high)
BX:	CP A,(XXA + XXB)	11 011 101 | 10 111 bx

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: CP A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxvnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**4 8**	**00x0x010**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

CPD
Compare and Decrement

CPD

Operation:

A – (HL)
HL ← HL – 1
BC ← BC – 1

This instruction is used for searching strings of byte data. The byte of data at the location addressed by the HL register is compared with the contents of the accumulator and the Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed. Next the HL register is decremented by one, thus moving the pointer to the previous element in the string. The BC register, used as a counter, is then decremented by one.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise
H: Set if there is a borrow from bit 4 of the result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CPD	11 101 101 | 10 101 001

Example:

Before instruction execution:

AF:	**3 B**	**szxhxvnc**
HL:	**1 2**	**1 5**
BC:	**0 0**	**0 1**

Data memory:

1215:	**3 B**

After instruction execution:

AF:	**3 B**	**01x0x01c**
HL:	**1 2**	**1 4**
BC:	**0 0**	**0 0**

Data memory:

1215:	**3 B**

CPDR

Operation: Repeat until BC = 0 or match: A – (HL)
HL ← HL – 1
BC ← BC – 1

This instruction is used for searching strings of byte data. The bytes of data starting at the location addressed by the HL register are compared with the contents of the accumulator until either an exact match is found or the string length is exhausted. The Sign and Zero flags are set to reflect the result of the last comparison. The contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed.

After each comparison, the HL register is decremented by one, thus moving the pointer to the previous element in the string. The BC register, used as a counter, is then decremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are zero at the start of this instruction, a string length of 65,536 bytes is indicated.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:
S: Set if the last result is negative; cleared otherwise
Z: Set if the last result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise
H: Set if there is a borrow from bit 4 of the last result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CPDR	11 101 101 | 10 111 001

Example: CPDR

Before instruction execution:

AF:	**F 3**	**szxhxvnc**
HL:	**1 1**	**1 8**
BC:	**0 0**	**0 7**

After instruction execution:

AF:	**F 3**	**01x0x11c**
HL:	**1 1**	**1 5**
BC:	**0 0**	**0 4**

Data memory:

1116:	**F 3**
1117:	**0 0**
1118:	**5 2**

Data memory:

1116:	**F 3**
1117:	**0 0**
1118:	**5 2**

CPI
Compare and Increment

CPI

Operation:

A – (HL)
HL ← HL + 1
BC ← BC – 1

This instruction is used for searching strings of byte data. The byte of data at the location addressed by the HL register is compared with the contents of the accumulator and the Sign and Zero flags are set to reflect the result of the comparison. The contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed.

Next the HL register is incremented by one, thus moving the pointer to the next element in the string. The BC register, used as a counter, is then decremented by one.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise
H: Set if there is a borrow from bit 4 of the result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CPI	11 101 101 | 10 100 001

Example:

CPI

Before instruction execution:

AF:	**3 B**	**szxhxvnc**
HL:	**1 2**	**1 5**
BC:	**0 0**	**0 1**

Data memory:

1215:	**3 B**

After instruction execution:

AF:	**3 B**	**01x0x01c**
HL:	**1 2**	**1 6**
BC:	**0 0**	**0 0**

Data memory:

1215:	**3 B**

CPIR

Operation: Repeat until BC = 0 or match: A – (HL)
HL ← HL + 1
BC ← BC – 1

This instruction is used for searching strings of byte data. The bytes of data starting at the location addressed by the HL register are compared with the contents of the accumulator until either an exact match is found or the string length is exhausted. The Sign and Zero flags are set to reflect the result of the comparison. The last contents of the accumulator and the memory bytes are unaffected. Twos-complement subtraction is performed.

After each comparison, the HL register is incremented by one, thus moving the pointer to the next element in the string. The BC register, used as a counter, is then decremented by one. If the result of decrementing the BC register is not zero and no match has been found, the process is repeated. If the contents of the BC register are zero at the start of this instruction, a string length of 65,536 bytes is indicated.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:
S: Set if the last result is negative; cleared otherwise
Z: Set if the last result is zero, indicating that the contents of the accumulator and the memory byte are equal; cleared otherwise
H: Set if there is a borrow from bit 4 of the last result; cleared otherwise
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Set
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CPIR	11 101 101 | 10 110 001

Example: CPIR

Before instruction execution:

AF:	**F 3**	**szxhxvnc**
HL:	**1 1**	**1 8**
BC:	**0 0**	**0 7**

Data memory:

1118:	**2 5**
1119:	**0 0**
111A:	**F 3**

After instruction execution:

AF:	**F 3**	**01x0x11c**
HL:	**1 1**	**1 B**
BC:	**0 0**	**0 4**

Data memory:

1118:	**2 5**
1119:	**0 0**
111A:	**F 3**

CPL
Complement Accumulator

CPL [A]

Operation: A ← NOT A

The contents of the accumulator are complemented (ones complement); all 1 bits are changed to 0 and vice-versa.

Flags:
S: Unaffected
Z: Unaffected
H: Set
V: Unaffected
N: Set
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	CPL A	00 101 111

Example: CPL A

Before instruction execution:

AF:	2 8	szxhxvnc

After instruction execution:

AF:	D 7	szx1xv1c

CPW [HL,]src src = R, IM, DA, X, RA

Operation: HL – src

The source operand is compared with the HL register and the flags are set accordingly. The contents of the source and HL are unaffected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 12 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of different signs and the result is the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	CPW HL,RR	11 101 101 | 11 rr 111
	CPW HL,XY	11 Φ11 101 | 11 101 101 | 11 100 111
IM:	CPW HL,nn	11 111 101 | 11 101 101 | 11 110 111 | n(low byte) | n(high byte)
DA:	CPW HL,(addr)	11 011 101 | 11 101 101 | 11 010 111 | addr(low) | addr(high)
X:	CPW HL,(XY + dd)	11 111 101 | 11 101 101 | 11 0Φ0 111 | d(low) | d(high)
RA:	CPW HL,<addr>	11 011 101 | 11 101 101 | 11 110 111 | disp(low) | disp(high)
IR:	CPW HL,(HL)	11 011 101 | 11 101 101 | 11 000 111

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 000 for BC, 010 for DE, 100 for HL, 110 for SP

Example: CPW HL,DE

Before instruction execution:

F:		szxhxvnc
DE:	0 0	1 0
HL:	A 1	2 3

After instruction execution:

F:		10x0x010
DE:	0 0	1 0
HL:	A 1	2 3

DAA
Decimal Adjust Accumulator

DAA

Operation: A ← Decimal Adjust A

The accumulator is adjusted to form two 4-bit BCD digits following a binary, twos-complement addition or subtraction on two BCD-encoded bytes. The table below indicates the operation performed for addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG).

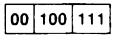
Operation of DAA Instruction

Operation	C Before DAA	Hex Value in Upper Digit (Bits 7-4)	H Before DAA	Hex Value in Lower Digit (Bits 3-0)	Number Added to Byte	C After DAA	H After DAA
	0	0-9	0	0-9	00	0	0
	0	0-8	0	A-F	06	0	1
ADD	0	0-9	1	0-3	06	0	0
ADC	0	A-F	0	0-9	60	1	0
INC	0	9-F	0	A-F	66	1	1
(N = 0)	0	A-F	1	0-3	66	1	0
	1	0-2	0	0-9	60	1	0
	1	0-2	0	A-F	66	1	1
	1	0-3	1	0-3	66	1	0
SUB	0	0-9	0	0-9	00	0	0
SBC	0	0-8	1	6-F	FA	0	1
DEC	1	7-F	0	0-9	A0	1	0
NEG	1	6-F	1	6-F	9A	1	1
(N = 1)							

The operation is undefined if the accumulator was not the result of a binary addition or subtraction of BCD digits.

Flags:
- **S:** Set if the most significant bit of the result is set; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** See table above
- **P:** Set if the parity of the result is even; cleared otherwise
- **N:** Not affected
- **C:** See table above

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	DAA	

Example: DAA

Before instruction execution:

AF:	2 8	szx0xp01

After instruction execution:

AF:	8 8	00x0x001

DEC
Decrement (Byte)

DEC dst dst = R, RX, IR, DA, X, SX, RA, SR, BX

Operation: dst ← dst − 1

The destination operand is decremented by one and the result is stored in the destination. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 4 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the destination was 80_H; cleared otherwise
- **N:** Set
- **C:** Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	DEC R	00 r 101
RX:	DEC RX	11 Φ11 101 | 00 rx 101
IR:	DEC (HL)	00 110 101
DA:	DEC (addr)	11 011 101 | 00 111 101 | addr(low) | addr(high)
X:	DEC (XX + dd)	11 111 101 | 00 xx 101 | d(low) | d(high)
SX:	DEC (XY + d)	11 Φ11 101 | 00 110 101 | d
RA:	DEC <addr>	11 111 101 | 00 000 101 | disp(low) | disp(high)
SR:	DEC (SP + dd)	11 011 101 | 00 000 101 | d(low) | d(high)
BX:	DEC (XXA + XXB)	11 011 101 | 00 bx 101

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: DEC (HL)

Before instruction execution:

F:		**szxhxvnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**8 8**

After instruction execution:

F:		**10x0x01c**
HL:	**2 4**	**5 4**

Data memory:

2454:	**8 7**

DEC[W] dst — dst = R
or
DECW dst — dst = IR, DA, X, RA

Operation: dst ← dst − 1

The destination operand is decremented by one. Twos-complement subtraction is performed.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	DECW RR	00 | rr | 011
	DECW XY	11 | Φ11 | 101 | 00 | 101 | 011
IR:	DECW (HL)	11 | 011 | 101 | 00 | 001 | 011
DA:	DECW (addr)	11 | 011 | 101 | 00 | 011 | 011 | addr(low) | addr(high)
X:	DECW (XY + dd)	11 | 111 | 101 | 00 | xy | 011 | d(low) | d(high)
RA:	DECW <addr>	11 | 011 | 101 | 00 | 111 | 011 | disp(low) | disp(high)

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 001 for BC, 011 for DE, 101 for HL, 111 for SP
- **xy:** 001 for (IX + dd), 011 for (IY + dd)

Example: DECW HL

Before instruction execution:

HL: 2 3 0 8

After instruction execution:

HL: 2 3 0 7

DI
Disable Interrupt

DI mask Mask = Hex value between 0 and $7F_H$

Operation: If mask(i) = 1 then MSR(i) ← 0

The designated interrupt control bits in the Master Status register (MSR) are cleared to 0, thus disabling all interrupts on these inputs; all other interrupt enables in the MSR are unaffected. If no mask is present then all interrupts are disabled.

Any combination of interrupt enables in the MSR can be specified. The seven bits in the mask field in the instruction correspond to the seven interrupt enable bits in the MSR, mask bit i corresponding to MSR bit i.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	DI	11 | 110 | 011
	DI mask	11 | 101 | 101 | 01 | 110 | 111 | mask

Mask = byte specifying which interrupts to disable: mask(i) corresponds to interrupt source i; mask(7) must be zero.

Example: DI 23H

Before instruction execution:

MSR:	0 0	7 F

After instruction execution:

MSR:	0 0	5 C

DIV
Divide (Byte)

DIV [HL,]src src = R, RX, IM, DA, X, SX, RA, SR, BX

Operation:

A ← HL ÷ src
L ← remainder

The contents of the HL register (dividend) are divided by the source operand (divisor) and the quotient is stored in the accumulator; the remainder is stored in the L register. The contents of the source and the H register are unaffected. Both operands are treated as signed, twos-complement integers and division is performed so that the remainder is of the same sign as the dividend.

There are three possible outcomes of the DIV instruction, depending on the division and the resulting quotient:

CASE 1: If the quotient is within the range -2^7 to 2^7-1 inclusive, then the quotient is left in the accumulator, the Overflow flag is cleared to 0, and the Sign and Zero flags are set according to the value of the quotient.

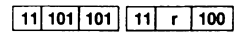
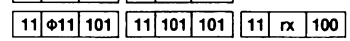
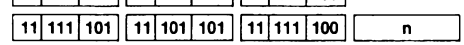
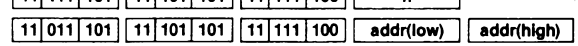
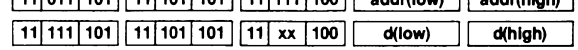
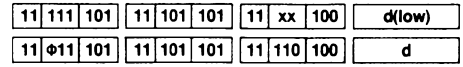
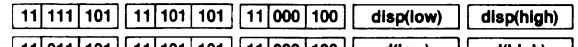
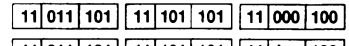
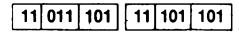
CASE 2: If the divisor is zero, the accumulator remains unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken.

CASE 3: If the quotient is outside the range -2^7 to 2^7-1, the accumulator remains unchanged, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0. Then the Division Exception trap is taken.

Flags:

S: Cleared if V flag is set; else set if the quotient is negative, cleared otherwise
Z: Set if the quotient or divisor is zero; cleared otherwise
H: Unaffected
V: Set if the divisor is zero or if the computed quotient lies outside the range from -2^7 to 2^7-1; cleared otherwise
N: Unaffected
C: Unaffected

Exceptions: Division Exception

Addressing Mode	Syntax	Instruction Format
R:	DIV HL,R	
RX:	DIV HL,RX	
IM:	DIV HL,n	
DA:	DIV HL,(addr)	
X:	DIV HL,(XX + dd)	
SX:	DIV HL,(XY + d)	
RA:	DIV HL,<addr>	
SR:	DIV HL,(SP + dd)	
BX:	DIV HL,(XXA + XXB)	
IR:	DIV HL,(HL)	

Field Encodings:

Φ:	0 for IX, 1 for IY
rx:	100 for high byte, 101 for low byte
xx:	001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx:	001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: DIV HL,C

Before instruction execution:

AF:	**5 5**	**szxhxvnc**
C:		**F E**
HL:	**F F**	**F D**

After instruction execution:

AF:	**0 1**	**00xhx0nc**
C:		**F E**
HL:	**F F**	**F F**

DIVU
Divide Unsigned (Byte)

DIVU [HL,]src src = R, RX, IM, DA, X, SX, RA, SR, BX

Operation:

A ← HL ÷ src
L ← remainder

The contents of the HL register (dividend) are divided by the source operand (divisor) and the quotient is stored in the accumulator; the remainder is stored in the L register. The contents of the source and the H register are not affected. Both operands are treated as unsigned, binary integers.

There are three possible outcomes of the DIVU instruction, depending on the division and the resulting quotient:

CASE 1: If the quotient is less than 2^8, then the quotient is left in the accumulator, the Overflow and Sign flags are cleared to 0 and the Zero flag is set according to the value of the quotient.

CASE 2: If the divisor is zero, the accumulator remains unchanged, the Zero and Overflow flags are set to 1 and the Sign flag is cleared to 0. Then the Division Exception trap is taken.

CASE 3: If the quotient is greater than or equal to 2^8, the accumulator remains unchanged, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0. Then the Division Exception trap is taken.

Flags:

S: Cleared
Z: Set if the quotient or divisor is zero; cleared otherwise
H: Unaffected
V: Set if the divisor is zero or if the computed quotient is greater than or equal to 2^8; cleared otherwise
N: Unaffected
C: Unaffected

Exceptions: Division Exception

Addressing Mode	Syntax	Instruction Format
R:	DIVU HL,R	11 101 101 | 11 r 101
RX:	DIVU HL,RX	11 Φ11 101 | 11 101 101 | 11 rx 101
IM:	DIVU HL,n	11 111 101 | 11 101 101 | 11 111 101 | n
DA:	DIVU HL,(addr)	11 011 101 | 11 101 101 | 11 111 101 | addr(low) | addr(high)
X:	DIVU HL,(XX + dd)	11 111 101 | 11 101 101 | 11 xx 101 | d(low) | d(high)
SX:	DIVU HL,(XY + d)	11 Φ11 101 | 11 101 101 | 11 110 101 | d
RA:	DIVU HL,<addr>	11 111 101 | 11 101 101 | 11 000 101 | disp(low) | disp(high)
SR:	DIVU HL,(SP + dd)	11 011 101 | 11 101 101 | 11 000 101 | d(low) | d(high)
BX:	DIVU HL,(XXA + XXB)	11 011 101 | 11 101 101 | 11 bx 101
IR:	DIVU HL,(HL)	11 101 101 | 11 110 101

Field Encodings:

Φ:	0 for IX, 1 for IY
rx:	100 for high byte, 101 for low byte
xx:	001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx:	001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: DIVU HL,C

Before instruction execution:

AF:	**5 5**	**szxhxvnc**
C:		**0 2**
HL:	**0 1**	**0 1**

After instruction execution:

AF:	**8 0**	**00xhx0nc**
C:		**0 2**
HL:	**0 1**	**0 1**

DIVUW
Divide Unsigned (Word)

DIVUW [DEHL,]src src = R, IM, DA, X, RA

Operation:

HL ← DEHL ÷ src
DE ← remainder

The contents of the DE and HL registers (with the most significant bits of the dividend in the DE register) are divided by the source operand (divisor) and the quotient is stored in the HL register and the remainder in the DE register. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers.

There are three possible outcomes of the DIVUW instruction, depending on the division and the resulting quotient:

CASE 1: If the quotient is less than 2^{16}, then the quotient is left in the HL register and the remainder is left in the DE register, the Overflow and Sign flags are cleared to 0, and the Zero flag is set according to the value of the quotient.

CASE 2: If the divisor is zero, the DE and HL registers remain unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken.

CASE 3: If the quotient is greater than $2^{16} - 1$, then the DE and HL registers remain unchanged, the Overflow flag is set to 1, and the Zero and Sign flags are cleared to 0. Then the Division Exception trap is taken.

Flags:

S: Cleared
Z: Set if the quotient or divisor is zero; cleared otherwise
H: Unaffected
V: Set if the divisor is zero or if the computed quotient is greater than or equal to 2^{16}; cleared otherwise
N: Unaffected
C: Unaffected

Exceptions: Division Exception

Addressing Mode	Syntax	Instruction Format
R:	DIVUW DEHL,RR	11 101 101 | 11 rr 011
	DIVUW DEHL,XY	11 Φ11 101 | 11 101 101 | 11 101 011
IM:	DIVUW DEHL,nn	11 111 101 | 11 101 101 | 11 111 011 | n(low) | n(high)
DA:	DIVUW DEHL,(addr)	11 011 101 | 11 101 101 | 11 011 011 | addr(low) | addr(high)
X:	DIVUW DEHL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 011 | disp(low) | disp(high)
RA:	DIVUW DEHL, <addr>	11 011 101 | 11 101 101 | 11 111 011 | disp(low) | disp(high)
IR:	DIVUW DEHL,(HL)	11 011 101 | 11 101 101 | 11 001 011

Field Encodings:

Φ: 0 for IX, 1 for IY
rr: 001 for BC, 011 for DE, 101 for HL, 111 for SP
xy: 001 for (IX + dd), 011 for (IY + dd)

Example: DIVUW DEHL,6

Before instruction execution:

F:		szxhxvnc
DE:	0 0	0 0
HL:	0 0	2 2

After instruction execution:

F:		00xhx0nc
DE:	0 0	0 4
HL:	0 0	0 5

DIVW
Divide (Word)

DIVW [DEHL,]src src = R, IM, DA, X, RA

Operation:

HL ← DEHL ÷ src
DE ← remainder

The contents of the DE and HL registers (with the DE register containing the most significant bits of the dividend) are divided by the source operand (divisor) and the quotient is stored in the HL register. The contents of the source are unaffected. Both operands are treated as signed, twos-complement integers and division is performed so that the remainder is of the same sign as the dividend.

There are three possible outcomes of the DIVW instruction, depending on the division and the resulting quotient:

CASE 1: If the quotient is within the range -2^{15} to $2^{15}-1$ inclusive, then the quotient is left in the HL register and the remainder is left in the DE register, the Overflow flag is cleared to 0, and the Sign and Zero flags are set according to the value of the quotient.

CASE 2: If the divisor is zero, the DE and HL registers remain unchanged, the Zero and Overflow flags are set to 1, and the Sign flag is cleared to 0. Then the Division Exception trap is taken.

CASE 3: If the quotient is outside the range -2^{15} to $2^{15}-1$, the DE and HL registers remain unchanged, the Overflow flag is set to 1, and the Sign and Zero flags are cleared to 0. Then the Division Exception trap is taken.

Flags:

S: Cleared if V flag is set; else set if the quotient is negative, cleared otherwise
Z: Set if the quotient or divisor is zero; cleared otherwise
H: Unaffected
V: Set if the divisor is zero or if the computed quotient lies outside the range from -2^{15} to $2^{15}-1$; cleared otherwise
N: Unaffected
C: Unaffected

Exceptions: Division Exception

Addressing Mode	Syntax	Instruction Format
R:	DIVW DEHL,RR	11 101 101 | 11 rr 010
	DIVW DEHL,XY	11 Φ11 101 | 11 101 101 | 11 101 010
IM:	DIVW DEHL,nn	11 111 101 | 11 101 101 | 11 111 010 | n(low) | n(high)
DA:	DIVW DEHL,(addr)	11 011 101 | 11 101 101 | 11 011 010 | addr(low) | addr(high)
X:	DIVW DEHL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 010 | d(low) | d(high)
RA:	DIVW DEHL,<addr>	11 011 101 | 11 101 101 | 11 111 010 | disp(low) | disp(high)
IR:	DIVW DEHL,(HL)	11 011 101 | 11 101 101 | 11 001 010

Field Encodings:

Φ: 0 for IX, 1 for IY
rr: 001 for BC, 011 for DE, 101 for HL, 111 for SP
xy: 001 for (IX + dd), 011 for (IY + dd)

Example: DIVW DEHL,6

Before instruction execution:

F:		szxhxvnc
DE:	0 0	0 0
HL:	0 0	2 2

After instruction execution:

F:		00xhx0nc
DE:	0 0	0 4
HL:	0 0	0 5

DJNZ
Decrement and Jump if Non-Zero

DJNZ dst dst = RA

Operation:

B ← B − 1
if B ≠ 0 then PC ← dst

The B register is decremented by one. If the result is non-zero, then the destination address is calculated and then loaded into the Program Counter (PC). Control then passes to the instruction whose address is pointed to by the PC. When the B register reaches zero, control falls through to the instruction following DJNZ. This instruction provides a simple method of loop control.

The destination address is calculated using Relative addressing. The displacement in the instruction is added to the PC; the PC value used is the address of the instruction following the DJNZ instruction. The 8-bit displacement is treated as a signed, twos-complement integer. Thus the branching range from the location of this instruction is −126 to +129 bytes.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
RA:	DJNZ addr	00 | 010 | 000 | disp

Example: DJNZ 1050H

Before instruction execution:

B:		1 2
PC:	1 0	7 6

After instruction execution:

B:		1 1
PC:	1 0	5 0

EI
Enable Interrupt

EI mask Mask = Hex value between 0 and $7F_H$

Operation: If mask(i) = 1 then MSR(i) ← 1

The designated control bits in the Master Status register (MSR) are set to 1, thus enabling interrupts on these inputs; all other interrupt enables in the MSR are unaffected. Note that during the execution of this instruction and the following instruction, all maskable interrupts (whether previously enabled or not) are automatically disabled for the duration of these two instructions.

Any combination of interrupt enables in the MSR can be specified. The seven bits in the mask field in the instruction correspond to the seven interrupt enable bits in the MSR, mask bit i corresponding to MSR bit i. If no mask is present, all interrupts are enabled.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	EI	11 111 011
	EI mask	11 101 101 | 01 111 111 | mask

Mask = byte specifying which interrupts to disable: mask(i) corresponds to interrupt source i; mask(7) must be zero.

Example: EI 49H

Before instruction execution:

MSR:	0 0	0 0

After instruction execution:

MSR:	0 0	4 9

EX AF,AF′

Operation: AF ↔ AF′

The control bit mapping the accumulator and flag registers into the primary bank or the auxiliary bank is complemented, thus effectively exchanging the accumulator and flag registers between the two banks.

Flags: Loaded from F′

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EX AF,AF′	00 001 000

Example: EX AF,AF′

Before instruction execution:

AF:	**2**	**3**	**F**	**3**
AF′:	**1**	**0**	**B**	**0**

After instruction execution:

AF:	**1**	**0**	**B**	**0**
AF′:	**2**	**3**	**F**	**3**

EX

Exchange Addressing Register with Top of Stack

EX (SP),dst dst = HL, IX, IY

Operation: (SP) ↔ dst

The contents of the destination register are exchanged with the contents of the top of stack. That is, the low-order byte contained in the register is exchanged with the contents of the memory address specified by the Stack Pointer (SP), and the high-order byte of the register is exchanged with the contents of the next highest memory address (SP + 1).

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EX (SP),HL	11 100 011
	EX (SP),XY	11 Φ11 101 | 11 100 011

Field Encoding: Φ: 0 for IX, 1 for IY

Example: EX (SP),HL

Before instruction execution:

HL:	2	1	9	3
SP:	8	2	0	0

Data memory:

8200:	2	A
8201:	B	3

After instruction execution:

HL:	B	3	2	A
SP:	8	2	0	0

Data memory:

8200:	9	3
8201:	2	1

EX H,L

Operation: H ↔ L

The contents of the H and L registers are exchanged.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EX H,L	11 101 101 | 11 101 111

Example: EX H,L

Before instruction execution:

HL: 1 2 3 4

After instruction execution:

HL: 3 4 1 2

EX

Exchange HL with Addressing Register

EX src,HL src = DE, IX, IY

Operation: src ↔ HL

The contents of the HL register are exchanged with the contents of the source.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EX DE,HL	11 101 011
	EX XY,HL	11 Φ11 101 11 101 011

Field Encoding: Φ: 0 for IX, 1 for IY

Example: EX DE,HL

Before instruction execution:

DE:	8	2	E	0
HL:	3	8	F	F

After instruction execution:

DE:	3	8	F	F
HL:	8	2	E	0

EX
Exchange with Accumulator

EX A,src src = R, RX, IR, DA, X, SX, RA, SR, BX

Operation: src ↔ A

The contents of the accumulator are exchanged with the contents of the source.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	EX A,R	11 101 101 | 00 r 111
RX:	EX A,RX	11 Φ11 101 | 11 101 101 | 00 rx 111
IR:	EX A,(HL)	11 101 101 | 00 110 111
DA:	EX A,(addr)	11 011 101 | 11 101 101 | 00 111 111 | addr(low) | addr(high)
X:	EX A,(XX + dd)	11 111 101 | 11 101 101 | 00 xx 111 | d(low) | d(high)
SX:	EX A,(XY + d)	11 Φ11 101 | 11 101 101 | 00 110 111 | d
RA:	EX A,<addr>	11 111 101 | 11 101 101 | 00 000 111 | disp(low) | disp(high)
SR:	EX A,(SP + dd)	11 011 101 | 11 101 101 | 00 000 111 | d(low) | d(high)
BX:	EX A,(XXA + XXB)	11 011 101 | 11 101 101 | 00 bx 111

Field Encodings:

- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: EX A,B

Before instruction execution:

A:	0 3
B:	8 2

After instruction execution:

A:	8 2
B:	0 3

EXTS
Extend Sign (Byte)

EXTS [A]

Operation:

L ← A
If A(7) = 0, then H ← 00 else H ← FF

The contents of the accumulator, considered as a signed, twos-complement integer, are sign-extended to 16 bits and the result is stored in the HL register. The contents of the accumulator are unaffected. This instruction is useful for conversion of short signed operands to longer signed operands.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EXTS A	11 101 101 01 100 100

Example: EXTS A

Before instruction execution:

A:		8 2
HL:	5 5	5 5

After instruction execution:

A:		8 2
HL:	F F	8 2

EXTS HL

Operation: If H(7) = 0, then DE ← 0000 else DE ← FFFF

The contents of the HL register, considered as a signed, twos-complement integer, are sign-extended to 32 bits and the result is stored in the DE and HL registers, with the DE register containing the most significant bits. This instruction is useful for conversion of signed operands to larger signed operands.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EXTS HL	11 101 101 | 01 101 100

Example: EXTS HL

Before instruction execution:

DE:	0 3	2 F
HL:	E F	3 0

After instruction execution:

DE:	F F	F F
HL:	E F	3 0

EXX

Exchange Byte/Word Registers with Alternate Bank

EXX

Operation:

BC ↔ BC′
DE ↔ DE′
HL ↔ HL′

The control bit mapping the byte/word registers into the primary or auxiliary bank of the CPU registers is complemented, thus effectively exchanging the B, C, D, E, H, and L registers between the two banks.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	EXX	11 | 011 | 001

Example: EXX

Before instruction execution:

BC:	23A0
DE:	1653
HL:	24FF
BC′:	380F
DE′:	E200
HL′:	1FA3

After instruction execution:

BC:	380F
DE:	E200
HL:	1FA3
BC′:	23A0
DE′:	1653
HL′:	24FF

HALT

Operation: CPU Halts

The CPU operation is suspended until an interrupt or reset request is received. This instruction is used to synchronize the Z280 MPU with external events, preserving its state until an interrupt or reset request is accepted. After an interrupt is serviced, the instruction following HALT is executed. While halted, memory refresh cycles still occur, and bus requests are honored.

For the Z80 Bus configuration of the Z280 MPU, the $\overline{\text{HALT}}$ signal is asserted when the Halt instruction is executed and remains asserted until an interrupt or reset request is accepted. For the Z-BUS configurations of the Z280 MPU, a special Halt bus transaction is performed when the halt instruction is executed.

If the Breakpoint-on-Halt control bit in the Master Status register is set to 1, the Halt instruction is not executed, and Breakpoint-on-Halt trap is taken instead.

Flags: No flags affected

Exceptions: Breakpoint, Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	HALT	01 110 110

IM
Interrupt Mode Select

IM p p = 0, 1, 2, 3

Operation: Interrupt Mode ← p

The interrupt mode of operation is set to one of four modes (see Chapter 6 for a description of the various modes for responding to interrupts). The current interrupt mode can be read from the Interrupt Status register.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	IM p	11 101 101 01 t 110

p mode	t encoding
0	000
1	010
2	011
3	001

Example: IM 3

Before instruction execution:

Interrupt Status register:

F 0	0 0

After instruction execution:

Interrupt Status register:

F 3	0 0

IN dst,(C) dst = R, RX, DA, X, RA, SR, BX

Operation: dst ← (C)

The byte of data from the selected peripheral is loaded into the destination. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8-A_{15} and the contents of the I/O Page register are placed on address lines A_{16}-A_{23}. The byte of data from the peripheral is then loaded into the destination.

Flags:
S: Set if the input data is negative; cleared otherwise
Z: Set if the input data is zero; cleared otherwise
H: Cleared
V: Set if the input data has even parity; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
R:	IN R,(C)	11 101 101 | 01 r 000
RX:	IN RX,(C)	11 Φ11 101 | 11 101 101 | 01 rx 000
DA:	IN (addr),(C)	11 011 101 | 11 101 101 | 01 111 000 | addr(low) | addr(high)
X:	IN (XX + dd),(C)	11 111 101 | 11 101 101 | 01 xx 000 | d(low) | d(high)
RA:	IN <addr>,(C)	11 111 101 | 11 101 101 | 01 000 000 | disp(low) | disp(high)
SR:	IN (SP + dd),(C)	11 011 101 | 11 101 101 | 01 000 000 | d(low) | d(high)
BX:	IN (XXA + XXB),(C)	11 011 101 | 11 101 101 | 01 bx 000

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: IN L,(C)

Before instruction execution:

F:		szxhxvnc
BC:	1 6	5 0
HL:	0 0	2 3

After instruction execution:

F:		00x0x00c
BC:	1 6	5 0
HL:	0 0	7 6

I/O Page register:

1 1

Byte 76_H available at I/O port 111650_H

IN
Input Accumulator

IN A,(n)

Operation: A ← (n)

The byte of data from the selected peripheral is loaded into the accumulator. During the I/O transaction, the 8-bit peripheral address from the instruction is placed on the low byte of the address bus, the contents of the accumulator are placed on address lines A_8–A_{15} and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte of data from the selected port is written into the accumulator.

Flags: No flags affected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	IN A,(n)	11 011 011 | n

Example: IN A,(66H)

Before instruction execution:

A: 4 2

After instruction execution:

A: F D

I/O Page register:

1 1

Byte FD_H available at I/O port 114266_H

INC dst dst = R, RX, IR, DA, X, SX, RA, SR, BX

Operation: dst ← dst + 1

The destination operand is incremented by one and the sum is stored in the destination. Twos-complement addition is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a carry from bit 3 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the destination was $7F_H$; cleared otherwise
- **N:** Cleared
- **C:** Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	INC R	00 r 100
RX:	INC RX	11 Φ11 101 | 00 rx 100
IR:	INC (HL)	00 110 100
DA:	INC (addr)	11 011 101 | 00 111 100 | addr(low) | addr(high)
X:	INC (XX + dd)	11 111 101 | 00 xx 100 | d(low) | d(high)
SX:	INC (XY + d)	11 Φ11 101 | 00 110 100 | d
RA:	INC <addr>	11 111 101 | 00 000 100 | disp(low) | disp(high)
SR:	INC (SP + dd)	11 011 101 | 00 000 100 | d(low) | d(high)
BX:	INC (XXA + XXB)	11 011 101 | 00 bx 100

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: INC (HL)

Before instruction execution:

F:	szxhxvnc
HL: 2 4	5 4

Data memory:

2454:	8 8

After instruction execution:

F:	10x0x00c
HL: 2 4	5 4

Data memory:

2454:	8 9

INC[W]
Increment (Word)

INC[W] dst dst = R
or
INCW dst dst = IR, DA, X, RA

Operation: dst ← dst + 1

The destination operand is incremented by one. Twos-complement addition is performed.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	INCW RR	00 rr 011
	INCW XY	11 Φ11 101 | 00 100 011
IR:	INCW (HL)	11 011 101 | 00 000 011
DA:	INCW (addr)	11 011 101 | 00 010 011 | addr(low) | addr(high)
X:	INCW (XY + dd)	11 111 101 | 00 xy 011 | d(low) | d(high)
RA:	INCW <addr>	11 011 101 | 00 110 011 | disp(low) | disp(high)

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 000 for BC, 010 for DE, 100 for HL, 110 for SP
- **xy:** 000 for (IX + dd), 010 for (IY + dd)

Example: INCW BC

Before instruction execution:

BC:	3 F	1 2

After instruction execution:

BC:	3 F	1 3

IND
INDW

Operation:

(HL) ← (C)
B ← B − 1
HL ← AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the selected peripheral is then loaded into the memory location addressed by the HL register. The HL register is then decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The B register, used as a counter, is then decremented by one.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	IND	11 101 101 | 10 101 010
	INDW	11 101 101 | 10 001 010

Example: INDW

Before instruction execution:

F:		**szxhxvnc**
BC:	**1 5**	**6 4**
HL:	**5 0**	**0 2**

After instruction execution:

F:		**s0xhxv1c**
BC:	**1 4**	**6 4**
HL:	**5 0**	**0 0**

I/O Page register:

3 3

Data memory:

5002:	**0 7**
5003:	**8 D**

Word $8D07_H$ available at I/O port 331564_H

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

INDR
Input, Decrement and Repeat (Byte, Word)

INDR
INDRW

Operation:

Repeat until B = 0: (HL) ← (C)
B ← B − 1
HL ← AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. The string of data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and decreasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The B register, used as a counter, is then decremented by one. If the result of decrementing the B register is zero, the instruction is terminated, otherwise the input sequence is repeated. Note that if the B register contains 0 at the start of the execution of this instruction, 256 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	INDR	11 101 101 | 10 111 010
	INDRW	11 101 101 | 10 011 010

Example: INDR

Before instruction execution:

F:		szxhxvnc
BC:	0 3	4 6
HL:	5 2	1 8

After instruction execution:

F:		s1xhxv1c
BC:	0 0	4 6
HL:	5 2	1 5

I/O Page register:

1 7

Data memory:

5216:	F F
5217:	3 B
5218:	9 A

Byte $9A_H$ available at
I/O port 170346_H,
then byte $3B_H$ available at
I/O port 170246_H,
then byte FF_H available at
I/O port 170146_H.

INI
Input and Increment (Byte, Word)

INI
INIW

Operation:

(HL) ← (C)
B ← B − 1
HL ← AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The B register, used as a counter, is then decremented by one.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	INI	11 101 101 10 100 010
	INIW	11 101 101 10 000 010

Example: INI

Before instruction execution:

F:		**szxhxvnc**
BC:	**1 5**	**6 4**
HL:	**5 0**	**0 2**

After instruction execution:

F:		**s0xhxv1c**
BC:	**1 4**	**6 4**
HL:	**5 0**	**0 3**

I/O Page register:

3 3

Data memory:

5002:	**7 A**

Byte $7A_H$ available at
I/O port 331564_H

INIR
INIRW

Operation:

Repeat until B = 0: (HL) ← (C)
B ← B − 1
HL ← AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block input of strings of data. The string of data from the selected peripheral is loaded into memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the selected peripheral is loaded into the memory location addressed by the HL register. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next destination for the input. The B register, used as a counter, is then decremented by one. If the result of decrementing the B register is zero, the instruction is terminated, otherwise the input sequence is repeated. Note that if the B register contains 0 at the start of the execution of this instruction, 256 bytes are input.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value at the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	INIR	11 101 101 | 10 110 010
	INIRW	11 101 101 | 10 010 010

Example: INIRW

Before instruction execution:

F:		szxhxvnc
BC:	0 2	5 5
HL:	4 0	0 2

After instruction execution:

F:		s1xhxv1c
BC:	0 0	5 5
HL:	4 0	0 6

I/O Page register:

3 1

Word $66D7_H$ available at
I/O port 310255_H
then word $A8FF_H$ available
at I/O port 310155_H.

Data memory:

4002:	D 7
4003:	6 6
4004:	F F
4005:	A 8

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

IN[W] HL,(C)

Operation: HL ← (C)

The word of data from the selected peripheral is loaded into the HL register. During the I/O transaction, the 8-bit peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15} and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. Then one word of data from the selected port is written into the HL register. For 8-bit data buses, the contents of L are undefined for external peripherals.

Flags: No flags affected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	IN HL,(C)	11 101 101 10 110 111

Example: INW HL,(C)

Before instruction execution:

BC:	2 6	5 0
HL:	3 3	3 3

After instruction execution:

BC:	2 6	5 0
HL:	8 7	4 D

I/O Page register:

1 0

Word $4D87_H$ available at I/O port 102650_H

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

JAF
Jump On Auxiliary Accumulator/Flag

JAF dst dst = RA

Operation: If auxiliary AF then PC ← dst

A conditional jump is performed if the auxiliary Accumulator/Flag registers are in use. If the jump is taken, the Program Counter is loaded with the destination address; otherwise the instruction following the JAF instruction is executed. This instruction employs an 8-bit signed, twos-complement displacement from the Program Counter to permit jumps within the range −125 to +130 bytes from the location of this instruction.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
RA:	JAF addr	11 011 101 | 00 101 000 | disp

Example: JAF 5000H

Before instruction execution:

Auxiliary Accumulator/Flag in use

PC:	4 F	E 6

After instruction execution:

PC:	5 0	0 0

JAR
Jump On Auxiliary Register File In Use

JAR dst dst = RA

Operation: If auxiliary file then PC ← dst

A conditional jump is performed if the auxiliary register file is in use. If the jump is taken, the Program Counter is loaded with the destination address; otherwise the instruction following the JAR instruction is executed. This instruction employs an 8-bit signed, twos-complement displacement from the Program Counter to permit jumps within the range −125 to +130 bytes from the location of this instruction.

Flags: No flags affected

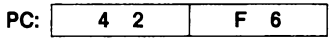
Exceptions: None

Addressing Mode	Syntax	Instruction Format
RA:	JAR addr	

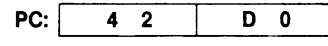
Example: JAR 42D0H

Before instruction execution:

Auxiliary file in use



After instruction execution:



JP
Jump

JP [cc,]dst dst = IR, DA, RA

Operation: If cc is satisfied then PC ← dst

A conditional jump transfers program control to the destination address if the setting of a selected flag satisfies the condition code "cc" specified in the instruction; an unconditional jump always transfers control to the destination address. If the jump is taken, the Program Counter (PC) is loaded with the destination address; otherwise the instruction following the Jump instruction is executed. For the Relative Address mode, the PC value used to calculate the destination address is the address of the next instruction following the Jump instruction; a 16-bit signed twos-complement displacement from the PC permits jumps within the range −32764 to +32771 bytes from the location of this instruction.

Each of the Zero, Carry, Sign, and Overflow flags can be individually tested and a jump performed conditionally on the setting of the flag.

When using DA mode with the JP instruction, the operand is not enclosed in parentheses.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format				
IR:	JP CC,(HL)	11 011 101	11 cc 010			
	JP (HL)	11 101 001				"unconditional jump"
	JP (XY)	11 Φ11 101	11 101 001			"unconditional jump"
DA:	JP CC,addr	11 cc 010	addr(low)	addr(high)		
	JP addr	11 000 011	addr(low)	addr(high)		"unconditional jump"
RA:	JP CC,<addr>	11 111 101	11 cc 010	disp(low)	disp(high)	
	JP <addr>	11 111 101	11 000 011	disp(low)	disp(high)	"unconditional jump"

Field Encodings:

Φ: 0 for IX, 1 for IY

cc: 000 for NZ, 001 for Z, 010 for NC, 011 for C, 100 for PO or NV, 101 for PE or V, 110 for P or NS, 111 for M or S

Example: JP C,5000H

Before instruction execution:

F:		szxhxvn1
PC:	2 6	8 4

After instruction execution:

F:		szxhxvn1
PC:	5 0	0 0

JR [cc,]dst dst = RA

Operation: If the cc is satisfied then PC ← dst

A conditional jump transfers program control to the destination address if the setting of a selected flag satisfies the condition code "cc" specified in the instruction; an unconditional jump always transfers control to the destination address. If the jump is taken, the Program Counter (PC) is loaded with the destination address; otherwise the instruction following the Jump Relative instruction is executed. These instructions employ an 8-bit signed, twos-complement displacement from the PC to permit jumps within the range −126 to +129 bytes from the location of this instruction.

Either the Zero or Carry flag can be tested and a jump performed conditionally on the setting of the flag.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format	
RA:	JR CC,addr	00 | cc | 000 | disp	
	JR addr	00 | 011 | 000 | disp	"unconditional jump"

Field Encoding: **cc:** 100 for NZ, 101 for Z, 110 for NC, 111 for C

Example: JR NZ,6000H

Before instruction execution:

F:		**s0xhxvnc**
PC:	**5 F**	**D 4**

After instruction execution:

F:		**s0xhxvnc**
PC:	**6 0**	**0 0**

LD
Load Accumulator

LD dst,src

dst = R, RX, IR, DA, X, SX, RA, SR, BX
src = A
or
dst = A
src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: dst ← src

The contents of the source are loaded into the destination. The contents of the source are not affected. Special instructions are provided so that the BC and DE registers can also be used in the IR addressing mode.

Flags: No flags affected

Exceptions: None

Load into Accumulator

Addressing Mode	Syntax	Instruction Format
R:	LD A,R	01 111 r
RX:	LD A,RX	11 Φ11 101 / 01 111 rx
IM:	LD A,n	00 111 110 / n
IR:	LD A,(HL)	01 111 110
	LD A,(RR)	00 rra 010
DA:	LD A,(addr)	00 111 010 / addr(low) / addr(high)
X:	LD A,(XX + dd)	11 111 101 / 01 111 xxa / d(low) / d(high)
SX:	LD A,(XY + d)	11 Φ11 101 / 01 111 110 / d
RA:	LD A,<addr>	11 111 101 / 01 111 000 / disp(low) / disp(high)
SR:	LD A,(SP + dd)	11 011 101 / 01 111 000 / d(low) / d(high)
BX:	LD A,(XXA + XXB)	11 011 101 / 01 111 bx

Load from Accumulator

Addressing Mode	Syntax	Instruction Format
R:	LD R,A	01 | r | 111
RX:	LD RX,A	11 | Φ11 | 101 | 01 | rx | 111
IR:	LD (HL),A	01 | 110 | 111
	LD (RR),A	00 | rrb | 010
DA:	LD (addr),A	00 | 110 | 010 | addr(low) | addr(high)
X:	LD (XX + dd),A	11 | 101 | 101 | 00 | xxb | 011 | d(low) | d(high)
SX:	LD (XY + d),A	11 | Φ11 | 101 | 01 | 110 | 111 | d
RA:	LD <addr>,A	11 | 101 | 101 | 00 | 100 | 011 | disp(low) | disp(high)
SR:	LD (SP + dd),A	11 | 101 | 101 | 00 | 000 | 011 | d(low) | d(high)
BX:	LD (XXA + XXB),A	11 | 101 | 101 | 00 | bx | 011

Field Encodings:

- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **rra:** 001 for BC, 011 for DE
- **rrb:** 000 for BC, 010 for DE
- **xxa:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **xxb:** 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Examples: LD A,(HL)

Before instruction execution:

A:	0 F
HL:	1 7 0 C

Data memory:

170C:	0 B

After instruction execution:

A:	0 B
HL:	1 7 0 C

Data memory:

170C:	0 B

LD
Load from I or R Register

LD A,src src = I, R

Operation: A ← src

The contents of the source are loaded into the accumulator. The contents of the source are not affected. The Sign and Zero flags are set according to the value of the data transferred; the Overflow flag is set according to the state of the Interrupt A Enable bit in the Master Status register. Note: The R register does not contain the refresh address and is not modified by refresh transactions.

Flags:
- **S:** Set if the data loaded into the accumulator is negative; cleared otherwise
- **Z:** Set if the data loaded into the accumulator is zero; cleared otherwise
- **H:** Cleared
- **V:** Set when loading the accumulator if the interrupt A Enable bit is set; cleared otherwise
- **N:** Cleared
- **C:** Unaffected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	LD A,I	11 101 101 01 010 111
	LD A,R	11 101 101 01 011 111

Example: LD A,R

Before instruction execution:

AF:	**1 0**	**szxhxvnc**
R:		**4 2**
MSR:	**4 0**	**7 F**

After instruction execution:

AF:	**4 2**	**00x0x10c**
R:		**4 2**
MSR:	**4 0**	**7 F**

LD dst,n dst = R, RX, IR, DA, X, SX, RA, SR, BX

Operation: dst ← n

The byte of immediate data is loaded into the destination.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	LD R,n	00 r 110 | n
RX:	LD RX,n	11 Φ11 101 | 00 rx 110 | n
IR:	LD (HL),n	00 110 110 | n
DA:	LD (addr),n	11 011 101 | 00 111 110 | addr(low) | addr(high) | n
X:	LD (XX + dd),n	11 111 101 | 00 xx 110 | d(low) | d(high) | n
SX:	LD (XY + d),n	11 Φ11 101 | 00 110 110 | d | n
RA:	LD <addr>,n	11 111 101 | 00 000 110 | disp(low) | disp(high) | n
SR:	LD (SP + dd),n	11 011 101 | 00 000 110 | d(low) | d(high) | n
BX:	LD (XXA + XXB),n	11 011 101 | 00 bx 110 | n

Field Encodings:

- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: LD A,55H

Before instruction execution:

A: 6 7

After instruction execution:

A: 5 5

LD
Load Register (Byte)

LD dst,src

dst = R
src = R, RX, IM, IR, SX
or
dst = R, RX, IR, SX
src = R

Operation: dst ← src

The contents of the source are loaded into the destination.

Flags: No flags affected

Exceptions: None

Load into Register

Addressing Mode	Syntax	Instruction Format
R:	LD R1,R2	01 | r1 | r2
RX:	LD R*,RX	11 | Φ11 | 101 | 01 | r* | rx
	LD RXA,RXB	11 | Φ11 | 101 | 01 | rxa | rxb
	LD RX,R*	11 | Φ11 | 101 | 01 | rx | r*
IM:	LD R,n	00 | r | 110 | n
	LD RX,n	11 | Φ11 | 101 | 00 | rx | 110 | n
IR:	LD R,(HL)	01 | r | 110
SX:	LD R,(XY + d)	11 | Φ11 | 101 | 01 | r | 110 | d

Load from Register

IR:	LD (HL),R	01 | 110 | r
SX:	LD (XY + d),R	11 | Φ11 | 101 | 01 | 110 | r | d

Field Encodings:

- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **rxa:** 100 for high byte, 101 for low byte
- **rxb:** 100 for high byte, 101 for low byte
  rxa and rxb refer to the same index register
- **r*:** Only registers A, B, C, D, and E can be accessed
- **r1,r2:** See Table 5-12

Example: LD A,B

Before instruction execution:

A:	0 3
B:	8 2

After instruction execution:

A:	8 2
B:	8 2

LD dst,A dst = I, R

Operation: dst ← A

The contents of the accumulator are loaded into the destination. Note: the R register does not contain the refresh address and is not modified by refresh transactions.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	LD I,A	11 101 101 01 000 111
	LD R,A	11 101 101 01 001 111

Example: LD I,A

Before instruction execution:

A:	0 D
I:	2 2

After instruction execution:

A:	0 D
I:	0 D

LDA
Load Address

LDA dst,src

dst = HL, IX, IY
src = DA, X, RA, SR, BX

Operation: dst ← address(src)

The address of the source operand is computed and loaded into the destination. The contents of the source are not affected. The address translation mechanism in the MMU is not used to determine if the address is valid.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
DA:	LDA HL,(addr)	00 100 001 | addr(low) | addr(high)
	LDA XY,(addr)	11 Φ11 101 | 00 100 001 | addr(low) | addr(high)
X:	LDA HL,(XX + dd)	11 101 101 | 00 xx 010 | d(low) | d(high)
	LDA XY,(XX + dd)	11 Φ11 101 | 11 101 101 | 00 xx 010 | d(low) | d(high)
RA:	LDA HL,<addr>	11 101 101 | 00 100 010 | disp(low) | disp(high)
	LDA XY,<addr>	11 Φ11 101 | 11 101 101 | 00 100 010 | disp(low) | disp(high)
SR:	LDA HL,(SP + dd)	11 101 101 | 00 000 010 | d(low) | d(high)
	LDA XY,(SP + dd)	11 Φ11 101 | 11 101 101 | 00 000 010 | d(low) | d(high)
BX:	LDA HL,(XXA + XXB)	11 101 101 | 00 bx 010
	LDA XY,(XXA + XXB)	11 Φ11 101 | 11 101 101 | 00 bx 010

Field Encodings:

Φ: 0 for IX, 1 for IY
xx: 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: LDA HL,(IX + 4)

Before instruction execution:

HL:	**2 3 0 8**
IX:	**E 3 2 4**

After instruction execution:

HL:	**E 3 2 8**
IX:	**E 3 2 4**

Address calculation:

```
  E324
+    4
------
  E328
```

LDCTL dst,src

dst = (C), USP
src = HL, IX, IY
or
dst = HL, IX, IY
src = (C), USP

Operation: dst ← src

This instruction loads the contents of a CPU control register into an addressing register, or the contents of an addressing register into a CPU control register. The contents of the source are loaded into the destination; the source register is unaffected. The address of the control register is specified by the contents of the C register, with the exception of the User Stack Pointer. The various CPU control registers have the following addresses:

Register	Address (Hexadecimal)
Master Status register (MSR)	00
Interrupt Status register	16
Interrupt/Trap Vector Table Pointer	06
I/O Page register *	08
Bus Timing and Initialization register *	FF
Bus Timing and Control register *	02
Stack Limit register	04
Trap Control register *	10
Cache Control register *	12
Local Address register *	14

* 8-bit control register

When writing to an 8-bit CPU control register, only the low-order byte of the specified source addressing register is written to the control register. When reading from an 8-bit CPU control register, the control register contents are loaded into the low-order byte of the destination addressing register, and the upper byte of the destination is undefined.

Note that the User Stack Pointer control register is accessed using special opcodes; the contents of the C register are not used for these opcodes. This form of the Load Control instruction allows the user-mode Stack Pointer to be accessed while in system-mode operation.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	LDCTL HL,(C)	11 101 101 | 01 100 110
	LDCTL XY,(C)	11 Φ11 101 | 11 101 101 | 01 100 110
	LDCTL (C),HL	11 101 101 | 01 101 110
	LDCTL (C),XY	11 Φ11 101 | 11 101 101 | 01 101 110
	LDCTL HL,USP	11 101 101 | 10 000 111
	LDCTL XY,USP	11 Φ11 101 | 11 101 101 | 10 000 111
	LDCTL USP,HL	11 101 101 | 10 001 111
	LDCTL USP,XY	11 Φ11 101 | 11 101 101 | 10 001 111

Field Encoding: Φ: 0 for IX, 1 for IY

Example: LDCTL (C),HL

Before instruction execution:

C:		0 8
HL:	5 5	3 A

I/O Page register:

0 0

After instruction execution:

C:		0 8
HL:	5 5	3 A

I/O Page register:

3 A

LDD

Operation:

(DE) ← (HL)
DE ← DE – 1
HL ← HL – 1
BC ← BC – 1

This instruction is used for block transfers of strings of data. The byte of data at the location addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then decremented by one, thus moving the pointers to the preceding elements in the string. The BC register, used as a counter, is then decremented by one.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	LDD	11 101 101 | 10 101 000

Example: LDD

Before instruction execution:

F:		**szxhxvnc**
HL:	**1 1**	**1 1**
DE:	**2 2**	**2 2**
BC:	**0 0**	**0 7**

Data memory:

1111:	**8 8**
2222:	**6 6**

After instruction execution:

F:		**szx0x00c**
HL:	**1 1**	**1 0**
DE:	**2 2**	**2 1**
BC:	**0 0**	**0 6**

Data memory:

1111:	**8 8**
2222:	**8 8**

LDDR
Load, Decrement and Repeat

LDDR

Operation: Repeat until BC = 0: (DE) ← (HL)
DE ← DE − 1
HL ← HL − 1
BC ← BC − 1

This instruction is used for block transfers of strings of data. The bytes of data starting at the location addressed by HL are loaded into memory starting at the location addressed by the DE register. The number of bytes moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 bytes are transferred. The effect of decrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a lower memory address. Placing the pointers at the highest address of the strings and decrementing the pointers ensures that the source string is copied without destroying the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:
S: Unaffected
Z: Unaffected
H: Cleared
V: Cleared
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	LDDR	11 101 101 | 10 111 000

Example: LDDR

Before instruction execution:

F:		**szxhxvnc**
HL:	**1 1**	**1 7**
DE:	**2 2**	**2 5**
BC:	**0 0**	**0 3**

Data memory:

1115:	**8 8**
1116:	**3 6**
1117:	**A 5**
2223:	**9 6**
2224:	**1 1**
2225:	**2 6**

After instruction execution:

F:		**szx0x00c**
HL:	**1 1**	**1 4**
DE:	**2 2**	**2 2**
BC:	**0 0**	**0 0**

Data memory:

1115:	**8 8**
1116:	**3 6**
1117:	**A 5**
2223:	**8 8**
2224:	**3 6**
2225:	**A 5**

LDI

Operation:

(DE) ← (HL)
DE ← DE + 1
HL ← HL + 1
BC ← BC − 1

This instruction is used for block transfers of strings of data. The byte of data at the location addressed by the HL register is loaded into the location addressed by the DE register. Both the DE and HL registers are then incremented by one, thus moving the pointers to the next elements in the strings. The BC register, used as a counter, is then decremented by one.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Set if the result of decrementing BC is not equal to zero; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	LDI	11 101 101 | 10 100 000

Example: LDI

Before instruction execution:

F:		**szxhxvnc**
HL:	**1 1**	**1 1**
DE:	**2 2**	**2 2**
BC:	**0 0**	**0 7**

After instruction execution:

F:		**szx0x00c**
HL:	**1 1**	**1 2**
DE:	**2 2**	**2 3**
BC:	**0 0**	**0 6**

Data memory:

1111:	**8 8**
2222:	**6 6**

Data memory:

1111:	**8 8**
2222:	**8 8**

LDIR

Load, Increment and Repeat

LDIR

Operation:

Repeat until BC = 0: (DE) ← (HL)
DE ← DE + 1
HL ← HL + 1
BC ← BC − 1

This instruction is used for block transfers of strings of data. The bytes of data starting at the location addressed by the HL register are loaded into memory starting at the location addressed by the DE register. The number of bytes moved is determined by the contents of the BC register. If the BC register contains zero when this instruction is executed, 65,536 bytes are transferred. The effect of incrementing the pointers during the transfer is important if the source and destination strings overlap with the source string starting at a higher memory address. Placing the pointers at the lowest address of the strings and incrementing the pointers ensures that the source string is copied without destroying the overlapping area.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
V: Cleared
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	LDIR	11 101 101 | 10 110 000

Example: LDIR

Before instruction execution:

F:		szxhxvnc
HL:	1 1	2 5
DE:	2 2	1 0
BC:	0 0	0 3

Data memory:

1125:	5 A
1126:	B 0
1127:	7 6

2210:	F F
2211:	9 A
2212:	2 7

After instruction execution:

F:		szx0x00c
HL:	1 1	2 8
DE:	2 2	1 3
BC:	0 0	0 0

Data memory:

1125:	5 A
1126:	B 0
1127:	7 6

2210:	5 A
2211:	B 0
2212:	7 6

LDUD
Load in User Data Space (Byte)

LDUD dst,src

dst = A
src = IR or SX in user data space
or
dst = IR or SX in user data space
src = A

Operation: dst ← src

The destination is loaded with the contents of the source. In loading from the user data space into the accumulator, the memory-mapping mechanism used in translating logical addresses for data in user mode operation is used to translate the source address. In loading into the user data space from the accumulator, the memory-mapping mechanism used in translating logical addresses for data in user-mode operation is used to translate the destination address. See Chapter 7 for an explanation of this mechanism. The contents of the source are unaffected.

The flags are set to reflect the success or failure of the transfer. If the transfer is unsuccessful, no trap is generated and no information is saved in the MMU. If the transfer is successful, the Carry flag is cleared to 0; if the transfer is unsuccessful, the Carry flag is set to 1. The other flags are unaffected if the transfer is successful. If the transfer is unsuccessful, the value of the Write Protect (WP) bit in the Page Descriptor register used by the MMU is loaded into the Z flag and the value of that Page Descriptor's Valid bit is loaded into the V flag.

Flags:
- **S:** Unaffected
- **Z:** For unsuccessful accesses, loaded with the value of the WP bit used by the MMU; unaffected otherwise
- **H:** Unaffected
- **V:** For unsuccessful accesses, loaded with the value of the Valid bit used by the MMU; unaffected otherwise
- **N:** Unaffected
- **C:** Set if the transfer is unsuccessful; cleared otherwise

Exceptions: Privileged Instruction

Load from User Data Space

Addressing Mode	Syntax	Instruction Format
IR:	LDUD A,(HL)	11 101 101 | 10 000 110
SX:	LDUD A,(XY + d)	11 Φ11 101 | 11 101 101 | 10 000 110 | d

Load into User Data Space

IR:	LDUD (HL),A	11 101 101 | 10 001 110
SX:	LDUD (XY + d),A	11 Φ11 101 | 11 101 101 | 10 001 110 | d

Field Encoding: Φ: 0 for IX, 1 for IY

Example: LDUD A,(HL)

Before instruction execution:

AF:	**0 F**	**szxhxvnc**
HL:	**8 D**	**0 7**

User data memory:

8D07:	**5 5**

After instruction execution:

AF:	**5 5**	**szxhxvn0**
HL:	**8 D**	**0 7**

User data memory:

8D07:	**5 5**

LDUP
Load in User Program Space (Byte)

LDUP dst,src

dst = A
src = IR or SX in user program space
or
dst = IR or SX in user program space
src = A

Operation: dst ← src

The destination is loaded with the contents of the source. In loading from the user program space into the accumulator, the memory-mapping mechanism used in translating logical addresses for program fetches (instructions or data using PC Relative adddressing mode) in user-mode operation is used to translate the source address. When loading into the user program space from the accumulator, the memory-mapping mechanism used in translating logical addresses for program accesses (instructions or data using PC Relative addressing mode) in user-mode operation is used to translate the destination address. See Chapter 7 for an explanation of this mechanism. The contents of the source are unaffected.

The flags are set to reflect the success or failure of the transfer. If the transfer is unsuccessful, no trap is generated and no information is saved in the MMU. If the transfer is successful, the Carry flag is cleared; if the transfer is unsuccessful, the Carry flag is set. The other flags are unaffected if the transfer is successful. If the transfer is unsuccessful, the value of the Write Protect (WP) bit in the Page Descriptor register used by the MMU is loaded into the Z flag and the value of that Page Descriptor's Valid bit is loaded into the V flag.

Flags:
- **S:** Unaffected
- **Z:** For unsuccessful accesses, loaded with the value of the WP bit used by the MMU; unaffected otherwise
- **H:** Unaffected
- **V:** For unsuccessful accesses, loaded with the value of the Valid bit used by the MMU; unaffected otherwise
- **N:** Unaffected
- **C:** Set if the transfer is unsuccessful; cleared otherwise

Exceptions: Privileged Instruction

Load from User Program Space

Addressing Mode	Syntax	Instruction Format
IR:	LDUP A,(HL)	11 101 101 | 10 010 110
SX:	LDUP A,(XY + d)	11 Φ11 101 | 11 101 101 | 10 010 110 | d

Load into User Program Space

Addressing Mode	Syntax	Instruction Format
IR:	LDUP (HL),A	11 101 101 | 10 011 110
SX:	LDUP (XY + d),A	11 Φ11 101 | 11 101 101 | 10 011 110 | d

Field Encoding: Φ: 0 for IX, 1 for IY

Example: LDUP A,(HL)

Before instruction execution:

AF:	**0 F**	**szxhxvnc**
HL:	**5 3**	**9 0**

User program memory:

5390:	**F F**

After instruction execution:

AF:	**F F**	**szxhxvn0**
HL:	**5 3**	**9 0**

User program memory:

5390:	**F F**

LDW
Load Immediate Word

LD[W] dst,nn — dst = R
or
LDW dst,nn — dst = IR, DA, RA

Operation: dst ← nn

The two bytes of immediate data are loaded into the destination. For register destinations, the low byte of the immediate operand is loaded into the low byte of the register and the high byte of the operand is loaded into the high byte of the register. For memory destinations, the low byte of the operand is loaded into the addressed location and the high byte of the operand is loaded into the next higher memory byte (addressed location incremented by one).

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format					
R:	LDW RR,nn	00 rr 001	n(low)	n(high)			
	LDW XY,nn	11 Φ11 101	00 100 001	n(low)	n(high)		
IR:	LDW (HL),nn	11 011 101	00 000 001	n(low)	n(high)		
DA:	LDW (addr),nn	11 011 101	00 010 001	addr(low)	addr(high)	n(low)	n(high)
RA:	LDW <addr>,nn	11 011 101	00 110 001	disp(low)	disp(high)	n(low)	n(high)

Field Encodings:
rr: 000 for BC, 010 for DE, 100 for HL, 110 for SP
Φ: 0 for IX, 1 for IY

Example: LDW (HL),3825H

Before instruction execution:

HL: 2 3 9 1

Data memory:

2391:	1 E
2392:	A 3

After instruction execution:

HL: 2 3 9 1

Data memory:

2391:	2 5
2392:	3 8

LD[W] dst,src

dst = HL, IX, IY
src = IM, DA, X, RA, SR, BX
or
dst = DA, X, RA, SR, BX
src = HL, IX, IY

Operation: dst ← src

The contents of the source are loaded into the destination. The contents of the source are unaffected. For register-to-memory transfers, the effective address of the memory operand corresponds to the low byte of the register and the memory byte at the effective address incremented by one corresponds to the high byte of the register.

Flags: No flags affected

Exceptions: None

Load into Addressing Register

Addressing Mode	Syntax	Instruction Format
IM:	LDW HL,nn	00 100 001 | n(low) | n(high)
	LDW XY,nn	11 Φ11 101 | 00 100 001 | n(low) | n(high)
DA:	LDW HL,(addr)	00 101 010 | addr(low) | addr(high)
	LDW XY,(addr)	11 Φ11 101 | 00 101 010 | addr(low) | addr(high)
X:	LDW HL,(XX + dd)	11 101 101 | 00 xx 100 | d(low) | d(high)
	LDW XY,(XX + dd)	11 Φ11 101 | 11 101 101 | 00 xx 100 | d(low) | d(high)
RA:	LDW HL,<addr>	11 101 101 | 00 100 100 | disp(low) | disp(high)
	LDW XY,<addr>	11 Φ11 101 | 11 101 101 | 00 100 100 | disp(low) | disp(high)
SR:	LDW HL,(SP + dd)	11 101 101 | 00 000 100 | d(low) | d(high)
	LDW XY,(SP + dd)	11 Φ11 101 | 11 101 101 | 00 000 100 | d(low) | d(high)
BX:	LDW HL, (XXA + XXB)	11 101 101 | 00 bx 100
	LDW XY, (XXA + XXB)	11 Φ11 101 | 11 101 101 | 00 bx 100

Load from Addressing Register

Addressing Mode	Syntax	Instruction Format
DA:	LDW (addr),HL	00 100 010 | addr(low) | addr(high)
	LDW (addr),XY	11 Φ11 101 | 00 100 010 | addr(low) | addr(high)
X:	LDW (XX + dd),HL	11 101 101 | 00 xx 101 | d(low) | d(high)
	LDW (XX + dd),XY	11 Φ11 101 | 11 101 101 | 00 xx 101 | d(low)
RA:	LDW <addr>,HL	11 101 101 | 00 100 101 | disp(low) | disp(high)
	LDW <addr>,XY	11 Φ11 101 | 11 101 101 | 00 100 101 | disp(low)
SR:	LDW (SP + dd),HL	11 101 101 | 00 000 101 | d(low) | d(high)
	LDW (SP + dd),XY	11 Φ11 101 | 11 101 101 | 00 000 101 | d(low) | d(high)
BX:	LDW (XXA + XXB), HL	11 101 101 | 00 bx 101
	LDW (XXA + XXB), XY	11 Φ11 101 | 11 101 101 | 00 bx 101

Field Encodings:

Φ: 0 for IX, 1 for IY

xx: 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)

bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: LDW HL,(HL + IX)

Before instruction execution:

HL:	1	5	0	2
IX:	F	F	F	E

Data memory:

1500:	A	2
1501:	0	3

After instruction execution:

HL:	0	3	A	2
IX:	F	F	F	E

Data memory:

1500:	A	2
1501:	0	3

Address calculation:

```
  1502
+ FFFE
------
  1500
```

LD[W] dst,src

dst = BC, DE, HL, SP
src = IM, IR, DA, SX
or
dst = IR, DA, SX
src = BC, DE, HL, SP

Operation: dst ← src

The contents of the source are loaded into the destination. The contents of the source are unaffected. For transfers between a register and memory, the effective address of the memory operand corresponds to the low byte of the register and the memory byte at the effective address incremented by one corresponds to the high byte of the register.

Flags: No flags affected

Exceptions: None

Load into Register

Addressing Mode	Syntax	Instruction Format	
IM:	LDW RR,nn	00 rra 001 | n(low) | n(high)	
IR:	LDW RR,(HL)	11 101 101 | 00 rra 110	
DA:	LDW RR,(addr)	11 101 101 | 01 rrb 011 | addr(low) | addr(high)	(except HL)
SX:	LDW RR,(XY + d)	11 Φ11 101 | 11 101 101 | 00 rra 110 | d	

Load from Register

IR:	LDW (HL),RR	11 101 101 | 00 rrb 110	
DA:	LDW (addr),RR	11 101 101 | 01 rra 011 | addr(low) | addr(high)	(except HL)
SX:	LDW (XY + d),RR	11 Φ11 101 | 11 101 101 | 00 rrb 110 | d	

Field Encodings:
rra: 000 for BC, 010 for DE, 100 for HL, 110 for SP
rrb: 001 for BC, 011 for DE, 101 for HL, 111 for SP
Φ: 0 for IX, 1 for IY

Example: LDW BC,3824H

Before instruction execution:

BC: 2 1 F 3

After instruction execution:

BC: 3 8 2 4

LD[W]
Load Stack Pointer

LD[W] dst,src

dst = SP
src = HL, IX, IY, IM, IR, DA, SX
or
dst = IR, DA, SX
src = SP

Operation: dst ← src

The contents of the source are loaded into the destination, where the source or destination is the Stack Pointer.

Flags: No flags affected

Exceptions: None

Load into Stack Pointer

Addressing Mode	Syntax	Instruction Format
R:	LDW SP,HL	11 111 001
	LDW SP,XY	11 Φ11 101 | 11 111 001
IM:	LDW SP,nn	00 110 001 | n(low) | n(high)
IR:	LDW SP,(HL)	11 101 101 | 00 110 110
DA:	LDW SP,(addr)	11 101 101 | 01 111 011 | addr(low) | addr(high)
SX:	LDW SP,(XY + d)	11 Φ11 101 | 11 101 101 | 00 110 110 | d

Load from Stack Pointer

IR:	LDW (HL),SP	11 101 101 | 00 111 110
DA:	LDW (addr),SP	11 101 101 | 01 110 011 | addr(low) | addr(high)
SX:	LDW (XY + d),SP	11 Φ11 101 | 11 101 101 | 00 111 110 | d

Field Encoding: Φ: 0 for IX, 1 for IY

Example: LDW SP,IX

Before instruction execution:

SP:	2 3 8 D
IX:	F F F 0

After instruction execution:

SP:	F F F 0
IX:	F F F 0

MULT
Multiply (Byte)

MULT [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: HL ← A × src

The contents of the accumulator are multiplied by the source operand and the product is stored in the HL register. The contents of the accumulator and the source are unaffected. Both operands are treated as signed, twos-complement integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the H register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in eight bits and the H register merely holds sign-extension data.

Flags:
S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Unaffected
V: Cleared
N: Unaffected
C: Set if the product is less than -2^7 or greater than or equal to 2^7; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	MULT A,R	11 101 101 | 11 r 000
RX:	MULT A,RX	11 Φ11 101 | 11 101 101 | 11 rx 000
IM:	MULT A,n	11 111 101 | 11 101 101 | 11 111 000 | n
IR:	MULT A,(HL)	11 101 101 | 11 110 000
DA:	MULT A,(addr)	11 011 101 | 11 101 101 | 11 111 000 | addr(low) | addr(high)
X:	MULT A,(XX + dd)	11 111 101 | 11 101 101 | 11 xx 000 | d(low) | d(high)
SX:	MULT A,(XY + d)	11 Φ11 101 | 11 101 101 | 11 110 000 | d
RA:	MULT A,<addr>	11 111 101 | 11 101 101 | 11 000 000 | disp(low) | disp(high)
SR:	MULT A,(SP + dd)	11 011 101 | 11 101 101 | 11 000 000 | d(low) | d(high)
BX:	MULT A,(XXA + XXB)	11 011 101 | 11 101 101 | 11 bx 000

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: MULT A,H

Before instruction execution:

AF:	**F E**	**szxhxvnc**
HL:	**1 2**	**0 0**

After instruction execution:

AF:	**F E**	**10xhx0n0**
HL:	**F F**	**D C**

MULTU
Multiply Unsigned (Byte)

MULTU [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: HL ← A × src

The contents of the accumulator are multiplied by the source operand and the product is stored in the HL register. The contents of the accumulator and the source are unaffected. Both operands are treated as unsigned, binary integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the H register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in eight bits and the H register merely holds zero.

Flags:
S: Cleared
Z: Set if the result is zero; cleared otherwise
H: Unaffected
V: Cleared
N: Unaffected
C: Set if the product is greater than or equal to 2^8; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	MULTU A,R	11 101 101 | 11 r 001
RX:	MULTU A,RX	11 Φ11 101 | 11 101 101 | 11 rx 001
IM:	MULTU A,n	11 111 101 | 11 101 101 | 11 111 001 | n
IR:	MULTU A,(HL)	11 101 101 | 11 110 001
DA:	MULTU A,(addr)	11 011 101 | 11 101 101 | 11 111 001 | addr(low) | addr(high)
X:	MULTU A,(XX + dd)	11 111 101 | 11 101 101 | 11 xx 001 | d(low) | d(high)
SX:	MULTU A,(XY + d)	11 Φ11 101 | 11 101 101 | 11 110 001 | d
RA:	MULTU A,<addr>	11 111 101 | 11 101 101 | 11 000 001 | disp(low) | disp(high)
SR:	MULTU A,(SP + dd)	11 011 101 | 11 101 101 | 11 000 001 | d(low) | d(high)
BX:	MULTU A,(XXA + XXB)	11 011 101 | 11 101 101 | 11 bx 001

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: MULTU A,H

Before instruction execution:

AF:	F E	szxhxvnc
HL:	0 2	F B

After instruction execution:

AF:	F E	00xhx0n1
HL:	0 1	F C

MULTUW
Multiply Unsigned (Word)

MULTUW [HL,]src src = R, IM, DA, X, RA

Operation: DEHL ← HL × src

The contents of the HL register are multiplied by the source operand and the product is stored in the DE and HL registers. The contents of the source are unaffected. Both operands are treated as unsigned, binary integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the DE register is required to represent the result; if the Carry flag is cleared to 0, the product can be represented correctly in 16 bits and the DE register merely holds zero.

Flags:
S: Cleared
Z: Set if the result is zero; cleared otherwise
H: Unaffected
V: Cleared
N: Unaffected
C: Set if the product is greater than or equal to 2^{16}; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	MULTUW HL,RR	11 101 101 | 11 rr 011
	MULTUW HL,XY	11 Φ11 101 | 11 101 101 | 11 100 011
IM:	MULTUW HL,nn	11 111 101 | 11 101 101 | 11 110 011 | n(low) | n(high)
DA:	MULTUW HL,(addr)	11 011 101 | 11 101 101 | 11 010 011 | addr(low) | addr(high)
X:	MULTUW HL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 011 | d(low) | d(high)
RA:	MULTUW HL,<addr>	11 011 101 | 11 101 101 | 11 110 011 | disp(low) | disp(high)
IR:	MULTUW HL,(HL)	11 011 101 | 11 101 101 | 11 000 011

Field Encodings:
Φ: 0 for IX, 1 for IY
rr: 000 for BC, 010 for DE, 100 for HL, 110 for SP
xy: 000 for (IX + dd), 010 for (IY + dd)

Example: MULTUW HL,DE

Before instruction execution:

F:		szxhxvnc
DE:	0 0	0 A
HL:	0 0	3 1

After instruction execution:

F:		00xhx0n0
DE:	0 0	0 0
HL:	0 1	E A

MULTW
Multiply (Word)

MULTW [HL,]src src = R, IM, DA, X, RA

Operation: DEHL ← HL × src

The contents of the HL register are multiplied by the source operand and the product is stored in the DE and HL registers. The contents of the source are unaffected. Both operands are treated as signed, twos-complement integers.

The initial contents of the HL register are overwritten by the result. The Carry flag is set to 1 to indicate that the DE register is required to represent the result; if the Carry flag is cleared to 0, the product can be correctly represented in 16 bits and the DE register merely holds sign-extension data.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Unaffected
- **V:** Cleared
- **N:** Unaffected
- **C:** Set if the product is less than -2^{15} or greater than or equal to 2^{15}; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	MULTW HL,RR	11 101 101 | 11 rr 010
	MULTW HL,XY	11 Φ11 101 | 11 101 101 | 11 100 010
IM:	MULTW HL,nn	11 111 101 | 11 101 101 | 11 110 010 | n(low) | n(high)
DA:	MULTW HL,(addr)	11 011 101 | 11 101 101 | 11 010 010 | addr(low) | addr(high)
X:	MULTW HL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 010 | d(low) | d(high)
RA:	MULTW HL,<addr>	11 011 101 | 11 101 101 | 11 110 010 | disp(low) | disp(high)
IR:	MULTW HL,(HL)	11 011 101 | 11 101 101 | 11 000 010

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 000 for BC, 010 for DE, 100 for HL, 110 for SP
- **xy:** 000 for (IX + dd), 010 for (IY + dd)

Example: MULTW HL,DE

Before instruction execution:

F:		szxhxvnc
DE:	0 0	0 A
HL:	0 0	3 1

After instruction execution:

F:		00xhx0n0
DE:	0 0	0 0
HL:	0 1	E A

NEG [A]

Operation: A ← −A

The contents of the accumulator are negated, that is, replaced by its twos-complement value. Note that 80_H is replaced by itself, because in twos-complement representation the negative number with greatest magnitude has no positive counterpart; for this case, the Overflow flag is set to 1.

Flags:

S: Set if the result is negative, cleared otherwise
Z: Set if the result is zero, cleared otherwise
H: Set if there was a borrow from the least significant bit of the high-order four bits of the result (bit 4); cleared otherwise
V: **Set if the contents of the accumulator was not 80_H before the operation; cleared otherwise.**
N: Set
C: **Set if the contents of the accumulator was not 00_H before the operation; cleared otherwise.**

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	NEG A	11 101 101 | 01 000 100

Example: NEG A

Before instruction execution:

AF:	2 8	szxhxvnc

After instruction execution:

AF:	D 8	10x0x010

NEG
Negate HL

NEG HL

Operation: HL ← − HL

The contents of the HL register are negated, that is, replaced by its twos-complement value. Note that 8000_H is replaced by itself, because in twos-complement representation the negative number with greatest magnitude has no positive counterpart; for this case, the Overflow flag is set to 1.

Flags:

S: Set if the result is negative, cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Set if there was a borrow from the least significant bit of the high-order four bits of the result (bit 12); cleared otherwise
V: Set if the contents of HL was 8000_H before the operation; cleared otherwise
N: Set
C: **Set if the contents of HL was not 000_H before the operation; cleared otherwise.**

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	NEG HL	11 101 101 01 001 100

Example: NEG HL

Before instruction execution:

F:	szxhxvnc
HL: 0 1	2 1

After instruction execution:

F:	10x1x010
HL: F E	D F

NOP

Operation: None

No operation.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	NOP	00 | 000 | 000

OR
OR

OR [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A OR src

A logical OR operation is performed between the corresponding bits in the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever either of the corresponding bits in the two operands is one; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:
S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if all bits of the result are zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Cleared

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	OR A,R	10 110 r
RX:	OR A,RX	11 Φ11 101 | 10 110 rx
IM:	OR A,n	11 110 110 | n
IR:	OR A,(HL)	10 110 110
DA:	OR A,(addr)	11 011 101 | 10 110 111 | addr(low) | addr(high)
X:	OR A,(XX + dd)	11 111 101 | 10 110 xx | d(low) | d(high)
SX:	OR A,(XY + d)	11 Φ11 101 | 10 110 110 | d
RA:	OR A,<addr>	11 111 101 | 10 110 000 | disp(low) | disp(high)
SR:	OR A,(SP + dd)	11 011 101 | 10 110 000 | d(low) | d(high)
BX:	OR A,(XXA + XXB)	11 011 101 | 10 110 bx

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: OR A,(HL)

Before instruction execution:

AF:	4 8	szxhxpnc
HL:	2 4	5 4

Data memory:

2454:	1 8

After instruction execution:

AF:	5 8	00x0x000
HL:	2 4	5 4

Data memory:

2454:	1 8

OTDR
OTDRW

Operation:

Repeat until B = 0: (C) ← (HL)
B ← B − 1
HL ← AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block output of strings of data. The string of data is loaded into the selected peripheral from memory at consecutive addresses, starting with the location addressed by the HL register and decreasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The B register, used as a counter, is decremented by one. The HL register is then decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output. If the result of decrementing the B register is zero, the instruction is terminated, otherwise the output sequence is repeated. Note that if the B register contains 0 at the start of the execution of this instruction, 256 bytes are output.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OTDR	11 101 101 | 10 111 011
	OTDRW	11 101 101 | 10 011 011

Example: OTDR

Before instruction execution:

F:		szxhxvnc
BC:	0 3	4 6
HL:	5 2	1 8

After instruction execution:

F:		s1xhsv1c
BC:	0 0	4 6
HL:	5 2	1 5

I/O Page register:

1 7

Byte $9B_H$ written to I/O port 170346_H,
then byte FF_H written to I/O port 170246_H,
then byte $A3_H$ written to I/O port 170146_H.

Data memory:

5216:	A 3
5217:	F F
5218:	9 B

OTIR
OTIRW

Operation:

Repeat until B = 0: (C) ← (HL)
B ← B − 1
HL ← AUTOINCREMENT (by one if byte, by two if word)

This instruction is used for block output of strings of data. The string of data is loaded into the selected peripheral from memory at consecutive addresses, starting with the location addressed by the HL register and increasing. During the I/O transactions, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The B register, used as a counter, is decremented by one. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output. If the result of decrementing B is zero, the instruction is terminated, otherwise the output sequence is repeated. Note that if the B register contains 0 at the start of the execution of this instruction, 256 bytes are output.

This instruction can be interrupted after each execution of the basic operation. The Program Counter value of the start of this instruction is saved before the interrupt request is accepted, so that the instruction can be properly resumed.

Flags:

S: Unaffected
Z: Set
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OTIR	11 101 101 | 10 110 011
	OTIRW	11 101 101 | 10 010 011

Example: OTIRW

Before instruction execution:

F:		szxhxvnc
BC:	0 2	4 4
HL:	5 0	0 4

After instruction execution:

F:		s1xhxv1c
BC:	0 0	4 4
HL:	5 0	0 8

I/O Page register:

3 1

Word $3A90_H$ written to I/O port 310244_H, then word $B867_H$ written to I/O port 310144_H.

Data memory:

5004:	9 0
5005:	3 A
5006:	6 7
5007:	B 8

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

OUT (C),src src = R, RX, DA, X, RA, SR, BX

Operation: (C) ← src

The byte of data from the source is loaded into the selected peripheral. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte of data from the source is then loaded into the selected peripheral.

Flags: No flags affected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
R:	OUT (C),R	11 101 101 | 01 r 001
RX:	OUT (C),RX	11 Φ11 101 | 11 101 101 | 01 rx 001
DA:	OUT (C),(addr)	11 011 101 | 11 101 101 | 01 111 001 | addr(low) | addr(high)
X:	OUT (C),(XX + dd)	11 111 101 | 11 101 101 | 01 xx 001 | d(low) | d(high)
RA:	OUT (C),<addr>	11 111 101 | 11 101 101 | 01 000 001 | disp(low) | disp(high)
SR:	OUT (C),(SP + dd)	11 011 101 | 11 101 101 | 01 000 001 | d(low) | d(high)
BX:	OUT (C),(XXA + XXB)	11 011 101 | 11 101 101 | 01 bx 001

Field Encodings:

- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: OUT (C),IXH

Before instruction execution:

BC:	1 6	5 0
IX:	F D	0 7

I/O Page register:

3 2

After instruction execution:

Byte FD_H written to I/O port 321650_H

OUT
Output Accumulator

OUT (n),A

Operation: (n) ← A

The contents of the accumulator are loaded into the selected peripheral. During the I/O transaction, the 8-bit peripheral address from the instruction is placed on the low byte of the address bus, the contents of the accumulator are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. Then the contents of the accumulator are written into the selected port.

Flags: No flags affected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OUT (n),A	11 | 010 | 011 | n

Example: OUT (55H),A

Before instruction execution:

A: 4 2

I/O Page register:

1 1

After instruction execution:

Byte 42_H written to I/O port 114255_H

OUTD
OUTDW

Operation:

(C) ← (HL)
B ← B − 1
HL ← AUTODECREMENT HL (by one if byte, by two if word)

This instruction is used for block output of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8-A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}-A_{23}. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The B register, used as a counter, is decremented by one. The HL register is decremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OUTD	11 101 101 | 10 101 011
	OUTDW	11 101 101 | 10 001 011

Example: OUTDW

Before instruction execution:

F:		szxhxvnc
BC:	1 5	6 4
HL:	5 0	0 6

After instruction execution:

F:		s0xhxv1c
BC:	1 4	6 4
HL:	5 0	0 4

I/O Page register:

3 3

Word $8D07_H$ written to I/O port 331564_H

Data memory:

5006:	0 7
5007:	8 D

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

OUTI
OUTIW

Operation:

(C) ← (HL)
B ← B − 1
HL ← AUTOINCREMENT HL (by one if byte, by two if word)

This instruction is used for block output of strings of data. During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte or word of data from the memory location addressed by the HL register is loaded into the selected peripheral. The B register, used as a counter, is decremented by one. The HL register is then incremented by one for byte transfers or by two for word transfers, thus moving the memory pointer to the next source for the output.

Flags:

S: Unaffected
Z: Set if the result of decrementing B is zero; cleared otherwise
H: Unaffected
V: Unaffected
N: Set
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OUTI	11 101 101 | 10 100 011
	OUTIW	11 101 101 | 10 000 011

Example: OUTI

Before instruction execution:

F:		**szxhxvnc**
BC:	**1 5**	**6 4**
HL:	**5 0**	**0 2**

After instruction execution:

F:		**s0xhxv1c**
BC:	**1 4**	**6 4**
HL:	**5 0**	**0 3**

I/O Page register:

3 3

Byte $7B_H$ written to I/O port 331564_H

Data memory:

5002:	**7 B**

OUT[W]
Output HL

OUT[W] (C),HL

Operation: (C) ← HL

The contents of the HL register are loaded into the selected peripheral. During the I/O transaction, the 8-bit peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. Then the contents of the HL register are written into the selected port. For 8-bit data buses, only the contents of the H register are transferred during a single bus transaction.

Flags: No flags affected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	OUTW (C),HL	11 101 101 | 10 111 111

Example: OUTW (C),HL

Before instruction execution:

BC:	2 6	5 0
HL:	3 A	8 4

After instruction execution:

Word $843A_H$ written to I/O port 172650_H

I/O Page register:

1 7

Note: Example assumes that a 16-bit data bus configuration of the Z280 MPU is used.

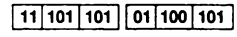
PCACHE

Operation: All cache entries invalidated

This instruction is used to invalidate all entries in the cache.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	PCACHE	

POP
POP

POP dst dst = BC, DE, HL, AF, IX, IY, IR, DA, RA

Operation:

dst ← (SP)
SP ← SP + 2

The content of the memory location addressed by the Stack Pointer (SP) are loaded into the destination. For register destinations, the byte at the memory location specified by the contents of the SP is loaded into the low byte of the destination register (or Flag register for AF) and the byte at the memory location one greater than the contents of the SP is loaded into the high byte of the destination register. The SP is then incremented by two. If the destination is a memory location, the destination and the top of the stack must be non-overlapping.

Flags: No flags affected (unless dst = AF)

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	POP RR	11 | rr | 001
	POP XY	11 | Φ11 | 101 | 11 | 100 | 001
IR:	POP (HL)	11 | 011 | 101 | 11 | 000 | 001
DA:	POP (addr)	11 | 011 | 101 | 11 | 010 | 001 | addr(low) | addr(high)
RA:	POP <addr>	11 | 011 | 101 | 11 | 110 | 001 | disp(low) | disp(high)

Field Encodings:

Φ: 0 for IX, 1 for IY
rr: 000 for BC, 010 for DE, 100 for HL, 110 for AF

Example: POP BC

Before instruction execution:

BC:	2	3	0	8
SP:	F	E	3	2

Data memory:

FE32:	2	3
FE33:	0	9

After instruction execution:

BC:	0	9	2	3
SP:	F	E	3	4

Data memory:

FE32:	2	3
FE33:	0	9

PUSH src src = BC, DE, HL, AF, IX, IY, IM, IR, DA, RA

Operation:

SP ← SP − 2
(SP) ← src

The Stack Pointer (SP) is decremented by two and the source is loaded into the location addressed by the updated SP; the low byte of the source (or Flag register for AF) is loaded into the addressed memory location and the upper byte of the source is loaded into the addressed memory location incremented by one. The contents of the source are unaffected. If the source is a memory location, the source and the new top of the stack must be non-overlapping.

Flags: No flags affected

Exceptions: System Stack Overflow Warning

Addressing Mode	Syntax	Instruction Format
R:	PUSH RR	11 | rr | 101
	PUSH XY	11 | Φ11 | 101 11 | 100 | 101
IM:	PUSH nn	11 | 111 | 101 11 | 110 | 101 n(low) n(high)
IR:	PUSH (HL)	11 | 011 | 101 11 | 000 | 101
DA:	PUSH (addr)	11 | 011 | 101 11 | 010 | 101 addr(low) addr(high)
RA:	PUSH <addr>	11 | 011 | 101 11 | 110 | 101 disp(low) disp(high)

Field Encodings:

Φ: 0 for IX, 1 for IY
rr: 000 for BC, 010 for DE, 100 for HL, 110 for AF

Example: PUSH BC

Before instruction execution:

BC:	0	9	2	3
SP:	F	E	3	4

Data memory:

FE32:	0	0
FE33:	0	0

After instruction execution:

BC:	0	9	2	3
SP:	F	E	3	2

Data memory:

FE32:	2	3
FE33:	0	9

RES
Reset Bit

RES b,dst dst = R, IR, SX

Operation: dst(b) ← 0

The specified bit b within the destination operand is cleared to 0. The other bits in the destination are unaffected. The bit number b must be between 0 and 7.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RES b,R	11 001 011 | 10 b r
IR:	RES b,(HL)	11 001 011 | 10 b 110
SX:	RES b,(XY + d)	11 Φ11 101 | 11 001 011 | d | 10 b 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: RES 1,A

Before instruction execution: **A:** 00010110

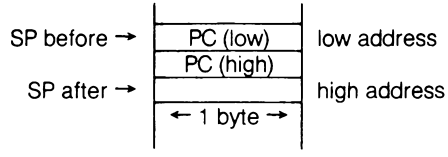
After instruction execution: **A:** 00010100

RET [cc]

Operation: If the cc is satisfied then: PC ← (SP)
SP ← SP + 2

This instruction is used to return to a previously executing procedure at the end of a procedure entered by a Call instruction. For a conditional return, one of the Zero, Carry, Sign, or Parity/Overflow flags is checked to see if its setting matches the condition code "cc" encoded in the instruction; if the condition is not satisfied, the instruction following the Return instruction is executed, otherwise a value is popped from the stack and loaded into the Program Counter (PC), thereby specifying the location of the next instruction to be executed. For an unconditional return, the return is always taken and a condition code is not specified.

The following figure illustrates the format of the PC on the stack for the Return instruction:



Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RET cc	11 | cc | 000
	RET	11 | 001 | 001

Field Encodings: **cc:** 000 for NZ, 001 for Z, 010 for NC, 011 for C, 100 for PO or NV, 101 for PE or V, 110 for P or NS, 111 for M or S

Example: RET NC

Before instruction execution:

F:		**szxhxvn0**
PC:	**2 5**	**2 8**
SP:	**F F**	**2 4**

Data memory:

FF24:	**3 3**
FF25:	**1 6**

After instruction execution:

F:		**szxhxvn0**
PC:	**1 6**	**3 3**
SP:	**F F**	**2 6**

Data memory:

FF24:	**3 3**
FF25:	**1 6**

RETI
Return from Interrupt

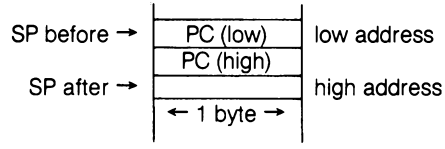
RETI

Operation:

PC ← (SP)
SP ← SP + 2

This instruction is used to return to a previously executing procedure at the end of a procedure entered by an interrupt while in interrupt mode 0, 1, or 2. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC).

The following figure illustrates the format of the PC on the stack for the Return from Interrupt instruction:



A special sequence of bus transactions is performed when this instruction is encountered in order to control Z80 family peripherals; see Chapter 12.

Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	RETI	11 101 101 | 01 001 101

Example: RETI

Before instruction execution:

PC:	**8 4**	**1 0**
SP:	**F F**	**C 6**

Data memory:

FFC6:	**7 2**
FFC7:	**1 9**

After instruction execution:

PC:	**1 9**	**7 2**
SP:	**F F**	**C 8**

Data memory:

FFC6:	**7 2**
FFC7:	**1 9**

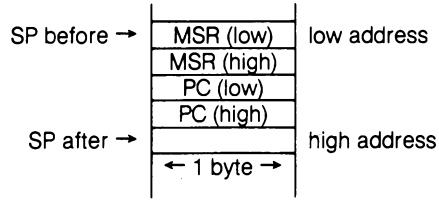
RETIL

Operation:

PS ← (SP)
SP ← SP + 4

This instruction is used to return to a previously executing procedure at the end of a procedure entered by an interrupt while in interrupt mode 3 or a trap. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC) and Master Status register (MSR).

The following figure illustrates the format of the program status (PC and MSR) on the system stack for the Return from Interrupt Long instruction:



Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	RETIL	11 101 101 01 010 101

Example: RETIL

Before instruction execution:

PC:	8 4	1 0
SP:	F F	C 6
MSR:	0 0	0 0

Data memory:

FFC6:	7 F
FFC7:	4 0
FFC8:	7 2
FFC9:	1 9

After instruction execution:

PC:	1 9	7 2
SP:	F F	C A
MSR:	4 0	7 F

Data memory:

FFC6:	7 F
FFC7:	4 0
FFC8:	7 2
FFC9:	1 9

RETN

Return from Nonmaskable Interrupt

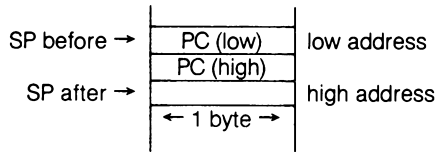
RETN

Operation:

PC ← (SP)
SP ← SP + 2
MSR(0-7) ← IFF(0-7)

This instruction is used to return to a previously executing procedure at the end of a procedure entered by a nonmaskable interrupt while in interrupt mode 0, 1, or 2. The contents of the location addressed by the Stack Pointer (SP) are popped into the Program Counter (PC). The previous setting of the interrupt masks in the Master Status register are restored.

The following figure illustrates the format of the PC on the stack for the Return from Nonmaskable Interrupt instruction:



Flags: No flags affected

Exceptions: Privileged Instruction

Addressing Mode	Syntax	Instruction Format
	RETN	11 101 101 | 01 000 101

Example: RETN

Before instruction execution:

PC:	8 4	1 0
SP:	F F	C 6
MSR:	4 0	0 0

After instruction execution:

PC:	1 9	7 2
SP:	F F	C 8
MSR:	4 0	7 F

Shadow Interrupt register:

7 F

Data memory:

FFC6:	7 2
FFC7:	1 9

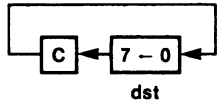
Data memory:

FFC6:	7 2
FFC7:	1 9

RL dst dst = R, IR, SX

Operation:

tmp ← dst
dst(0) ← C
C ← dst(7)
dst(n + 1) ← tmp(n) for n = 0 to 6



The contents of the destination operand are concatenated with the Carry flag and together they are rotated left one bit position. Bit 7 of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 0 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RL R	11 001 011 | 00 010 r
IR:	RL (HL)	11 001 011 | 00 010 110
SX:	RL (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 010 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: RL D

Before instruction execution:

F:	**szxhxpn0**
D:	**10001111**

After instruction execution:

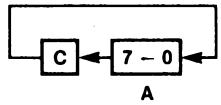
F:	**00x0x101**
D:	**00011110**

RLA
Rotate Left Accumulator

RLA

Operation:

tmp ← A
A(0) ← C
C ← A(7)
A(n + 1) ← tmp(n) for n = 0 to 6



The contents of the accumulator are concatenated with the Carry flag and together they are rotated left one bit position. Bit 7 of the accumulator is moved to the Carry flag and the Carry flag is moved to bit 0 of the destination.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RLA	00 010 111

Example:

RLA

Before instruction execution:

AF:	01110110	szxhxpn1

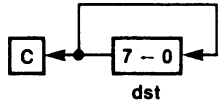
After instruction execution:

AF:	11101101	szx0xp00

RLC dst dst = R, IR, SX

Operation:

tmp ← dst
C ← dst(7)
dst(0) ← tmp(7)
dst(n + 1) ← tmp(n) for n = 0 to 6



The contents of the destination operand are rotated left one bit position. Bit 7 of the destination operand is moved to the bit 0 position and also replaces the Carry flag.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RLC R	11 001 011 | 00 000 r
IR:	RLC (HL)	11 001 011 | 00 000 110
SX:	RLC (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 000 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: RLC B

Before instruction execution:

F:	**szxhxpnc**
B:	**10001000**

After instruction execution:

F:	**00x0x101**
B:	**00010001**

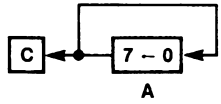
RLCA

Rotate Left Circular (Accumulator)

RLCA

Operation:

tmp ← A
C ← A(7)
A(0) ← tmp(7)
A(n + 1) ← tmp(n) for n = 0 to 6



The contents of the accumulator are rotated left one bit position. Bit 7 of the accumulator is moved to the bit 0 position and also replaces the Carry flag.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 7 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RLCA	00 000 111

Example:

RLCA

Before instruction execution:

AF:	10001000	szxhxpnc

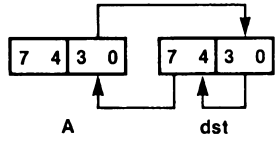
After instruction execution:

AF:	00010001	szx0xp01

RLD

Operation:

tmp(0:3) ← A(0:3)
A(0:3) ← dst(4:7)
dst(4:7) ← dst(0:3)
dst(0:3) ← tmp(0:3)



The low digit of the accumulator is logically concatenated to the destination byte whose memory address is in the HL register. The resulting three-digit quantity is rotated to the left by one BCD digit (four bits). The lower digit of the source is moved to the upper digit of the source; the upper digit of the source is moved to the lower digit of the accumulator, and the lower digit of the accumulator is moved to the lower digit of the source. The upper digit of the accumulator is unaffected. In multiple-digit BCD arithmetic, this instruction can be used to shift to the left a string of BCD digits, thus multiplying it by a power of ten. The accumulator serves to transfer digits between successive bytes of the string. This is analogous to the use of the Carry flag in multiple-precision shifting using the RL instruction.

Flags:

S: Set if the accumulator is negative after the operation; cleared otherwise
Z: Set if the accumulator is zero after the operation; cleared otherwise
H: Cleared
P: Set if the parity of the accumulator is even after the operation; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RLD	11 101 101 | 01 101 111

Example:

RLD

Before instruction execution:

AF:	**3 7**	**szxhxpnc**
HL:	**5 0 0 0**	

Data memory:

5000:	**0 4**

After instruction execution:

AF:	**3 0**	**00x0x10c**
HL:	**5 0 0 0**	

Data memory:

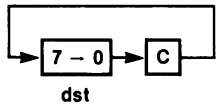
5000:	**4 7**

RR
Rotate Right

RR dst dst = R, IR, SX

Operation:

tmp ← dst
dst(7) ← C
C ← dst(0)
dst(n) ← tmp(n + 1) for n = 0 to 6



The contents of the destination operand are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the destination operand is moved to the Carry flag and the Carry flag is moved to bit 7 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RR R	11 001 011 | 00 011 r
IR:	RR (HL)	11 001 011 | 00 011 110
SX:	RR (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 011 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: RR B

Before instruction execution:

F:	szxhxpn0
B:	11011101

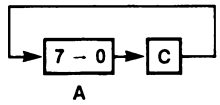
After instruction execution:

F:	00x0x001
B:	01101110

RRA

Operation:

tmp ← dst
A(7) ← C
C ← A(0)
A(n) ← tmp(n + 1) for n = 0 to 6



The contents of the accumulator are concatenated with the Carry flag and together they are rotated right one bit position. Bit 0 of the accumulator is moved to the Carry flag and the Carry flag is moved to bit 7 of the accumulator.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RRA	00 011 111

Example:

RRA

Before instruction execution:

AF:	**11100001**	**szxhxpn0**

After instruction execution:

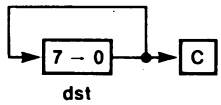
AF:	**01110000**	**szx0xp01**

RRC
Rotate Right Circular

RRC dst dst = R, IR, SX

Operation:

tmp ← dst
C ← dst(0)
dst(7) ← tmp(0)
dst(n) ← tmp(n + 1) for n = 0 to 6



The contents of the destination operand are rotated right one bit position. Bit 0 of the destination operand is moved to the bit 7 position and also replaces the Carry flag.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	RRC R	11 001 011 | 00 001 r
IR:	RRC (HL)	11 001 011 | 00 001 110
SX:	RRC (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 001 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: RRC A

Before instruction execution:

AF:	**00110001**	**szxhxpnc**

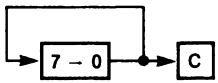
After instruction execution:

AF:	**10011000**	**10x0x001**

RRCA

Operation:

tmp ← A
C ← A(0)
A(7) ← temp(0)
A(n) ← tmp(n + 1) for n = 0 to 6



The contents of the accumulator are rotated right one bit position. Bit 0 of the accumulator is moved to the bit 7 position and also replaces the Carry flag.

Flags:

S: Unaffected
Z: Unaffected
H: Cleared
P: Unaffected
N: Cleared
C: Set if the bit rotated from bit 0 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RRCA	00 001 111

Example:

RRCA

Before instruction execution:

AF:	**00010001**	**szxhxpnc**

After instruction execution:

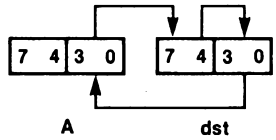
AF:	**10001000**	**szx0xp01**

RRD
Rotate Right Digit

RRD

Operation:

tmp(0:3) ← A(0:3)
A(0:3) ← dst(0:3)
dst(0:3) ← dst(4:7)
dst(4:7) ← tmp(0:3)



The low digit of the accumulator is logically concatenated to the destination byte whose memory address is in the HL register. The resulting three-digit quantity is rotated to the right by one BCD digit (four bits). The lower digit of the source is moved to the upper digit of the source; the upper digit of the source is moved to the lower digit of the accumulator, and the lower digit of the accumulator is moved to the lower digit of the source. The upper digit of the accumulator is unaffected. In multiple-digit BCD arithmetic, this instruction can be used to shift to the right a string of BCD digits, thus multiplying it by a power of ten. The accumulator serves to transfer digits between successive bytes of the string. This is analogous to the use of the Carry flag in multiple-precision shifting using the RR instruction.

Flags:

S: Set if the accumulator is negative; cleared otherwise
Z: Set if the accumulator is zero after the operation; cleared otherwise
H: Cleared
P: Set if the parity of the accumulator is even after the operation; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RRD	11 101 101 01 100 111

Example:

RRD

Before instruction execution:

AF:	**0 6**	**szxhxpnc**
H:	**5 0**	**0 0**

Data memory:

5000:	**3 2**

After instruction execution:

AF:	**0 2**	**00x0x00c**
H:	**5 0**	**0 0**

Data memory:

5000:	**6 3**

RST address

Operation:

SP ← SP – 2
(SP) ← PC
PC ← address

The current Program Counter (PC) is pushed onto the stack and the PC is loaded with a constant address encoded in the instruction. Execution then begins at this address. The restart instruction allows for a call to one of eight fixed locations as shown in the table below. The table also indicates the encoding of the address used in the instruction encoding. (The address is in hexadecimal, the encoding in binary.)

Address	t encoding
00_H	000
08_H	001
10_H	010
18_H	011
20_H	100
28_H	101
30_H	110
38_H	111

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	RST address	11 | t | 111

Field Encoding: t: See table above

Example: RST 18H

Before instruction execution:

PC:	**4 6**	**2 0**
SP:	**F F**	**C 4**

Data memory:

FFC3:	**F F**
FFC4:	**F F**

After instruction execution:

PC:	**0 0**	**1 8**
SP:	**F F**	**C 2**

Data memory:

FFC3:	**2 0**
FFC4:	**4 6**

SBC
Subtract with Carry (Byte)

SBC [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A − src − C

The source operand together with the Carry flag is subtracted from the accumulator and the difference is stored in the accumulator. The contents of the source are not affected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 4 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs and the result is the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SBC A,R	10 011 r
RX:	SBC A,RX	11 Φ11 101 | 10 011 rx
IM:	SBC A,n	11 011 110 | n
IR:	SBC A,(HL)	10 011 110
DA:	SBC A,(addr)	11 011 101 | 10 011 111 | addr(low) | addr(high)
X:	SBC A,(XX + dd)	11 111 101 | 10 011 xx | d(low) | d(high)
SX:	SBC A,(XY + d)	11 Φ11 101 | 10 011 110 | d
RA:	SBC A,<addr>	11 111 101 | 10 011 000 | disp(low) | disp(high)
SR:	SBC A,(SP + dd)	11 011 101 | 10 011 000 | d(low) | d(high)
BX:	SBC A,(XXA + XXB)	11 011 101 | 10 011 bx

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: SBC A,(HL)

Before instruction execution:

AF:	4 8	szxhxvn1
HL:	2 4	5 4

Data memory:

2454:	1 8

After instruction execution:

AF:	2 F	00x1x010
HL:	2 4	5 4

Data memory:

2454:	1 8

SBC dst,src

dst = HL
src = BC, DE, HL, SP
or
dst = IX
src = BC, DE, IX, SP
or
dst = IY
src = BC, DE, IY, SP

Operation: dst ← dst − src − C

The source operand together with the Carry flag is subtracted from the destination and the result is stored in the destination. The contents of the source are not affected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative, cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 12 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, the operands are of different signs and the result is of the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise.

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	SBC HL,RR	11 | 101 | 101 | 01 | rr | 010
	SBC XY,RR	11 | Φ11 | 101 | 11 | 101 | 101 | 01 | rr | 010

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 000 for BC, 010 for DE, 100 for subtract register from itself, 110 for SP

Example: SBC HL,DE

Before instruction execution:

F:		**szxhxvn1**
DE:	**0 0**	**1 1**
HL:	**0 1**	**0 0**

After instruction execution:

F:		**00x0x010**
DE:	**0 0**	**1 1**
HL:	**0 0**	**E E**

SC
System Call

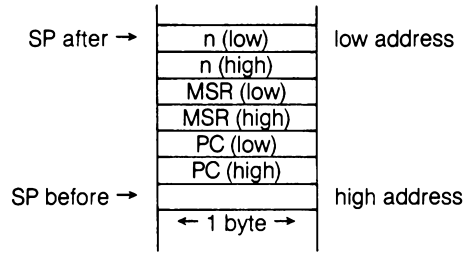
SC nn

Operation:

SP ← SP – 4
(SP) ← PS
SP ← SP – 2
(SP) ← nn
PS ← System Call Program Status

This instruction is used for controlled access to operating system software in a manner similar to a trap or interrupt. The current program status is pushed onto the system stack followed by a 16-bit constant embedded in the instruction. The program status consists of the Master Status register (MSR) and the updated Program Counter (PC), which points to the first instruction byte following the SC instruction. Next the 16-bit constant in the System Call instruction is pushed onto the system stack. The system Stack Pointer is always used regardless of whether system or user mode is in effect. The new program status is loaded from the Interrupt/Trap Vector Table entry associated with the SC instruction. CPU control is passed to the procedure whose address is the PC value contained in the new program status.

The following figure illustrates the format of the saved program status on the system stack:



Flags: No flags affected

Exceptions: System Call Trap, System Stack Overflow Warning

Addressing Mode	Syntax	Instruction Format
	SC nn	11 101 101 | 01 110 001 | n(low) | n(high)

Example: SC 0155H

Before instruction execution:

PC:	4 6	2 0
SP:	F F	C 9
MSR:	4 0	7 F

After instruction execution:

PC:	9 0	8 8
SP:	F F	C 3
MSR:	0 0	2 3

Interrupt/Trap Vector Table Pointer:

3 6	5 2

Data memory:

FFC3:	5 5
FFC4:	0 1
FFC5:	7 F
FFC6:	4 0
FFC7:	2 0
FFC8:	4 6

Physical memory:

365250:	2 3
365251:	0 0
365252:	8 8
365253:	9 0

Note: The physical memory addresses are 24-bit addresses emitted by the MMU. The data memory addresses are the 16-bit addresses from the CPU.

SCF
Set Carry Flag

SCF

Operation: C ← 1

The Carry flag is set to 1.

Flags:
S: Unaffected
Z: Unaffected
H: Cleared
V: Unaffected
N: Cleared
C: Set

Exceptions: None

Addressing Mode	Syntax	Instruction Format
	SCF	00 110 111

Example: SCF

Before instruction execution:

F: szxhxvnc

After instruction execution:

F: szx0xv01

SET b,dst dst = R, IR, SX

Operation: dst(b) ← 1

The specified bit b within the destination operand is set to 1. The other bits in the destination are unaffected. The bit to be set is specified by a 3-bit field in the instruction; this field contains the binary encoding for the bit number to be set. The bit number must be between 0 and 7.

Flags: No flags affected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SET b,R	11 001 011 | 11 b r
IR:	SET b,(HL)	11 001 011 | 11 b 110
SX:	SET b,(XY + d)	11 Φ11 101 | 11 001 011 | d | 11 b 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: SET 1,A

Before instruction execution:

A: 00010100

After instruction execution:

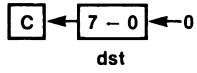
A: 00010110

SLA
Shift Left Arithmetic

SLA dst dst = R, IR, SX

Operation:

tmp ← dst
C ← dst(7)
dst(0) ← 0
dst(n + 1) ← tmp(n) for n = 0 to 6



The contents of the destination operand are shifted left one bit position. Bit 7 of the destination operand is moved to the Carry flag and zero is shifted into bit 0 of the destination.

Flags:

S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 7 was a 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SLA R	11 001 011 | 00 100 r
IR:	SLA (HL)	11 001 011 | 00 100 110
SX:	SLA (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 100 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: SLA L

Before instruction execution:

F:	**szxhxpnc**
L:	**10110001**

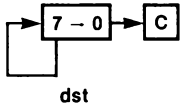
After instruction execution:

F:	**00x0x001**
L:	**01100010**

SRA dst dst = R, IR, SX

Operation:

tmp ← dst
C ← dst(0)
dst(7) ← tmp(7)
dst(n) ← tmp(n + 1) for n = 0 to 6



The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and bit 7 remains unchanged.

Flags:

S: Set if the result is negative; cleared otherwise
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 0 was 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SRA R	11 001 011 | 00 101 r
IR:	SRA (HL)	11 001 011 | 00 101 110
SX:	SRA (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 101 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: SRA (IX + 3)

Before instruction execution:

F:	szxhxpnc
IX: 1 0	0 0

Data memory:

1003:	10111000

After instruction execution:

F:	10x0x000
IX: 1 0	0 0

Data memory:

1003:	11011100

Address calculation:

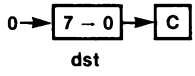
```
  1000
+    3
------
  1003
```

SRL
Shift Right Logical

SRL dst dst = R, IR, SX

Operation:

tmp ← dst
C ← dst(0)
dst(7) ← 0
dst(n) ← tmp(n + 1) for n = 0 to 6



The contents of the destination operand are shifted right one bit position. Bit 0 of the destination operand is moved to the Carry flag and zero is shifted into bit 7 of the destination.

Flags:

S: Cleared
Z: Set if the result is zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Set if the bit shifted from bit 0 was 1; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SRL R	11 001 011 | 00 111 r
IR:	SRL (HL)	11 001 011 | 00 111 110
SX:	SRL (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 111 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: SRL B

Before instruction execution:

F:	**szxhxpnc**
B:	**10001111**

After instruction execution:

F:	**00x0x101**
B:	**01000111**

SUB [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A − src

The source operand is subtracted from the accumulator and the difference is stored in the accumulator. The contents of the source are unaffected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 4 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs and the result is the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SUB A,R	10 010 r
RX:	SUB A,RX	11 Φ11 101 | 10 010 rx
IM:	SUB A,n	11 010 110 | n
IR:	SUB A,(HL)	10 010 110
DA:	SUB A,(addr)	11 011 101 | 10 010 111 | addr(low) | addr(high)
X:	SUB A,(XX + dd)	11 111 101 | 10 010 xx | d(low) | d(high)
SX:	SUB A,(XY + d)	11 Φ11 101 | 10 010 110 | d
RA:	SUB A,<addr>	11 111 101 | 10 010 000 | disp(low) | disp(high)
SR:	SUB A,(SP + dd)	11 011 101 | 10 010 000 | d(low) | d(high)
BX:	SUB A,(XXA + XXB)	11 011 101 | 10 010 bx

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rx:** 100 for high byte, 101 for low byte
- **xx:** 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
- **bx:** 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: SUB A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxvnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**3 0**	**00x0x010**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

SUBW
Subtract (Word)

SUBW [HL,]src src = R, IM, DA, X, RA

Operation: HL ← HL − src

The source operand is subtracted from the HL register and the difference is stored in the HL register. The contents of the source are unaffected. Twos-complement subtraction is performed.

Flags:
- **S:** Set if the result is negative; cleared otherwise
- **Z:** Set if the result is zero; cleared otherwise
- **H:** Set if there is a borrow from bit 12 of the result; cleared otherwise
- **V:** Set if arithmetic overflow occurs, that is, if the operands are of the opposite signs and the result is the same sign as the source; cleared otherwise
- **N:** Set
- **C:** Set if there is a borrow from the most significant bit of the result; cleared otherwise.

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	SUBW HL,RR	11 101 101 | 11 rr 110
	SUBW HL,XY	11 Φ11 101 | 11 101 101 | 11 101 110
IM:	SUBW HL,nn	11 111 101 | 11 101 101 | 11 111 110 | n(low) | n(high)
DA:	SUBW HL,(addr)	11 011 101 | 11 101 101 | 11 011 110 | addr(low) | addr(high)
X:	SUBW HL,(XY + dd)	11 111 101 | 11 101 101 | 11 xy 110 | d(low) | d(high)
RA:	SUBW HL,<addr>	11 011 101 | 11 101 101 | 11 111 110 | disp(low) | disp(high)
IR:	SUBW HL,(HL)	11 011 101 | 11 101 101 | 11 001 110

Field Encodings:
- **Φ:** 0 for IX, 1 for IY
- **rr:** 001 for BC, 011 for DE, 101 for HL, 111 for SP
- **xy:** 001 for (IX + dd), 011 for (IY + dd)

Example: SUBW HL,DE

Before instruction execution:

F:		szxhxvnc
DE:	0 0	1 0
HL:	A 1	2 3

After instruction execution:

F:		10x0x010
DE:	0 0	1 0
HL:	A 1	1 3

TSET
Test and Set

TSET dst dst = R, IR, SX

Operation:

S ← dst(7)
dst ← FF_H

Bit 7 within the destination operand is tested, and the Sign flag is set to 1 if the specified bit is 1, otherwise the Sign flag is cleared to 0. The contents of the destination are then set to all 1s. For memory operands, the operand is always fetched from the external memory; on the Z-BUS interface, the status lines indicate a Test and Set operation during the memory read transaction.

Between the data read and subsequent write transactions, bus request is not granted. The data is read from memory, even if it is also present in the cache.

Flags:

S: Set if bit 7 is 1; cleared otherwise
Z: Unaffected
H: Unaffected
P: Unaffected
N: Unaffected
C: Unaffected

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	TSET R	11 001 011 | 00 110 r
IR:	TSET (HL)	11 001 011 | 00 110 110
SX:	TSET (XY + d)	11 Φ11 101 | 11 001 011 | d | 00 110 110

Field Encoding: Φ: 0 for IX, 1 for IY

Example: TSET (HL)

Before instruction execution:

F:	szxhxpnc
HL: 0 3	8 2

After instruction execution:

F:	0zxhxpnc
HL: 0 3	8 2

Data memory:

0382: 00010111

Data memory:

0382: 11111111

TSTI
Test Input

TSTI (C)

Operation: F ← test (C)

During the I/O transaction, the peripheral address from the C register is placed on the low byte of the address bus, the contents of the B register are placed on address lines A_8–A_{15}, and the contents of the I/O Page register are placed on address lines A_{16}–A_{23}. The byte of data from the selected peripheral is tested and the CPU flags set accordingly. No CPU register or memory location is modified.

Flags:
S: Set if the tested byte is negative; cleared otherwise
Z: Set if the tested byte is zero; cleared otherwise
H: Cleared
P: Set if the parity of the tested byte is even; cleared otherwise
N: Cleared
C: Unaffected

Exceptions: Privileged Instruction (if the Inhibit User I/O bit in the Trap Control register is set to 1)

Addressing Mode	Syntax	Instruction Format
	TSTI (C)	11 101 101 01 110 000

Example: TSTI (C)

Before instruction execution:

F:		szxhxpnc
BC:	5 0	4 6

After instruction execution:

F:	10x0x10c

I/O Page register:

1 2

Byte 93_H available at I/O port 125046_H.

XOR
Exclusive OR

XOR [A,]src src = R, RX, IM, IR, DA, X, SX, RA, SR, BX

Operation: A ← A XOR src

A logical EXCLUSIVE OR operation is performed between the corresponding bits of the source operand and the accumulator and the result is stored in the accumulator. A 1 bit is stored wherever the corresponding bits in the two operands are different; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:
S: Set if the most significant bit of the result is set; cleared otherwise
Z: Set if all bits of the result are zero; cleared otherwise
H: Cleared
P: Set if the parity of the result is even; cleared otherwise
N: Cleared
C: Cleared

Exceptions: None

Addressing Mode	Syntax	Instruction Format
R:	XOR A,R	10 | 101 | r
RX:	XOR A,RX	11 | Φ11 | 101 | 10 | 101 | rx
IM:	XOR A,n	11 | 101 | 110 | n
IR:	XOR A,(HL)	10 | 101 | 110
DA:	XOR A,(addr)	11 | 011 | 101 | 10 | 101 | 111 | addr(low) | addr(high)
X:	XOR A,(XX + dd)	11 | 111 | 101 | 10 | 101 | xx | d(low) | d(high)
SX:	XOR A,(XY + d)	11 | Φ11 | 101 | 10 | 101 | 110 | d
RA:	XOR A,<addr>	11 | 111 | 101 | 10 | 101 | 000 | disp(low) | disp(high)
SR:	XOR A,(SP + dd)	11 | 011 | 101 | 10 | 101 | 000 | d(low) | d(high)
BX:	XOR A,(XXA + XXB)	11 | 011 | 101 | 10 | 101 | bx

Field Encodings:
Φ: 0 for IX, 1 for IY
rx: 100 for high byte, 101 for low byte
xx: 001 for (IX + dd), 010 for (IY + dd), 011 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

Example: XOR A,(HL)

Before instruction execution:

AF:	**4 8**	**szxhxpnc**
HL:	**2 4**	**5 4**

Data memory:

2454:	**1 8**

After instruction execution:

AF:	**5 0**	**00x0x100**
HL:	**2 4**	**5 4**

Data memory:

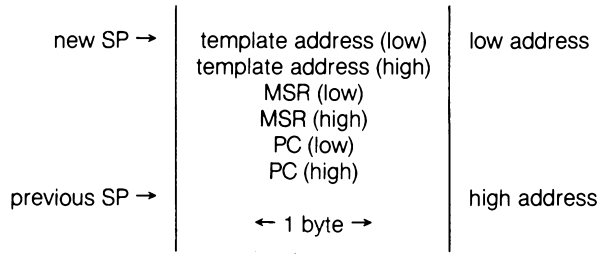
2454:	**1 8**

EXTENDED INSTRUCTION
EPU Internal Operation

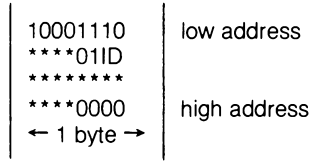
Operation: EPU ← template

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), and template address. The format of the system stack after the trap is indicated by the following figure:



The format for the EPU template for this instruction is indicated in the following figure:



where ID is the two bit ID number specifying the EPU to process this instruction and * indicates bits that encode the operation to be performed.

The template has no alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned.

Flags: No flags affected

Exceptions: Extended Instruction

Addressing Mode	Operation	Instruction Format
	EPU Internal Operation	11 | 101 | 101 | 10 | 011 | 111 | template 1 | template 2 | template 3 | template 4

The template is a 4-byte field.

Operation:

EPU ← template
A ← EPU

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed. Next data from the EPU is loaded into the accumulator.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), and template address. The format of the system stack after the trap is indicated by the following figure:

new SP →	template address (low)	low address
	template address (high)	
	MSR (low)	
	MSR (high)	
	PC (low)	
	PC (high)	
previous SP →		high address
	← 1 byte →	

The format for the EPU template for this instruction is indicated in the following figure:

10001110	low address
****00ID	
****0000	
****0000	high address
← 1 byte →	

where ID is the 2-bit ID number specifying the EPU to process this instruction and * indicates bits that encode the operation to be performed.

The template has no alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. The CPU places the data on AD_8–AD_{15} into the accumulator.

Flags:

S: Set if the byte loaded into the accumulator has a 1 in bit 7; cleared otherwise
Z: Set if the byte loaded into the accumulator is zero; cleared otherwise
H: Cleared
P: Set if the parity of the byte loaded into the accumulator is even; cleared otherwise
N: Cleared
C: Unaffected

Exceptions:

Extended Instruction

Addressing Mode	Operation	Instruction Format
	A ← EPU	11 | 101 | 101 | 10 | 010 | 111 | template 1 | template 2 | template 3 | template 4

The template is a 4-byte field.

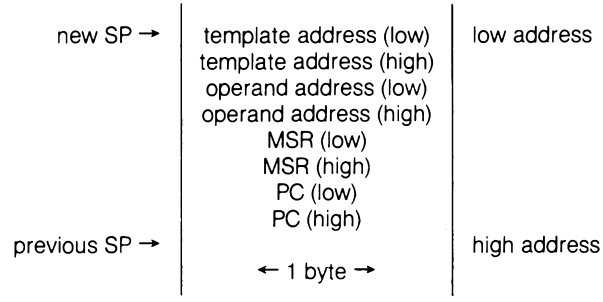
src = IR, DA, X, RA, SR, BX

Operation:

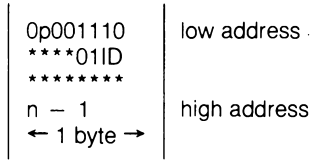
EPU ← template
EPU ← src

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed on the input operand. Next the data starting at the memory location determined by the source calculation is fetched from memory and loaded into the EPU; successive transfers are performed until the entire operand has been fetched. The number of bytes in the source operand is encoded in the fourth byte of the template. For PC Relative addressing mode, the address of the template is used instead of the address of the next instruction.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the following instruction, Master Status register (MSR), operand logical address, and template logical address. The format of the system stack after the trap is indicated by the following figure:



The format for the EPU template for this instruction is indicated in the following figure:



where p encodes whether the data resides in program memory (p = 1; Relative addressing mode) or data memory; ID is the 2-bit ID number specifying the EPU to process this instruction, * indicates bits that encode the operation to be performed, and n specifies the number of bytes of data to be transferred to the EPU.

Neither the template nor the operand has an alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. Table 10-2 shows the sequences of transactions for the various cases of data transfers to the EPU.

Flags: No flags affected

Exceptions: Extended Instruction

Addressing Mode	Operation	Instruction Format
IR:	EPU ← (HL)	11 101 101 | 10 100 110 | template 1 | template 2 | template 3 | template 4
DA:	EPU ← (addr)	11 101 101 | 10 100 111 | addr(low) | addr(high) | template 1 | template 2 | template 3 | template 4
X:	EPU ← (XX + dd)	11 101 101 | 10 xx 100 | d(low) | d(high) | template 1 | template 2 | template 3 | template 4
RA:	EPU ← <addr>	11 101 101 | 10 100 100 | disp(low) | disp(high) | template 1 | template 2 | template 3 | template 4
SR:	EPU ← (SP + dd)	11 101 101 | 10 000 100 | d(low) | d(high) | template 1 | template 2 | template 3 | template 4
BX:	EPU ← (XXA + XXB)	11 101 101 | 10 bx 100 | template 1 | template 2 | template 3 | template 4

Field Encodings: **xx:** 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

All templates are 4-byte fields.

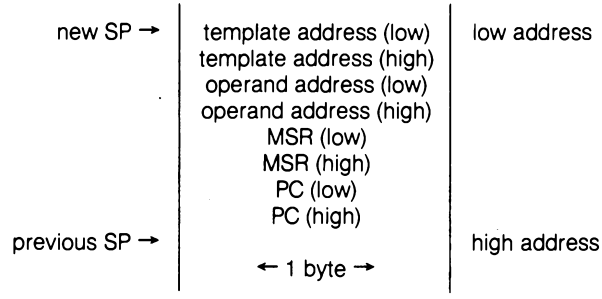
dst = IR, DA, X, RA, SR, BX

Operation:

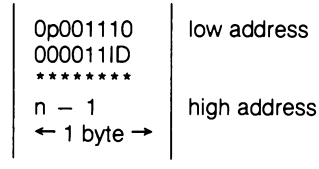
EPU ← template
dst ← EPU

If the EPU Enable bit in the Trap Control register is set to 1, indicating an EPU is in the system, then the 4-byte template embedded in the instruction is fetched from memory and loaded into the EPU, thus indicating to the EPU the operation to be performed. Next the data from the EPU is stored into memory starting at the location specified by the destination address; successive transfers are performed until the entire operand has been stored. The number of bytes in the source operand is encoded in the fourth byte of the template. For PC Relative addressing mode, the address of the template is used instead of the address of the next instruction.

If the EPU Enable control bit in the Trap Control register is cleared to 0, an EPU trap is initiated. The trap causes the following information to be pushed onto the system stack (in the following order): Program Counter (PC) of the next instruction, Master Status register (MSR), operand address, and template address. The format of the system stack after the trap is indicated by the following figure:



The format for the EPU template for this instruction is indicated in the following figure:



where p encodes whether the data resides in program space (p = 1; Relative addressing mode) or data memory; ID is the 2-bit ID number specifying the EPU to process this instruction, * indicates bits that encode the operation to be performed, and n specifies the number of bytes of data to be transferred from the EPU.

Neither the template nor the operand has an alignment restriction. The CPU fetches the template from external memory using two word transactions if the template is aligned on an even address, or a byte transaction followed by two word transactions if the template is unaligned. Table 10-2 shows the sequences of transactions for the various cases of data transfers from the EPU.

Flags: No flags affected

Exceptions: Extended Instruction

Addressing Mode	Operation	Instruction Format
IR:	(HL) ← EPU	11 101 101 | 10 101 110 | template 1 | template 2 | template 3 | template 4
DA:	(addr) ← EPU	11 101 101 | 10 101 111 | addr(low) | addr(high) | template 1 | template 2 | template 3 | template 4
X:	(XX + dd) ← EPU	11 101 101 | 10 xx 101 | d(low) | d(high) | template 1 | template 2 | template 3 | template 4
RA:	<addr> ← EPU	11 101 101 | 10 100 101 | disp(low) | disp(high) | template 1 | template 2 | template 3 | template 4
SR:	(SP + dd) ← EPU	11 101 101 | 10 000 101 | d(low) | d(high) | template 1 | template 2 | template 3 | template 4
BX:	(XXA + XXB) ← EPU	11 101 101 | 10 bx 101 | template 1 | template 2 | template 3 | template 4

Field Encodings:
xx: 101 for (IX + dd), 110 for (IY + dd), 111 for (HL + dd)
bx: 001 for (HL + IX), 010 for (HL + IY), 011 for (IX + IY)

All templates are 4-byte fields.

Chapter 6.
Interrupts and Traps

6.1 INTRODUCTION

Exceptions are conditions that can alter the normal flow of program execution. The Z280 CPU supports three kinds of exceptions: interrupts, traps, and resets.

Interrupts are asynchronous events generated by a device external to the CPU; peripheral devices use interrupts to request service from the CPU. Traps are synchronous events generated internally in the CPU by particular conditions that can occur during the attempted execution of an instruction. Thus, the difference between traps and interrupts is their origin. A trap condition is always reproducible by re-executing the program that created the trap, whereas an interrupt is generally independent of the currently executing task.

A hardware reset overrides all other conditions, including interrupts and traps. It occurs when the RESET line is activated, and it causes certain CPU control registers to be initialized. Resets are discussed in detail in Chapter 11.

6.2 INTERRUPTS

Two kinds of interrupts are activated by four different pins on the Z280 MPU. The nonmaskable interrupt (NMI) is an interrupt that cannot be disabled (masked) by software. Typically, NMI is reserved for high-priority external events that need immediate attention, such as an imminent power failure. Maskable interrupts are interrupts that can be disabled (masked) via software by clearing the appropriate bits in the Interrupt Request Enable field of the Master Status register.

There are seven maskable interrupts in the Z280 MPU architecture. Three of these interrupts are external inputs to the device (Interrupts A, B, and C); the other four maskable interrupts are asserted by the on-chip peripherals. The seven Interrupt Request Enable bits in the Master Status register control which of the requested interrupts are accepted. Interrupt requests are grouped as listed in Table 6-1, with each group controlled by a separate Interrupt Request Enable bit. The list is presented in order of decreasing priority, with sources within a group listed in order of decreasing priority.

The Enable Interrupt (EI) instruction is used to selectively enable the maskable interrupts (by setting the appropriate bits in the MSR to 1) and the Disable Interrupt (DI) instruction is used to selectively disable interrupts (by clearing the appropriate bits in the MSR to 0). When an interrupt source has been disabled, the CPU ignores any requests from that source. Because maskable interrupt requests are not retained by the CPU, the request signal on a maskable interrupt line must be asserted until the CPU acknowledges the request.

When enabling interrupts with the EI instruction, all maskable interrupts are automatically disabled (whether previously enabled or not) for the duration of the execution of the EI instruction and the immediately following instruction.

Interrupts are always accepted between instructions. The block move, block search, and block I/O instructions can be interrupted after any iteration.

Table 6-1. Grouping of Maskable Interrupt Requests

Members of Interrupt Group	Enable bit in MSR
Maskable Interrupt A line	0
Counter/Timer 0, DMA Channel 0	1
Maskable Interrupt B line	2
Counter/Timer 1, UART Receiver, DMA Channel 1	3
Maskable Interrupt C line	4
UART Transmitter, DMA Channel 2	5
Counter/Timer 2, DMA Channel 3	6

The Z280 CPU has four modes for handling externally generated interrupts, selectable using the IM instruction. The first three modes extend the Z80 CPU interrupt modes to accommodate the Z280 MPU's additional interrupt inputs in a compatible fashion. The fourth mode allows more flexibility in interrupt handling, providing support for nested interrupts and a sophisticated vectoring scheme. The on-chip peripherals always use this fourth interrupt mode, regardless of which mode is selected for the external interrupts. The current interrupt mode in effect can be read from the Interrupt Status register.

6.2.1 Interrupt Mode 0

Interrupt mode 0 is similar to the 8080 CPU interrupt response mode. For mode 0, an externally generated interrupt (maskable or nonmaskable) causes the User/$\overline{\text{System}}$ bit and the Single-Step bit in the Master Status register to be cleared to 0, thereby placing the CPU in system mode with single-stepping disabled. All the Interrupt Request Enable bits in the MSR are also cleared to zero, which disables the maskable interrupts. The previous condition of the MSR is not saved.

For nonmaskable interrupts, the current value in the Program Counter is saved on the system stack, using the System Stack Pointer, and the constant 0066_H is loaded into the Program Counter. Location 0066_H in system program memory is, then, the starting logical address of the nonmaskable interrupt service routine; this logical address can, of course, be translated into a physical memory address by the MMU.

For maskable interrupts, the interrupting device must place a Call or Restart instruction opcode on the data bus during the interrupt acknowledge bus transaction. The Z280 CPU reads this opcode and executes it; thus, the interrupting device, instead of memory, provides the first instruction of the service routine. Typically, a Restart instruction is used, since the Restart opcode is only one byte long, meaning that the interrupting peripheral needs to supply only one byte of information. Alternatively, a 3-byte call to any location can be executed.

6.2.2 Interrupt Mode 1

In interrupt mode 1, the Z280 CPU automatically executes a Restart to a fixed location when an interrupt occurs. An externally generated interrupt (maskable or nonmaskable) causes the User/$\overline{\text{System}}$ bit, the Single-Step bit, and all Interrupt Request Enable bits in the Master Status register to be cleared to 0, which puts the CPU in system mode with single-stepping disabled. The previous condition of the MSR is not saved. The current value in the Program Counter is pushed onto the system-mode stack. For nonmaskable interrupts, the constant 0066_H is then loaded into the Program Counter; thus, 0066_H is the starting address of the nonmaskable interrupt service routine. For maskable interrupts, the constant 0038_H is loaded into the Program Counter; 0038_H will be the starting address of the maskable interrupt service routine. These logical addresses can be converted to physical addresses by the MMU.

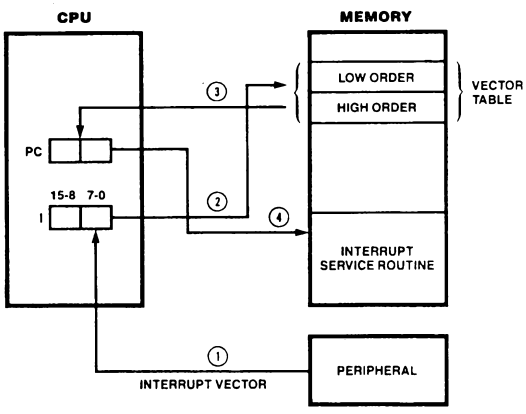
6.2.3 Interrupt Mode 2

Interrupt mode 2 is a vectored interrupt response mode for maskable interrupts, wherein the interrupting device identifies the starting location of the service routine using an 8-bit vector read by the CPU during the interrupt acknowledge cycle.

An externally generated interrupt (maskable or nonmaskable) causes the User/$\overline{\text{System}}$ bit, the Single-Step bit, and the Interrupt Enable Request bits in the Master Status register to be cleared to 0, which puts the CPU in system mode with single-stepping disabled. The previous condition of the MSR is not saved. The current value in the Program Counter is pushed onto the system mode stack.

For nonmaskable interrupts, the constant 0066_H is then loaded into the Program Counter; thus, 0066_H is the starting address of the nonmaskable interrupt service routine. For maskable interrupts, the programmer must maintain a table in memory of the 16-bit starting addresses for every maskable interrupt service routine. This table can be located anywhere in the system mode data memory address space, starting on a 256-byte memory boundary. When a maskable interrupt is accepted, a 16-bit pointer into this table is generated in order to select the starting address of the appropriate service routine from the table entries. The peripheral generating the interrupt places an 8-bit vector on the data bus in response to the interrupt acknowledge. This vector becomes the lower eight bits of the pointer into the table. The upper eight bits of the pointer are the contents of the I register. This pointer is treated as an address in the system data memory space that can be translated to a physical address by the MMU. The actual logical address of the service routine is found by referencing the word located at the address formed by concatenating the I register's contents with the vector. Figure 6-1

illustrates the sequence of events for processing mode 2 maskable interrupts. A reset clears the I register to all zeros.



NOTES:
1. Interrupt vector generated by peripheral is read by CPU during interrupt acknowledge cycle.
2. Vector combined with I register contents form 16-bit memory address pointing to vector table.
3. Two bytes are read sequentially from vector table. These two bytes are read into the PC.
4. Processor control is transferred to interrupt service routine and execution continues.

Figure 6-1. Mode 2 Interrupt Processing

The Master Status register is not saved when processing interrupts under interrupt modes 0, 1, and 2. If the Z280 CPU is running in the user mode when an interrupt occurs, the MSR is automatically changed to system mode when the interrupt is acknowledged, without recording the previous user mode of operation. Similarly, the single-step mode and the maskable interrupts are automatically disabled during interrupt processing, with no saving of the previous status. Thus, to resume processing of an interrupted user-mode program after the execution of an interrupt service routine, the operating system must change the Master Status register in order to switch back to user mode; the Return from Interrupt Long instruction can be used for this purpose.

In interrupt modes 0, 1, and 2, a nonmaskable interrupt automatically disables all maskable interrupts (as in the Z80 CPU). All of the Interrupt Request Enable bits (bits 0 through 6 in the MSR) are copied to a special register in the CPU called the Interrupt Shadow register. The Interrupt Request Enable bits are then cleared to all zeros. A Return from Nonmaskable Interrupt instruction restores the previous settings of the Interrupt Request Enable bits by copying the contents of the Interrupt Shadow register into bits 0 through 6 of the MSR. The nesting is only one level deep (again, as in the Z80 CPU).

For a Z80 Bus configuration of the Z280 MPU, only one interrupt line (either Interrupt A, Interrupt B, or Interrupt C) can be used if interrupt modes 0, 1, or 2 and the Z80 family peripherals are used; Z80 peripherals being serviced on multiple interrupt lines would all be affected by a Return from Interrupt (RETI) instruction.

6.2.4 Interrupt Mode 3

Interrupt mode 3 exploits the advanced features of the Z280 MPU architecture. When an interrupt request is accepted (maskable or nonmaskable), the Master Status register, Program Counter, and a 16-bit "reason code" are automatically stored on the system-mode stack. Next, new values for the MSR and PC are fetched from a table in memory called the Interrupt/Trap Vector Table, thereby determining the operating modes and starting address of the service routine (see section 6.5). The reason code for externally generated interrupts is the contents of the data bus during the interrupt acknowledge, and is usually supplied by the interrupting device. For 8-bit data bus configurations of the Z280 MPU, the upper byte of the reason code is all zeros. For interrupts from the on-chip peripherals, the reason code is identical to the vector address in the Interrupt/Trap Vector Table, thereby identifying the interrupting device. The Interrupt/Trap Vector Table Pointer register in the CPU is used to reference the Interrupt/Trap Vector Table during mode 3 interrupt processing.

Interrupt mode 3 is the intended mode of operation when using the advanced features of the Z280 MPU architecture, such as system and user modes and single-stepping, since the Master Status register of the interrupted task is automatically saved and another loaded for the service routine. This allows each service routine to be executed in the appropriate mode without affecting the status of the interrupted task. Also, vector tables can be provided for both maskable and nonmaskable interrupts when in mode 3.

Interrupt mode 3 is always used for processing interrupts from the Z280 MPU's on-chip peripherals, regardless of which mode is selected for the external interrupt requests.

Table 6-2 summarizes interrupt processing for all four modes.

Table 6-2. Interrupt Modes

Interrupt Mode	Interrupt Type	Saved Status Information	Effect on MSR	Effect on PC
0	Nonmaskable	PC	System mode, Single-Step and interrupts disabled	Set to 66_H
0	Maskable	*	"	*
1	Nonmaskable	PC	"	Set to 66_H
1	Maskable	PC	"	Set to 38_H
2	Nonmaskable	PC	"	Set to 66_H
2	Maskable	PC	"	Fetched from address formed by I register and interrupt vector
3	Nonmaskable	MSR, PC, and reason code	Fetched from Interrupt/ Trap Vector Table	Fetched from Interrupt/ Trap Vector Table
3	Maskable	MSR, PC, and reason code	"	"

*: Depends on instruction returned by interrupting device during acknowledge cycle.

6.3 TRAPS

The Z280 CPU architecture supports eight types of traps, all of which are generated internally in the MPU. The Privileged Instruction, System Call, Access Violation, and Division Exception traps cannot be disabled. I/O instructions can be specified as privileged instructions in the Trap Control register. The Extended Instruction, System Stack Overflow Warning, Single-Step, and Breakpoint-on-Halt traps can be selectively enabled or disabled in the Trap Control register and MSR.

Traps are processed by saving the current program status (PC and MSR) on the system stack and loading new program status from the Interrupt/Trap Vector Table, in a manner similar to interrupts using interrupt mode 3. The current interrupt mode has no effect on trap processing. Thus, the Interrupt/Trap Vector Table must be present in memory and the Interrupt/Trap Vector Table Pointer in the CPU must be initialized before executing any instruction that could generate a trap. Traps can occur only if executing Z280 MPU instructions that are not part of the Z80 CPU instruction set or if trap-generating features of the Z280 CPU (such as stack overflow warnings) have been explicitly enabled.

6.3.1 Extended Instruction Trap

The Extended Instruction trap occurs when the Z280 CPU encounters an extended instruction while the EPU Enable bit in the Trap Control register is a zero. For instructions that transfer data between an EPU and memory, the following information is pushed onto the system stack when processing the Extended Instruction trap: the address of the next instruction, the MSR, the address of the memory operand, and the address of the template portion of the extended instruction (in that order). For Load Accumulator from EPU and EPU Internal Operation instructions, the address of the next instruction, the MSR, and the address of the template in the extended instruction are saved. The PC and MSR values for the service routine are then loaded from the Interrupt/Trap Vector Table. The Interrupt/Trap Vector Table contains four different entries for Extended Instruction traps, one for each type of extended instruction.

The Extended Instruction trap allows the program to simulate (in software) the operation of an EPU in a trap service routine when no EPUs are present in the system.

6.3.2 Privileged Instruction Trap

The Privileged Instruction trap occurs when the Z280 CPU encounters a privileged instruction while in the user mode (the User/$\overline{\text{System}}$ bit in the MSR is set to 1). I/O instructions can be privileged instructions, depending on the contents of the Trap Control register. The following information is saved on the system stack when processing a Privileged Instruction trap: the address of the instruction causing the trap and the MSR (in that order).

The Privileged Instruction trap protects the operating system environment by preventing user mode programs from executing instructions that could disrupt the system.

6.3.3 System Call Trap

The System Call trap occurs whenever a System Call instruction is executed. The following information is saved on the system stack when processing a System Call trap: the address of the next instruction, the MSR, and the 16-bit immediate operand encoded in the System Call instruction (in that order).

The System Call trap provides a means by which a user mode program can request an operating system function, thereby allowing for an orderly transition between the user and system modes.

6.3.4 Access Violation Trap

The Access Violation trap occurs whenever the Z280 MPU's on-chip MMU detects an illegal memory access. Specifically, this trap occurs when the MMU's translation mode is enabled and either the address to be translated implies using a page descriptor register whose Valid bit is zero or the access is a write to a page whose Write-Protect bit is set to 1. The following information is saved on the system stack when processing an Access Violation trap: the address of the instruction causing the trap and the MSR (in that order). Information about the logical address that caused the fault is saved in the MMU (see Chapter 7).

The Access Violation trap facilitates the implementation of virtual memory systems using the Valid bit in the page descriptor registers and allows information in memory to be write-protected.

6.3.5 System Stack Overflow Warning Trap

The System Stack Overflow Warning trap can occur only if the Stack Overflow Warning bit in the Trap Control register is set to 1. If so, then for each push to the system stack, the 12 most significant bits of the Stack Pointer are compared to the contents of the Stack Limit register and a trap is generated if they match. The following information is saved on the system stack when processing a System Stack Overflow Warning trap (but no second System Stack Overflow Warning trap is generated): the address of the next instruction and the MSR (in that order). The Stack Overflow Warning bit in the Trap Control register is automatically cleared to 0 when this trap occurs in order to prevent repeated traps.

The System Stack Overflow Warning trap notifies the operating system of potential stack overflow problems.

6.3.6 Division Exception Trap

The Division Exception trap occurs while executing a Divide instruction if the divisor is zero (divide by zero case) or the quotient cannot be represented in the destination precision (overflow case); the CPU flags are set to distinguish between these two situations (see the descriptions for the Divide instructions in Chapter 5). The following information is saved on the system stack when processing a Division Exception trap: the address of the Divide instruction and the MSR (in that order).

6.3.7 Single-Step Trap

Two control bits in the Master Status register are used to control Single-Step traps: the Single-Step bit (bit 8) and the Single-Step Pending bit (bit 9). The Single-Step trap occurs when the Single-Step Pending bit in the MSR is set to 1. To enter single-step mode, wherein a Single-Step trap is executed after each instruction, the Single-Step bit in the MSR is set to 1. At the beginning of instruction execution, the state of the Single-Step Pending bit is checked; if it is set, a Single-Step trap is executed. Then, the state of the Single-Step bit is copied into the Single-Step Pending bit and the instruction is executed. If the instruction generates another trap (such as a Privileged Instruction trap), that trap handling routine is executed before the Single-Step Pending bit is again checked and the Single-Step trap is processed. This execution sequence is illustrated in Figure 6-2. Note that once the Single-Step bit gets set, a Single-Step trap does not occur until after the next instruction, because the Single-Step Pending bit is checked before being loaded with the state of the Single-Step bit. Single-Step traps are then executed after each instruction until the Single-Step bit in the MSR is cleared to 0.

The Single-Step Pending bit in the MSR is automatically cleared by a Division Exception, Access Violation, Privileged Instruction, or Breakpoint-on-Halt trap, so that the saved MSR value put on the stack as a result of trap processing will have a 0 in bit position 9. For each of those trap types, the address of the

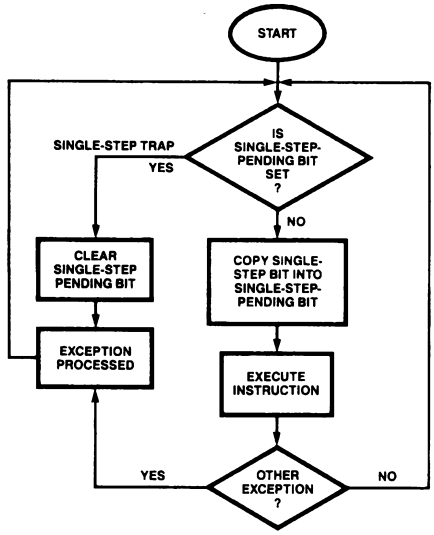


Figure 6-2. Instruction Execution Sequence

actual trapping instruction is saved on the stack (as opposed to the address of the next instruction). The trapping instruction can be re-executed upon returning from the trap service routine, in which case another Single-Step trap is not desired before instruction execution. Similarly, the Single-Step Pending bit is automatically cleared by a Single-Step trap, to ensure that only one Single-Step trap occurs per instruction.

When executing a Return From Interrupt Long (RETIL) instruction to return from an interrupt or trap service routine, the Single-Step Pending bit in the MSR for the interrupted program is the OR of the Single-Step Pending bit in the MSR of the service routine and the Single-Step Pending bit in the MSR value that was saved during trap processing. Thus, if the service routine was being executed in single-step mode, a Single-Step trap occurs after execution of the RETIL instruction, before resumption of the interrupted program.

The following information is saved on the system stack when processing a Single-Step trap: the address of the next instruction and the MSR (in that order).

The Single-Step trap facilitates the debugging of Z280 CPU code. The following text explains four methods for entering single-step operations.

a. PUSH a PC value for the instruction you wish to jump to.
   PUSH an MSR value with the desired combination of the Single-Step (SS) and Single-Step Pending (SSP) bits.
   Execute and RETIL instruction.

b. Execute a LDCTL instruction with the desired combination of the SS and SSP bits.

c. Execute a System Call (SC) with an identifier that you reserve for a single-step entry.
   POP the identifier and branch to the remaining single-step code routine.
   POP the MSR.
   Set the desired combinations of SS and SSP.
   PUSH the new MSR.
   Execute the RETIL instruction.

This method can be used only in the User Mode of operation.

d. Use the "Breakpoint-on-Halt" trap by substituting a HALT opcode for the first byte of an instruction where single-stepping is to start. The trap service routine should look something like this:

   POP the MSR.
   Set the desired combinations of SS and SSP.
   PUSH the MSR.
   Restore the instruction byte that the HALT opcode replaced.
   Execute the RETIL instruction.

   Both interrupt and trap routines can be single-stepped by setting the appropriate SS and SSP combination in the MSR entry in the Interrupt/Trap Vector Table.

   Instructions that cause a trap but will be re-executed (ie: privileged, divide, page fault) automatically clear the SSP bit in the PUSHed MSR. This ensures that only one single-step trap will occur for these instructions.

Table 6-3. Trap Types

Trap Type	Can be Disabled	Status Saved
Extended Instruction	Yes	Address of next instruction MSR value Address of operand in memory (if applicable) Address of EPU template
Privileged Instruction	No	Address of instruction causing trap MSR value
System Call	No	Address of next instruction MSR value 16-bit reason code from SC instruction
Access Violation	No	Address of instruction causing trap MSR value
System Stack Overflow	Yes	Address of next instruction MSR value
Division Exception	No	Address of instruction causing trap MSR value
Single-Step	Yes	Address of next instruction MSR value
Breakpoint-on-Halt	Yes	Address of Halt instruction MSR value

Table 6-4. Interrupt Acknowledge Encoding for Z80 Bus Configuration

AD_2	AD_1	Interrupt Being Acknowledged
0	0	Interrupt A
0	1	Nonmaskable Interrupt
1	0	Interrupt B
1	1	Interrupt C

6.3.8 Breakpoint-on-Halt Trap

The Breakpoint-on-Halt trap occurs if a Halt instruction is encountered while the Breakpoint-on-Halt Enable bit in the MSR is set to 1. The following information is saved on the system stack when processing a Breakpoint-on-Halt trap: the address of the Halt instruction and the MSR (in that order).

The Breakpoint-on-Halt trap provides a breakpoint facility that is useful in debugging environments in which breakpoints on instruction boundaries are desired.

The trap types and the status saved during the processing of each trap are summarized in Table 6-3.

6.4 INTERRUPT AND TRAP HANDLING

The Z280 CPU response to an interrupt request or trap condition consists of up to five steps: acknowledging the external request (externally-generated interrupts only), saving current program status, loading new program status, executing the service routine, and returning to the interrupted program. Interrupts are accepted and processed between instructions, with the exception of the block move, search, and I/O instructions, which can be interrupted between any iteration. Traps are detected during instruction execution, with the exception of the Single-Step trap, as described previously. Thus, a trap condition is processed before handling any pending interrupts.

6.4.1 Interrupt Acknowledge

An interrupt acknowledge bus transaction is required only for externally-generated interrupts. The main effect of the interrupt acknowledge is to establish communication between the requestor and the Z280 CPU.

For Z80 Bus configurations of the Z280 MPU, the type of interrupt being acknowledged is indicated on bus lines AD_1 and AD_2 while the Address Strobe is being asserted during the interrupt acknowledge cycle, as per Table 6-4.

For the Z80 Bus configurations of the Z280 MPU, no external acknowledge cycle is generated for nonmaskable interrupts in interrupt modes 0, 1, and 2, or for maskable interrupts in interrupt mode 1. For maskable interrupts in interrupt modes 0, 2, and 3, and for nonmaskable interrupts in mode 3, 8-bit data is read from the AD_0-AD_7 bus lines during the acknowledge cycle; this data is used as dictated by the interrupt mode in effect, as described in section 6.2. For maskable interrupts in interrupt mode 0, successive bytes are read on AD_0-AD_7 until a complete instruction has been fetched, via repetition of the acknowledge cycle.

For Z-BUS configurations of the Z280 MPU, any interrupt from an external source is acknowledged. The type of interrupt being acknowledged is indicated by the ST_0-ST_3 status lines during the acknowledge cycle. A word of data is read from the address/data bus during the acknowledge cycle and used as dictated by the interrupt mode in effect. For interrupt modes 2 and 3, the lower byte of this data is used as the interrupt vector. For maskable interrupts in interrupt mode 0, successive bytes are read on AD_0-AD_7 until a complete instruction has been fetched, via repetition of the acknowledge cycle.

Acknowledge cycles are always executed in system mode, regardless of the mode of the interrupted program. The MSR of the interrupted program is not affected by this change in mode. The CPU stays in system mode until the start of execution of the service routine. In interrupt modes 0, 1, and 2, the service routine starts in system mode; in interrupt mode 3, the MSR of the service routine is determined by the contents of the Interrupt/Trap Vector Table.

Interrupt requests from the on-chip peripherals never generate an acknowledge cycle and are always processed using interrupt mode 3. Similarly, traps do not generate acknowledges.

6.4.2 Status Saving

During exception processing, the status of the interrupted program is saved on the system stack. In interrupt mode 0, the Program Counter is automatically saved when processing nonmaskable interrupts; the instruction returned by the peripheral device will determine what status information is saved when processing maskable interrupts. For interrupts in interrupt mode 1 or 2, the Program Counter is automatically saved. For interrupts in interrupt mode 3, the Program Counter and MSR of the interrupted task are saved, followed by the "reason code" (Figure 6-3). For external interrupt requests, the reason code is the value read from the data bus during the interrupt acknowledge cycle; the upper byte of the reason code is all zeros for 8-bit data bus (Z80 Bus) configurations of the Z280 MPU. For interrupts from the on-chip peripherals, the reason code is the offset address in the Interrupt/Trap Vector Table that corresponds to the MSR value entry for that interrupt type.

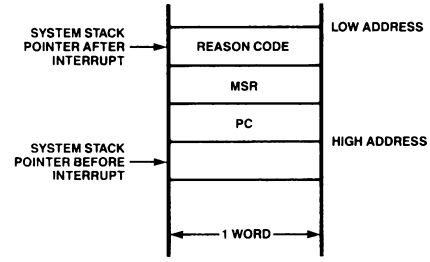


Figure 6-3. Format of Saved Status on System Stack Due to a Mode 3 Interrupt

The Program Counter value saved during interrupt processing is the address of the next instruction in the interrupted routine, except for interrupts during block move, block search, and block I/O instructions. The block instructions can be interrupted between any one iteration of their operation, in which case the PC value saved is the address of the block instruction itself.

The status saved as a result of a trap depends on the type of trap being executed, as noted in Figure 6-3. The PC and MSR values are always saved during trap processing, along with other trap-dependent information.

If any memory write operation involved in saving status information during interrupt or trap processing causes a memory access violation, a special "fatal condition" is entered, as described in section 6.6.

6.4.3 Loading New Program Status

After saving the status of the interrupted program, new program status values (i.e., new values for the PC and MSR) are automatically loaded, in accordance with the interrupt mode and any data read during the acknowledge cycle. This new program status determines the operating modes and starting address of the service routine.

For externally generated interrupts in interrupt modes 0, 1, and 2, the Master Status register is automatically modified to specify system mode with the Single-Step trap and all maskable interrupts disabled. For externally generated interrupts in interrupt mode 3, all internally generated interrupts, and all traps, the new MSR value is loaded from the Interrupt/Trap Vector Table.

For externally generated maskable interrupts processed using interrupt mode 0, the first instruction of the service routine is supplied by the interrupting device. This must be a Call or Restart instruction that loads the PC with the starting address of the service routine. For nonmaskable interrupts in interrupt mode 0, the PC is set to 0066_H, and all maskable interrupts are automatically disabled.

In interrupt mode 1, the PC is set to 0038_H for externally generated maskable interrupts and to 0066_H for nonmaskable interrupts.

For externally generated maskable interrupts in interrupt mode 2, the PC is fetched from an Interrupt Vector table in system data memory; the logical address of the fetched PC value is formed by concatenating the contents of the I register with the 8-bit vector returned by the interrupting device during the acknowledge cycle. For nonmaskable interrupts, the PC is set to 0066_H.

For externally generated interrupts in interrupt mode 3, all internally generated interrupts, and all traps, the PC and MSR values for the service routine are fetched from the Interrupt/Trap Vector Table (see section 6.5). The new value for the MSR is at a fixed location in this table. Externally generated interrupts can be vectored or nonvectored in interrupt mode 3, as determined by the contents of the Interrupt Status register. For nonvectored interrupts and all traps, the new PC value is at a fixed location in the Interrupt/Trap Vector Table; for vectored interrupts, the location of the new PC in the table is dependent on the 8-bit vector read during the acknowledge cycle.

The value loaded into the Program Counter during exception processing is a logical address that can be translated to a physical address by the MMU when the CPU fetches the first instruction of the service routine.

6.4.4 Executing the Service Routine

In interrupt mode 0, the interrupting device provides the Restart or Call instruction that begins the service routine; this instruction saves the Program Counter value of the interrupted routine and provides the address of the service routine. In the other interrupt modes and for traps, the starting address of the service routine is determined automatically during interrupt processing, as described in the preceding section. This program is now executed.

For externally generated interrupts in interrupt modes 0, 1, and 2, all maskable interrupts are automatically disabled; therefore the service routine is protected from additional interrupts until the MSR is altered via a Load Control, Enable Interrupt, Return from Nonmaskable Interrupt, or Return from Interrupt Long instruction. Interrupts in mode 3 and all traps cause a new MSR to be loaded from the Interrupt/Trap Vector Table; the value of this MSR determines which interrupts are enabled during the service routine. Service routines that enable interrupts before exiting permit interrupts to be handled in a nested fashion.

6.4.5 Returning from a Service Routine

Three different instructions are available for returning from an interrupt or trap service routine: Return from Nonmaskable Interrupt, Return from Interrupt, and Return from Interrupt Long. All three are privileged instructions, since they must retrieve values from the system stack.

The Return from Nonmaskable Interrupt (RETN) instruction is used to return from nonmaskable interrupts in interrupt modes 0, 1, and 2. This instruction pops the word on the top of the stack into the Program Counter, restoring the Program Counter value present before the interrupt, and loads the Interrupt Request Enable bits in the MSR with the contents of the Interrupt Shadow register.

The Return from Interrupt (RETI) instruction is used to return from externally generated maskable interrupts in interrupt modes 0, 1, and 2. This instruction pops the word on the top of the stack into the Program Counter, which restores the Program Counter value present before the interrupt. The RETI instruction also causes a special bus

transaction that fetches this instruction from external memory (regardless of whether it is contained in the on-chip cache), with the appropriate bus control and status signals to indicate that an instruction fetch is occurring; this is used to reset the interrupt logic of the Z80 family peripherals.

The Return from Interrupt Long (RETIL) instruction is used to return from interrupts in interrupt mode 3 and all traps, since it causes both the MSR and PC values to be popped from the stack. If this instruction is used to return from an interrupt processed with another interrupt mode (e.g., if RETIL is used to return from a mode 2, instead of a mode 3, interrupt), an MSR value must be pushed onto the stack in the service routine prior to execution of the RETIL. For interrupts in interrupt mode 3 and all traps, the service routine must pop the reason code or other trap-dependent information off the stack before executing RETIL. Unlike RETI, RETIL causes no special bus activity and, therefore, cannot be used to automatically reset Z80 family peripherals.

6.5 INTERRUPT/TRAP VECTOR TABLE

During interrupt processing under interrupt mode 3 and all trap processing, the PC and MSR values that determine the starting location and operating modes of the appropriate service routine are fetched from a table in memory called the Interrupt/Trap Vector Table. This table holds an MSR and PC value for the service routine for every possible type of interrupt and trap. The particular values fetched from the table during exception processing are a function of the type of exception that occurred and, for vectored external interrupts, the vector returned by the peripheral during the acknowledge cycle. The format of the Interrupt/Trap Vector Table is given in Table 6-5. Each entry in the Interrupt/Trap Vector Table consists of two words--an MSR value followed by a PC value. If an external interrupt is vectored, as determined by the contents of the Interrupt Status register, the 8-bit vector returned by the peripheral is used as an index into a list of up to 128 possible PC values for the service routine; only even-valued vectors are supported by the Z280 CPU architecture. Thus, for a vectored interrupt, there is only one starting MSR value for all the possible service routines, but up to 128 potential PC values. The NMI and Interrupt A requests share the same vectors.

For example, suppose an interrupt is requested by the on-chip counter/timer 0. If that interrupt request is enabled (bit 1 in the MSR is set to 1), the interrupt is processed as follows: the current PC and MSR values are saved on the system stack; an identifier word with the value 14_H is saved on the system stack; a new value for the MSR is fetched from location 14_H in the Interrupt/Trap Vector Table; a new value for the PC is fetched from location 16_H in the Interrupt/Trap Vector Table; execution of the service routine is begun.

If an interrupt request is received from an external source on interrupt line A under interrupt mode 3 and that interrupt request is enabled (bit 0 in the MSR is set to 1), then interrupt processing proceeds as follows:

- An acknowledge cycle is executed, during which data is read from the external data bus.
- The current PC and MSR values are saved on the system stack
- The data read from the bus during the acknowledge cycle is saved on the system stack as the identifier word.
- A new value for the MSR is fetched from location 08_H in the Interrupt/Trap Vector Table
- A new value for the PC is fetched either from location 0A in the Interrupt/Trap Vector Table (if bit 13 of the Interrupt Status register is 0, indicating that Interrupt A is not vectored) or from the location in the Interrupt/Trap Vector Table found by adding the lower byte of the data read from the bus during the acknowledge cycle (the interrupt vector) to 70_H (if bit 13 of the Interrupt Status register is 1, indicating that Interrupt A is vectored).
- Execution of the service routine is begun.

For vectored interrupts, the interrupt vector returned during the acknowledge cycle must be even-valued in order to reference a valid PC value in the Interrupt/Trap Vector Table.

The Interrupt/Trap Vector Table Pointer register must be initialized to hold the most significant 12 bits of the starting physical address of the Interrupt/Trap Vector Table. The Interrupt/Trap Vector Table must start on a 4K byte boundary in physical memory (that is, a memory address whose 12 least significant bits are all zeros).

Table 6-5. Interrupt/Trap Vector Table Format

Address in Table (Hexadecimal)	Contents
00	Reserved
04	NMI vector
08	Interrupt line A vector
0C	Interrupt line B vector
10	Interrupt line C vector
14	Counter/Timer 0 vector
18	Counter/Timer 1 vector
1C	Reserved
20	Counter/Timer 2 vector
24	DMA channel 0 vector
28	DMA channel 1 vector
2C	DMA channel 2 vector
30	DMA channel 3 vector
34	UART receiver vector
38	UART transmitter vector
3C	Single-Step trap vector
40	Breakpoint-on-Halt trap vector
44	Division Exception trap vector
48	Stack Overflow Warning trap vector
4C	Access Violation trap vector
50	System Call trap vector
54	Privileged Instruction trap vector
58	EPU ← Memory Extended Instruction trap vector
5C	Memory ← EPU Extended Instruction trap vector
60	A ← EPU Extended Instruction trap vector
64	EPU Internal Operation Extended Instruction trap vector
68-6C	Reserved
70-16E	128 Program Counter values for NMI and interrupt line A vectors (MSR values from position 04 and 08 in this table, respectively)
170-26E	128 Program Counter values for interrupt line B (MSR value from position 0C in this table)
270-36E	128 Program counter values for interrupt line C (MSR value from position 10 in this table)

6.6 THE FATAL CONDITION

During interrupt and trap processing, the CPU automatically attempts to save status information about the interrupted program on the system stack. If the MMU is enabled, an access violation can occur during the status saving process if a write is attempted to an invalidated page or to a page that is write-protected. Detection of an access violation during the status saving process causes the Z280 CPU to enter a special fatal condition; the following steps are taken automatically when the fatal condition occurs: the current PC contents are written to the HL register, the current MSR contents are written to the DE register, all the Interrupt Request Enable bits in the MSR are cleared to 0, and the CPU enters a Halt state. This Halt state is identical to the Halt state caused by the execution of a Halt instruction, with one exception: a Halt state induced by a fatal condition can be exited only by a reset.

Chapter 7.
Memory Management Unit

7.1 INTRODUCTION

The Z280 MPUs include an on-chip paged Memory Management Unit (MMU), which allows the MPUs to address more than 64K bytes of physical memory. Memory management with the MMU involves two issues: memory allocation and memory protection. The allocation of memory is controlled by allowing the MMU to translate the 16-bit logical addresses from the Z280 CPU into the 24-bit physical addresses output by the MPU. Thus, a given programming task can be relocated to any area of physical memory, regardless of the logical addresses used by that task. During this translation process, the MMU also monitors the type of memory access being made; the MMU can inhibit accesses or write-protect memory areas, thereby allowing memory to be protected from unwanted or unintended modes of use.

The MMU partitions the 64K logical address space of the Z280 CPU into fixed-sized memory pages and maps those pages into the physical address space. Separate mapping facilities are available for the system and user modes of operation; translation can be performed in either one or in both modes. Optionally, the MMU provides for separating instruction fetches from data references, which allows the user to define up to four different logical address spaces: system mode program, system mode data, user mode program, and user mode data. If the program and data address spaces are separated, the MMU uses a page size of 8192 (8K) bytes; if not, the page size is 4096 (4K) bytes.

The MMU is programmed via I/O references to its control registers. The MMU records which pages have been modified and can inhibit the cache mechanism to prevent the writing of data to the on-chip cache. Access Violation traps are generated when an error condition is detected (such as an attempted write to a read-only page). Access violations cause the currently executing instruction to be aborted, and allow that instruction to be restarted in a manner compatible with virtual memory requirements. Upon reset, the MMU is disabled, allowing logical addresses to pass through to physical memory without translation.

7.2 MMU ARCHITECTURE

The Z280 MMU consists of two sets of 16 page descriptor registers, used to translate addresses and assign memory attributes on a page-by-page basis, and a Master Control register that governs MMU operation. There is one page descriptor register associated with each logical page of memory. One set of 16 page descriptor registers is dedicated to system mode operation and the other set to user mode operation. The MMU registers are accessed using I/O instructions.

When translation is enabled for a particular mode (system or user), as determined by the contents of the MMU Master Control register, the MMU translates memory addresses whenever the CPU is operating in that mode, using the set of page descriptor registers dedicated to that mode. However, there are two exceptions to that rule:

- When the CPU is fetching program status information from the Interrupt/Trap Vector Table in response to an interrupt under interrupt mode 3 or a trap, the Interrupt Trap Vector Table Pointer register is used to determine the physical address of the program status information.

- The Load in User Program (LDUP) and Load in User Data (LDUD) instructions are executed in system mode but use the user mode page descriptor registers to translate the data operand's address.

Memory addresses generated by the on-chip DMA channels are 24-bit physical addresses that are not translated by the MMU. Only memory addresses, and not I/O addresses, are translated by the MMU.

While an address is being translated, any attributes associated with the logical page containing that address are checked. The attributes for a page are determined by the contents of that page's page descriptor register. Pages can be write-protected and/or made non-cacheable using these attributes. A non-cacheable page is one whose contents cannot be copied into the on-chip cache during program execution; thus, accesses to loca-

tions in non-cacheable pages always use the external bus. This attribute is useful in multiprocessor systems with shared memory areas, where each processor must be able to access the most current version of the information in the shared memory area, or in systems with memory-mapped I/O devices. The MMU also maintains a status bit for each page, which indicates if that page has been modified.

Each page descriptor register contains a Valid bit, which indicates if that descriptor contains valid information. Attempts to access an address contained in a page with an invalid descriptor and attempts to write to an address in a page that is write-protected generate Access Violation traps. An Access Violation trap causes the currently executing instruction to be aborted, facilitating the development of virtual memory systems. A special I/O port in the MMU (Invalidation I/O port) is available for resetting the valid bits in a whole group of page descriptor registers with a single I/O instruction.

For system mode operation, user mode operation, or both, the MMU can be configured to separate instruction fetches from data fetches, therefore separating the program address space from the data address space. This allows a Z280 MPU program to contain up to 64K bytes of code and operate on up to 64K bytes of data. With the program/data separation mode in effect, the 16 page descriptor registers for that mode are partitioned into two sets of eight descriptors: one set for instruction fetches and one set for data fetches. An instruction fetch or data reference using the PC relative addressing mode is translated using the page descriptor registers associated with the program address space; data accesses using other addressing modes and accesses to the interrupt vector table under interrupt mode 2 use the page descriptor registers associated with the data address space. In this mode, pages are 8K bytes long. Two control bits in the MMU Master Control register specify independently whether program/data separation is in effect for system mode and whether program/data separation is in effect for user mode.

When translation is disabled for a particular mode (system or user), the MMU does not translate memory addresses or perform attribute checking while the CPU is operating in that mode. For a memory access when the MMU is disabled, the logical memory address passes through the MMU without translation to physical address outputs A_0-A_{15} and physical address outputs A_{16}-A_{23} are all zeros. When the MMU is disabled all memory is assumed to be both writeable and cacheable.

7.3 PAGE DESCRIPTOR REGISTERS

There are two sets of 16 page descriptor registers in the MMU, one set for system mode operation and one set for user mode operation. Each page descriptor register is 16 bits long, consisting of a 12-bit page frame address field and a 4-bit attribute field (Figure 7-1).

Figure 7-1. Page Descriptor Register

The page frame address field contains the most significant 12 bits (if program/data separation is not in effect) or most significant 11 bits (if program/data separation is in effect) of the starting physical address for that page. The low-order bits of the page's base physical address are assumed to be all zeros; thus, pages always start on 4K byte boundaries in physical memory without program/data separation, or 8K byte boundaries with program/data separation.

The least significant four bits of each page descriptor register are attribute and status bits for that page, as described below:

Modified Bit (M). This status bit is automatically set to 1 whenever a write is successfully performed to a logical address in the page; it can be cleared to 0 only by writing to the page descriptor register via a software command. If the Valid bit is 0, the contents of this bit are undefined.

Cacheable Bit (C). When this bit is set to 1, information from the page can be stored in the on-chip cache memory. When this bit is cleared to 0, the cache control mechanism is inhibited from retaining a copy of information from the page.

Write-Protect Bit (WP). When set to 1, write operations to addresses in the page generate an Access Violation trap and the write is inhibited. When this bit is cleared to 0, all valid accesses to the page are allowed.

Valid Bit (V). This bit is set to 1 to indicate that the page descriptor register contains valid information about the page. When cleared to 0, all accesses to addresses in the page are inhibited and generate Access Violation traps.

7.4 ADDRESS TRANSLATION

If address translation is enabled, logical addresses are translated to physical addresses in one of two ways, depending on the program/data separation mode, as specified in the MMU Master Control register. The format of the page descriptor registers is independent of which mode is in effect.

7.4.1 Address Translation Without Program/Data Separation

When program/data separation is not in effect, the 16-bit logical address from the CPU is divided into two fields, a 4-bit index field used to select one of the 16 page descriptor registers, and a 12-bit offset field that forms the lower 12 bits of the resulting physical address. The upper 12 bits of the physical address are provided by the page frame address field of the selected page descriptor register. The pages are 4K bytes long. This translation mechanism is illustrated in Figure 7-2. Page descriptor register 0 is the descriptor for logical addresses 0000_H to $0FFF_H$, page descriptor register 1 is the descriptor for logical addresses 1000_H to $1FFF_H$, and so on. Thus, the index portion of the logical address selects the page descriptor register. The page frame address field of that page descriptor register then determines the actual starting address for that page in physical memory; the low-order 12 bits of the logical address specify the offset within that 4K byte page.

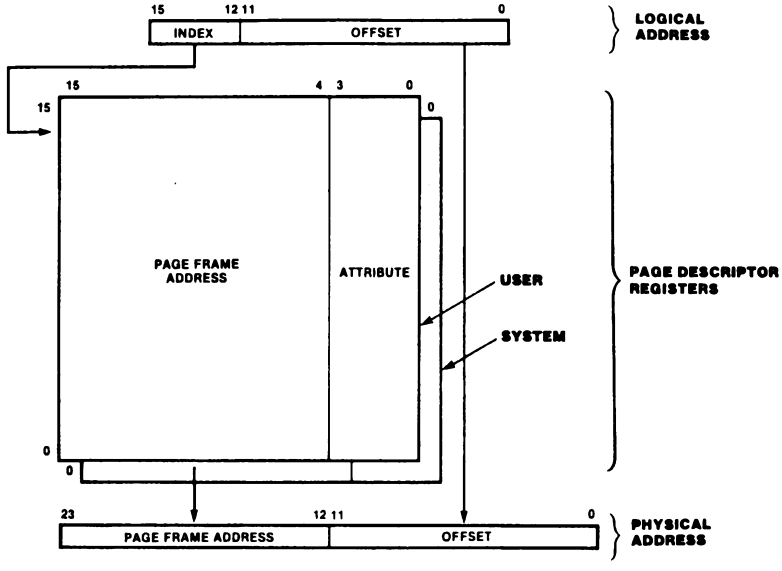


Figure 7-2. Address Translation without Program/Data Separation

7.4.2 Address Translation With Program/Data Separation

When program/data separation is in effect, the 16-bit logical address from the CPU is divided into a 3-bit index and a 13-bit offset. A Program/Data address control signal from the CPU becomes the most significant bit of the 4-bit index that selects the appropriate page descriptor register; the three most significant bits of the logical address form the least significant bits of this index. The upper 11 bits of the page frame address field in the selected page descriptor register provide the upper 11 bits of the resulting physical address. The least significant 13 bits of the logical address form the low order 13 bits of the physical address, as illustrated in Figure 7-3. Page descriptor register 0 is the descriptor for logical addresses 0000_H-$1FFF_H$ in the data address space, Page descriptor register 1 is the descriptor for logical addresses 2000_H-$3FFF_H$ in the data address space, and so on through page descriptor register 7; page descriptor register 8 is the descriptor for logical addresses 0000_H-$1FFF_H$ in the program address space, page descriptor register 9 is the descriptor for logical addresses 2000_H-$3FFF_H$ in the program address space, and so on. Thus, each page is 8K bytes long, where the starting address of the page in physical memory is determined by the page frame address field in the selected page descriptor register, and the 13 least significant bits of the logical address specify the offset within that 8K byte page. In this mode, the least significant bit of the page frame address field in each page descriptor register is not used; this bit is modified by translation, and values read from it are unpredictable.

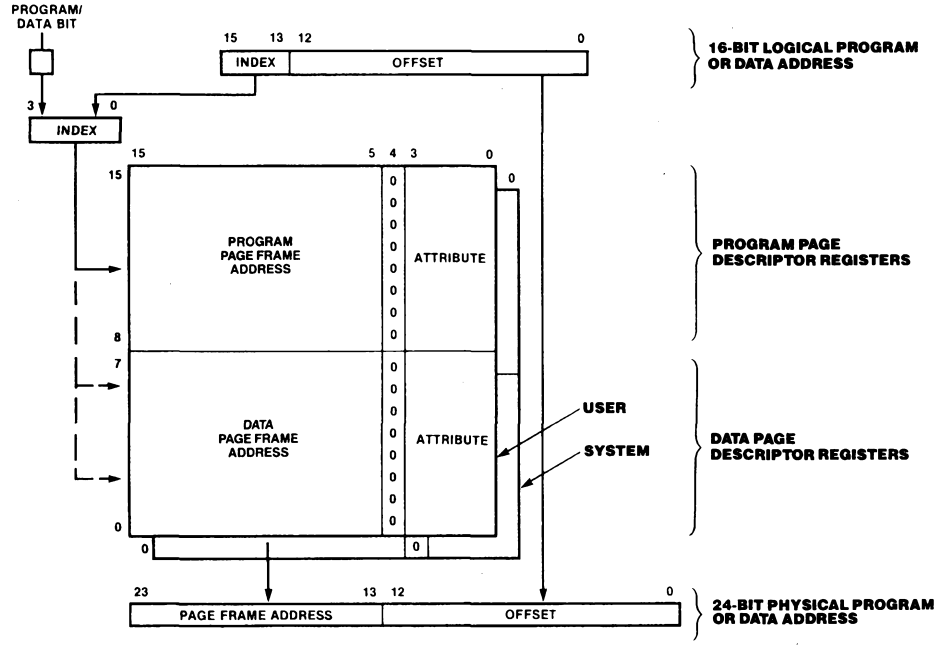


Figure 7-3. Address Translation with Program/Data Separation

7.5 MMU CONTROL REGISTERS

Besides the two sets of 16 page descriptor registers, the MMU contains a Master Control register and a Page Descriptor Register Pointer. The 16-bit Master Control register controls the operation of the MMU; the 8-bit Page Descriptor Register Pointer is used to select a particular page descriptor register during I/O accesses to the descriptors.

The 16-bit MMU Master Control register is shown in Figure 7-4. This register consists of four control bits and a 5-bit status field; the fields in this register are described below:

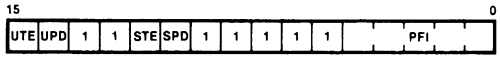


Figure 7-4. MMU Master Control Register

User Mode Translate Enable (UTE). When this bit is set to 1, logical memory addresses generated during user-mode operation are translated to physical addresses with attribute checking. When this bit is cleared to 0, the logical addresses are passed through the MMU to the address outputs with zeros in the most significant bits and no attribute checking or modified bit setting is performed.

User Mode Program/Data Separation Enable (UPD). When this bit is set to 1, instruction fetches and data accesses using the PC Relative addressing mode use user-mode Page Descriptor registers 8 through 15, and data references using other addressing modes use user-mode Page Descriptor registers 0 through 7; the page size is 8K bytes. When this bit is cleared to 0, both instruction and data fetches use user-mode Page Descriptor registers 0 through 15 and the page size is 4K bytes.

System Mode Translate Enable (STE). When this bit is set to 1, logical memory addresses generated during system-mode operation are translated to physical addresses with attribute checking. When this bit is cleared to 0, the logical addresses are passed through the MMU to the address outputs with zeros in the most significant bits and no attribute checking or modified bit setting is performed.

System Mode Program/Data Separation Enable (SPD). When this bit is set to 1, instruction fetches and data accesses using the PC Relative addressing mode use system-mode Page Descriptor registers 8 through 15, and data references using other addressing modes use system-mode Page Descriptor registers 0 through 7; the page size is 8K bytes. When this bit is cleared to 0, both instruction and data fetches use system-mode Page Descriptor registers 0 through 15 and the page size is 4K bytes.

Page Fault Identifier (PFI) Field. This 5-bit status field latches an identification number that indicates which Page Descriptor register was being accessed when an access violation was detected. The encoding used is given in Table 7-1.

The MMU Master Control register is programmed via a word output instruction to I/O port address $FFxxF0_H$ (where "x" indicates a "don't care") and is read via a word input instruction to that same port. A reset clears this register to all zeros, thereby disabling address translation and attribute checking in the MMU. Bits 5 through 9, 12, and 13 in this register are not used.

The Page Descriptor registers in the MMU are accessed using the Page Descriptor Register Pointer (PDR Pointer). The 8-bit PDR Pointer contains the address of one of the Page Descriptor registers; the encoding is given in Table 7-1. The permissible contents of the PDR Pointer are 00_H through $1F_H$. The PDR Pointer is accessed via byte I/O instructions to port address $FFxxF1_H$.

Table 7-1. Page Descriptor Register Addresses

PDR Pointer or PFI Field	Selected Page Descriptor Register
00	User Page Descriptor 0
01	User Page Descriptor 1
•	•
•	•
•	•
0E	User Page Descriptor 14
0F	User Page Descriptor 15
10	System Page Descriptor 0
11	System Page Descriptor 1
•	•
•	•
•	•
1E	System Page Descriptor 14
1F	System Page Descriptor 15

7.6 ACCESSING PAGE DESCRIPTOR REGISTERS

Data is read or written to the Page Descriptor registers via I/O instructions. Three different types of accesses are allowed, each of which is implemented with its own unique I/O port address.

7.6.1 Descriptor Select Port

Moves of one word of data to or from a Page Descriptor register are accomplished through I/O port address $FFxxF5_H$, the Descriptor Select Port. The Page Descriptor register accessed is the one addressed by the PDR Pointer; the PDR Pointer itself is unaffected. Any word I/O instruction can be used.

7.6.2 Block Move Port

Block moves of data into and out of Page Descriptor registers are accomplished by word accesses to I/O port address $FFxxF4_H$. The Page Descriptor register accessed is the one addressed by the PDR Pointer. Any word I/O instruction can be used. After the access, the contents of the PDR Pointer are automatically incremented by one; thus, a single block I/O instruction can be used to access several successive Page Descriptor registers. For example, if the PDR Pointer is initialized to 00, the execution of an INIRW instruction to I/O port $FFxxF4_H$ causes data from successive Page Descriptor registers starting with user Page Descriptor register 0 to be loaded into memory.

For accesses to the Page Descriptor registers using the Descriptor Select port or the Block Move port, the permissible contents of the PDR Pointer are the addresses for the Page Descriptors given in Table 7-1: 00_H to $1F_H$. Execution of an I/O instruction to ports $FFxxF4_H$ or $FFxxF5_H$ when the contents of the PDR Pointer are outside of this permitted range will have unpredictable results.

7.6.3 Invalidation Port

The Valid bits in the Page Descriptor registers can be cleared to 0 via byte writes to I/O port address $FFxxF2_H$, thereby invalidating the contents of the Page Descriptor registers. Individual Valid bits can subsequently be set by writing to individual Page Descriptor registers using the Descriptor Select port or the Block Move port. The Page Descriptor registers invalidated by a write to port $FFxxF2_H$ depend on the data written to that port, as delineated in Table 7-2. When writing to the invalidation port only the least significant four bits are sampled; the upper four bits are not used. Reading port $FFxxF2_H$ returns unpredictable data.

Table 7-2. MMU Invalidation Port

Data Written to Port FFxxF2 (Hexadecimal)	Page Descriptor Registers Invalidated
01	System Page Descriptor Registers 0–7
02	System Page Descriptor Registers 8–15
03	System Page Descriptor Registers 0–15
04	User Page Descriptor Registers 0–7
08	User Page Descriptor Registers 8-15
0C	User Page Descriptor Registers 0–15

The I/O port addresses for the MMU registers are listed in Table 7-3.

Table 7-3. I/O Port Addresses for MMU Control Registers

Port Address	Register
$FFxxF0_H$	Master Control Register
$FFxxF1_H$	Page Descriptor Register Pointer
$FFxxF5_H$	Descriptor Select Port
$FFxxF4_H$	Block Move Port
$FFxxF2_H$	Invalidation Port

Changing an MMU control register or Page Descriptor register does not cause a flush of the CPU instruction pipeline. While an instruction that changes an MMU register is executing, up to two subsequent instructions can be pre-fetched into the CPU pipeline; execution of these subsequent instructions must have benign results. In other words, when changing an MMU register, up to two subsequent instructions can be fetched before the change to the MMU register is guaranteed to take effect. (However, no data accesses are pre-fetched.) Therefore, when initially enabling the MMU for address translation, the instruction that enables the MMU and the next two instructions must be in a page whose logical addresses are identical to physical addresses (so that it is immaterial exactly when the MMU begins the translation process for those instruction fetches). When altering a page descriptor register while translation is enabled, neither of the next two instructions should reside in the page associated with the Page Descriptor register being changed.

7.7 INSTRUCTION ABORTS

Detection of a page fault (due to an attempted access to an invalidated page) or a write-protect violation (due to an attempted write to a write-protected page) causes the currently executing instruction to be immediately aborted and generates an Access Violation trap. The starting address of the instruction that caused the violation and the current MSR value are automatically saved on the system stack when processing an Access Violation trap. Furthermore, the MMU latches the address of the referenced Page Descriptor register in the PFI field of the MMU Master Control register whenever a violation occurs.

For most instructions, the CPU registers are not modified during the execution of aborted instructions; i.e., their contents are the same as before the aborted instruction began. The exceptions are the block move, block search, and block I/O instructions; when aborted, the CPU registers are the same as just before the iteration of the instruction in which the violation occurred. In either case, no modification of CPU registers is necessary before restarting the aborted instruction.

The instruction abort mechanism of the Z280 MPU facilitates the implementation of virtual memory in Z280-based systems. In a virtual memory system, a cleared Valid bit in the Page Descriptor register can be used to indicate when a memory page is not currently mapped into main memory. If an access is attempted to such a page, the instruction is aborted and the Access Violation trap service routine is invoked. The service routine can determine which Page Descriptor register is involved by reading the PFI field of the MMU Master Control register, swap the appropriate page from the secondary storage device into main memory, adjust the appropriate Page Descriptor registers, and then restart the aborted instruction. The aborted instruction is automatically restarted by using the Return from Interrupt Long instruction to retrieve the original PC and MSR values from the system stack. No adjustments to other CPU registers are required. During the swapping process, the modified status bit in the page descriptor register can be used to determine if a page has been modified since the last time it was copied to a secondary storage device.

Chapter 8.
On-Chip Memory

8.1 INTRODUCTION

The Z280 MPU has 256 bytes of on-chip memory. This on-chip memory can operate in either of two modes, as determined by the contents of the Cache Control register (see Chapter 3). In one mode, the on-chip memory is dedicated to fixed physical memory locations; the memory addresses that are mapped into the on-chip memory are determined under program control. In the other mode, the on-chip memory acts as a cache for either instructions, data, or both. When acting as a cache, the set of memory locations mapped into the on-chip memory at a given time is determined by the action of the executing program; the memory locations that were most recently accessed are stored in the cache. Memory accesses to locations mapped into the on-chip memory do not generate external bus transactions and, therefore, are faster than accesses to external memory; thus, use of the on-chip memory leads to faster, more efficient program execution. On reset, the on-chip memory is automatically enabled for use as a cache for instructions only.

8.2 CACHE MEMORY MODE

If the M/$\overline{C}$ bit in the Cache Control register is cleared to 0, then the 256 bytes of on-chip memory are treated as a cache. Cache memories are small, high-speed memory buffers situated between the processor and main memory. (Main memory is the semiconductor memory accessed via bus transactions.) For each memory access, control logic in the MPU checks if the memory location involved is currently stored in the cache. If so, the access is made to the cache, usually without generating an external bus transaction; if not, the access is made to main memory and the contents of the cache may be updated.

Z280 MPU cache organization is illustrated in Figure 8-1. The cache is arranged as 16 lines of 16 bytes each. Each line of the cache can hold a copy of sixteen consecutive bytes of memory in physical memory locations whose 20 most significant address bits are identical. Thus, for example, one line of the cache could hold the data from physical memory locations 153820_H to $15382F_H$. The 20 bits of physical address associated with one line of 16 bytes in the cache is called the tag address for that line. Each line of the cache also has 16 valid bits associated with it; each byte in the line is associated with one valid bit. The valid bit is used to indicate if the corresponding byte in the cache holds a valid copy of the memory contents at the associated physical memory location.

Lines in the cache are allocated using a Least-Recently Used (LRU) algorithm. If a read access is made to a physical memory address not currently stored in the cache (a cache "miss"), and the MMU does not assert cache inhibit, the line in the

	20 BITS	16 BITS	16 x 8 BITS
LINE 0	TAG 0	VALID BITS	CACHE DATA
LINE 1	TAG 1	VALID BITS	CACHE DATA
LINE 2	TAG 2	VALID BITS	CACHE DATA
•	•	•	•
•	•	•	•
•	•	•	•
LINE 15	TAG 15	VALID BITS	CACHE DATA

Tag n = the 20 Address bits associated with line n
Valid bits = 16 bits that indicate which bytes in the cache contain valid data
Cache data = 16 bytes

Figure 8-1. Cache Organization

cache that has been least recently accessed is selected to hold the newly read data. All bytes in the selected line are marked invalid except for the bytes containing the newly accessed data. A cache miss on a data write does not cause a line to be allocated to the memory location accessed.

On a cache miss during a memory read, one or two bytes (depending on the bus size) are fetched from main memory and written to the cache. The cache does not prefetch beyond the currently requested byte or word, with one exception; if burst mode operations are specified in the Cache Control register, burst mode transactions are used when fetching instructions.

The cache can be configured to hold only instructions, only data, or both instructions and data, as determined by the contents of the Cache Control register. If the cache contains data, writes to data at locations in the cache also generate external bus transactions to update the appropriate memory locations; thus, external memory is always guaranteed to contain valid information.

Tables 8-1 and 8-2 summarize cache operation. Whether or not a given memory operation accesses the cache depends on a number of factors: the type of access being made (program read, data read, or data write), whether the cache is enabled for that type of access, the type of instruction being executed, whether the MMU asserts cache inhibit, and whether the CPU or a DMA device initiates the transaction. The Cache Control register determines if the cache is used for instruction fetches or data accesses or both. Execution of the Test and Set (TSET) instruction, Return from Interrupt (RETI) instruction, and the extended instructions force external bus transactions, regardless of the contents of the cache, as described below. If the MMU is enabled, the access can be cacheable or noncacheable, as determined by the contents of the page descriptor register in use. If the MMU is not enabled, all transactions are considered to be cacheable. Both the CPU and on-chip DMA channels can access the cache. For DMA operations, only data read and data write transactions can occur. The state of the Cache Data Disable control bit in the Cache

Table 8-1. CPU Accesses to On-Chip Memory as Cache

Operation	Hit/Miss	Cache Instruction	Cache Data	Cache Activity Contents	Cache Activity LRU	Bus Transaction	Cache/Memory Supplies Information
MMU Cache Inhibit → Cacheable Transaction							
Instruction Read	Hit	Disabled	Don't care	Updated	No change	Yes	Memory
		Enabled	Don't care	No change	Updated	None	Cache
	Miss	Disabled	Don't care	Updated*	No change	Yes	Memory
		Enabled	Don't care	Updated	Updated	Yes	Memory
Data Read (non Test & Set)	Hit	Don't care	Disabled	Updated	No change	Yes	Memory
		Don't care	Enabled	No change	Updated	None	Cache
	Miss	Don't care	Disabled	Updated*	No change	Yes	Memory
		Don't care	Enabled	Updated	Updated	Yes	Memory
Data Read (Test & Set)	Don't care	Don't care	Don't care	Updated*	No change	Yes	Memory
Data Write	Hit	Don't care	Disabled	Updated	No change	Yes	—
		Don't care	Enabled	Updated	Updated	Yes	—
	Miss	Don't care	Disabled	No change	No change	Yes	—
		Don't care	Enabled	No change	No change	Yes	—
EPU-to-Memory	Don't care	Don't care	Don't care	Updated*	No change	Yes	EPU
Memory-to-EPU	Don't care	Don't care	Don't care	No change	No change	Yes	Memory
EPU Template	Don't care	Don't care	Don't care	No change	No change	Yes	Memory
RETI Opcode	Don't care	Don't care	Don't care	No change	No change	Yes	Memory
MMU Cache Inhibit → Noncacheable Transaction							
Don't care	Don't care	Don't care	Don't care	Updated*	No change	Yes	Memory

*Updated if a cache line contains the accessed location, otherwise unaffected.

Control register is ignored during DMA transactions; therefore, an on-chip DMA device always updates the cache contents during DMA write operations to memory locations that are currently mapped into the cache.

For read operations, a cache "hit" is a reference to a location with a valid entry in the cache, and a cache "miss" is a reference to a location that has no valid entry in the cache. In the general case (and assuming the transaction is cacheable), read operations that are cache hits cause the data to be read from the cache without generating an external bus transaction. Read operations that are cache misses cause the data to be read from the external memory via an external bus cycle and update the cache contents. Updating the cache contents may involve replacing the least-recently used line of the cache with a new line that contains the read location. For write operations, a cache hit is a write to a location in the cache, even if the destination byte is marked as invalid in the cache, and a cache miss is a write to a location that is not in the cache. Write operations that are cache hits cause both the cache and external memory to be updated, and write operations that are cache misses have no effect on the cache. Memory write operations always generate external bus transactions.

Exceptions to the above rules include the Test and Set, Return from Interrupt, and extended instructions. Data read operations during execution of a Test and Set instruction always read the data from the main memory with an external bus transaction, regardless of whether or not the location read is valid in the cache. This ensures that the most recent value for a semaphore is read from external memory in the case that the semaphore is in shared memory in a multiprocessor system; another processor may have changed the semaphore after it was last read into the MPU's cache.

If an RETI opcode is fetched from the cache, the instruction fetch cycles are repeated with external bus transactions; this ensures that Z80 family peripherals connected to the Z280 MPU with an interrupt request daisy chain can detect the RETI opcode fetch (a requirement for the proper operation of the Z80 family peripherals).

If extended instructions are resident in the cache, the EPU template portion of those instructions is always read using external bus transactions. This ensures that an Extended Processing Unit (EPU) that is monitoring the external bus can detect and read the template during those instruction fetch cycles. If the extended instruction results in a transfer of data between the EPU and memory, all the involved data transactions occur on the external bus. Cache hits during EPU-to-memory write transactions result in the updating of cache contents as well as external memory.

For memory reads, the LRU algorithm logic is updated to reflect that the associated cache line is the most-recently accessed line if the read was an instruction fetch in a cache enabled for instructions or a data fetch in a cache enabled for data. For data writes, the LRU algorithm logic is updated only for a cache hit in a cache that is enabled for data.

When the on-chip DMA controllers transfer data to memory, cache contents are modified if the write is to a location mapped into the cache, but the LRU algorithm is unaffected. EPU-to-memory transactions have the same effect. The cache is not affected by the activity of external DMA controllers.

On reset, all the valid bits in the cache are cleared to 0, marking all cache entries as invalid, and the on-chip memory is configured as a cache for instructions only.

Table 8-2. On-Chip DMA Accesses (Both Flowthrough and Flyby) Effect on On-Chip Memory as Cache

Memory Operation	Hit/Miss	Cache Instruction	Cache Data	Cache Activity Contents	Cache Activity LRU	Bus Transaction	Cache/Memory Supplies Information
Read	Hit	Don't care	Don't care	Updated	No change	Yes	Memory
	Miss	Don't care	Don't care	Updated*	No change	Yes	Memory
Write	Hit	Don't care	Don't care	Updated	No change	Yes	—
	Miss	Don't care	Don't care	No change	No change	Yes	—

*Updated if a cache line contains the accessed location, otherwise unaffected.

8.3 FIXED-ADDRESS MODE

When the M/$\overline{C}$ bit in the Cache Control register is set to 1, the on-chip memory is treated as fixed physical memory locations. Accesses to these memory locations never generate external bus transactions and, therefore, are faster than memory accesses that use the external bus (Table 8-3).

In this mode, the on-chip memory is still organized as 16 lines of 16 bytes each, with a 20-bit tag address that specifies the 16 physical memory locations in each line. All locations are assumed to contain valid information, whether or not they have been initialized; the individual valid bits associated with each byte in the line are ignored in this mode. The Cache Data Disable and Cache Instruction Disable bits in the Cache Control register are also ignored in this mode, and no distinction is made as to whether the CPU is accessing instructions or data.

Before entering this mode, the user must initialize the tag addresses for all 16 lines of on-chip memory. The values for these tags determine the 256 physical memory addresses that are mapped into the on-chip memory. This is accomplished by enabling the on-chip memory as a cache for data only, reading data from 16 physical memory locations that are in different cache lines, and then setting the M/$\overline{C}$ bit in the Cache Control register to 1 to enable the fixed-address mode for the on-chip memory. Altering the M/$\overline{C}$ bit in the Cache Control register does not affect the contents of the on-chip memory, including the tag addresses.

Note that each line of the on-chip memory must be assigned a unique tag address before entering this mode so that no unpredictable addresses are mapped into the on-chip memory. If instructions are to be fetched from the on-chip memory while in this mode, Return from Interrupt (RETI) instructions and the templates within extended instructions should never be resident in the on-chip memory; in each case, the operation of devices external to the MPU depends on these instructions being fetched with external bus transactions, as mentioned in section 8.2. Data to be transferred to or from an EPU cannot be resident in on-chip memory either, since this data must be transferred to the EPU over the external bus.

Table 8-3. DMA/CPU Accesses to On-Chip Memory as Fixed Memory Location

Memory Operation	Hit/Miss	Cache Instruction	Cache Data	Cache Activity		Bus Transaction	Cache/Memory Supplies Information
				Contents	LRU		
Read	Hit	Don't care	Don't care	No change	No change	No	Cache
	Miss	Don't care	Don't care	No change	No change	Yes	Memory
Write	Hit	Don't care	Don't care	Updated	No change	No	—
	Miss	Don't care	Don't care	No change	No change	Yes	—

Chapter 9.
On-Chip Peripherals

9.1 INTRODUCTION

The Z280 MPU features a number of peripheral devices on-chip in addition to the CPU, MMU, and cache memory. These peripheral devices include a clock oscillator, dynamic RAM refresh controller, four direct memory access (DMA) controllers, three counter/timers, and a universal asynchronous receiver/transmitter (UART).

The DMA channels, counter/timers, and UART are user-programmable devices that can be configured to operate in several different modes. These devices are accessed using I/O instructions; however, no external I/O bus transactions are generated when the on-chip peripherals are accessed by the CPU. These devices can generate interrupt requests to the Z280 MPU, as described below and in Chapter 6. Interrupts from these on-chip peripherals are always processed using interrupt mode 3, regardless of which interrupt mode is used for externally generated interrupts.

9.2 CLOCK OSCILLATOR

The Z280 MPU has an on-chip clock oscillator/generator that can be connected directly to a crystal or any other suitable clock source. The frequency of the processor clock is one-half of the frequency of the external clock source or crystal. The processor clock can be further divided by a factor of 1, 2, or 4 to provide the bus timing clock, as specified by the contents of the Bus Timing and Initialization register (see Chapter 3). The bus timing clock is output by the MPU for use by the rest of the system.

The on-chip clock oscillator, a high-gain amplifier, is enabled by either connecting a crystal across the Clock/Crystal Input (XTAL1) and Crystal Output (XTAL0) pins or connecting a clock input to the Clock/Crystal Input pin. The crystal must be a parallel resonant fundamental type.

9.3 REFRESH CONTROLLER

An on-chip memory refresh controller in the Z280 MPU is available for generating memory refresh operations in systems utilizing dynamic RAMs. Operation of this mechanism is controlled by the Refresh Rate register, which is located in the Z280 MPU's I/O address space. If enabled, memory refreshes are performed at a rate specified by the contents of this register.

The format of the 8-bit Refresh Rate register is shown in Figure 9-1. This register enables the refresh mechanism and determines the frequency of refresh transactions. The fields in this register are described below.

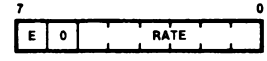


Figure 9-1. Refresh Rate Register

Refresh Enable (E) bit. When this bit is set to 1, the refresh mechanism is enabled. When this bit is cleared to 0, the refresh mechanism is disabled and refresh transactions are not generated.

Refresh Rate field. The contents of this 6-bit field determine the frequency of refresh transactions if the Refresh Enable bit is set to 1. A value of n ($0 < n < 63$) in this field specifies a refresh rate of once every 4n processor clock cycles; a value of 0 in this field indicates a refresh rate of every 256 processor clock cycles.

The Refresh Rate register is accessed via byte I/O operations to I/O port address $FFxxE8_H$ (where x means "don't care"). Bit 6 of this register is not used. On reset, the Refresh Rate register is initialized to 88_H, thereby enabling memory refresh at a rate of 32 processor clock cycles per refresh. This register can be read at any time to

determine if refresh is enabled and the current refresh rate.

A 10-bit refresh address is output on address lines A_0-A_9 during a refresh transaction. This refresh address is incremented by one for Z80 bus (8-bit data bus) configuration and by two for Z-BUS (16-bit data bus) configuration of the Z280 MPU between refresh transactions. The refresh address is not accessible by the programmer and is not affected by a reset.

During instruction execution, the actual refresh transactions are generated as soon as possible after the refresh period has elapsed. Generally, the refresh transaction is executed after the last clock cycle of the bus transaction in progress at the time that the refresh period elapsed. If the CPU receives an interrupt request during that same bus transaction, the refresh transaction is inserted before processing the interrupt. When the Z280 MPU does not have control of the bus due to a bus request, refresh transactions cannot be executed; while the MPU is in this state, internal circuitry records the number of refresh periods that have elapsed (that is, the number of "missed" refresh transactions). When the Z280 MPU regains control of the bus, the refresh mechanism automatically issues the missed refresh cycles. Similarly, if the refresh period elapses while the MPU is in a wait state (due to $\overline{WAIT}$ being asserted) during a bus transaction, the number of missed refresh transactions is recorded internally, and those refresh cycles are issued after $\overline{WAIT}$ is deactivated and the bus transaction is completed. The internal circuitry can record up to 256 such missed refresh operations.

Pseudo-static memories and some peripheral devices (such as the Z8000 family of peripherals) require a minimum transaction rate on the bus for correct operation. If the refresh mechanism is disabled by clearing the Refresh Enable bit in the Refresh Rate register, the rate field in this register is used to determine the minimum transaction rate on the bus. In this mode, if the refresh timer reaches 0 and no external bus transaction has occurred since the last time the refresh timer elapsed, then a refresh transaction will be generated. Thus, in a system that does not require memory refresh transactions, the Refresh Rate field in the Refresh Rate register must be initialized to an appropriate value even if memory refresh operations are disabled.

9.4 COUNTER/TIMERS

The Z280 MPU's three on-chip 16-bit counter/timers can be configured to satisfy a broad range of counting and timing applications, including event counting, interval timing, watchdog timing, and clock generation. Each counter/timer is composed of a 16-bit downcounter, a 16-bit time constant register, and two 8-bit control and status registers (the Counter/Timer Configuration register and the Counter/Timer Command/Status register). The three independent devices are referred to as counter/timer 0 (C/T 0), counter/timer 1 (C/T 1), and counter/timer 2 (C/T 2). Figure 9-2 is a block diagram of a Z280 MPU counter/timer.

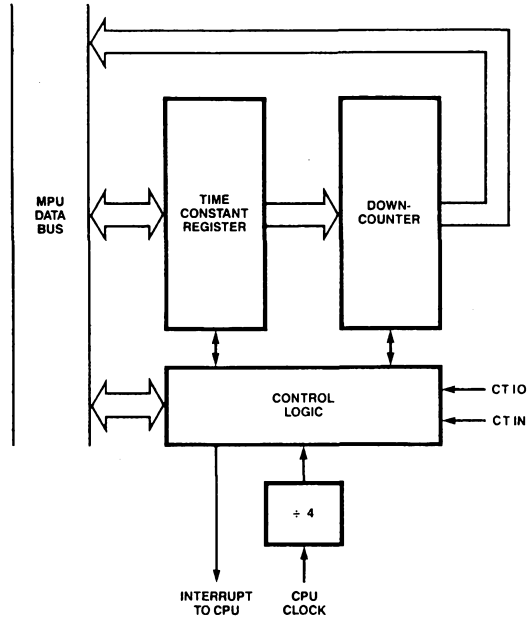


Figure 9-2. Counter/Timer Block Diagram

C/T 0 and C/T 1 can be programmably linked to form a 32-bit counter/timer.

Two external connections are available for each counter/timer: a Counter/Timer I/O pin (C/T I/O) that can act as a gate or trigger input or a counter/timer output, and a Counter/Timer Input pin (C/T IN) that can serve as a count, gate, trigger, or gate/trigger input. The contents of the Counter/Timer Configuration register determine the pin functions for a given application.

The counter/timers can operate in counter mode or in timer mode. In counter mode, the downcounter decrements the count on the occurrence of an external event; specifically, the counter is clocked by a rising edge on the Counter/Timer Input pin. In timer mode, the downcounter is clocked by an internal signal--the CPU clock divided by four.

Gate and trigger inputs to the downcounter can be used to control counter/timer activity. Both hardware and software gate and trigger signals are available. Either retriggerable or nonretriggerable modes can be specified.

The counter/timer's "terminal count" condition is when the downcounter holds a count of 0. This terminal count condition can be used to generate an interrupt request to the CPU. Counter/timers can generate a counter/timer output signal when the terminal count is reached. Upon reaching terminal count, a counter/timer can be programmed either to discontinue counting (single-cycle mode) or to reload the initial time constant value and continue counting (continuous mode).

9.4.1 Counter/Timer Operating Modes

The counter/timers have two basic operating modes, distinguished by the clocking signal to the downcounter: counter mode and timer mode. The current mode for counter/timer operation is determined by the contents of the Counter/Timer Configuration register.

In counter mode operation, the counter/timer monitors an external input line and records low-to-high transitions on that line. The Counter/Timer Input pin is used as the counter's input signal; if the appropriate enabling conditions are met, a low-to-high transition on that pin will cause the contents of the downcounter to be decremented by one. The decrement operation in the downcounter is actually performed on the first rising edge of the scaled processor clock (CPU clock divided by 4) after the low-to-high transition on the C/T IN signal. Typically, counter mode is used in event-counting types of applications.

In timer mode operation, the counter/timer monitors the internal CPU clock scaled by four for low-to-high transitions. If the appropriate enabling conditions are met, such a transition causes the contents of the downcounter to be decremented by one. No external inputs are required in the timer mode of operation. Timer mode is used in applications such as delay interval timing, watchdog timing, and clock generation.

In either mode, the maximum count frequency is the CPU clock divided by four.

9.4.2 Gates and Triggers

Gate and trigger inputs are used to control counter/timer activity in either counter mode or timer mode.

Gate signals are used in applications where counting or timing is to occur only during certain specified intervals; the counter/timer will count or time only while the gating condition is met. For applications where an external pin is configured as a gate input, counting or timing operations are performed only while the gate input is high. A software gate bit (one bit of the Counter/Timer Command/Status register) is used as a filter for the gate input; while the software gate bit is cleared to 0, the gating condition is not met regardless of the state of the gating line. In other words, the gating condition is a logical AND of the hardware and software gates; both the gate input must be high and the software gate bit must be set to 1 for the counter/ timer to be operating. If no external pins are configured as a gating signal, then the software gate bit must be set to 1 to satisfy the gating condition.

Figure 9-3 illustrates the gating facility in an application where the counter/timer is in counter mode with both the gate and the count signals coming from external pins. This example assumes that the software gate bit has been set to 1. The contents of the downcounter are decremented on a low-to-high transition of the count input only if the gate input is high.

If trigger mode is selected, a countdown sequence for a counter/timer begins only after a triggering condition occurs; a counting or timing operation can begin only after a low-to-high transition is detected on the trigger. If an external input is used as a trigger, that line is monitored by the counter/timer. Alternatively, a software trigger bit (one bit in the Counter/Timer Command/Status register) can be set to 1 from a previously cleared value to activate the counter/timer. The

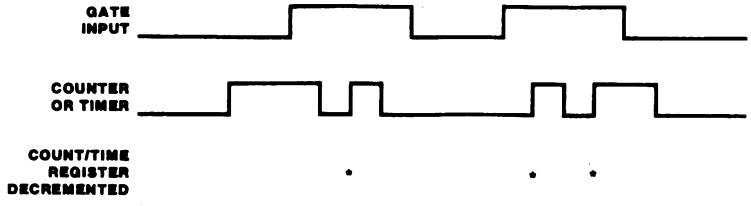


Figure 9-3. Counter Operation with Gate Only

trigger condition is a logical OR of the hardware and software triggers; that is, either a hardware or software trigger will activate an enabled counter/timer.

Figure 9-4 illustrates trigger operation in an application where the counter/timer is in the counter mode with both the trigger and count inputs provided by external pins. This example assumes that the software trigger bit does not make a low to high transition. The contents of the downcounter are decremented on a low-to-high transition of the count input only after a low-to-high transition on the trigger input has been detected.

Either a retriggerable or nonretriggerable operation can be specified. In the retriggerable mode, the occurrence of a trigger condition causes the counter/timer to reload its initial time constant value regardless of the current contents of the downcounter. This mode is used in applications such as watchdog timers. In the nonretriggerable mode, after the first trigger condition starts counter/timer activity, subsequent trigger conditions are ignored. Nonretriggerable mode is used in applications such as delay counters that measure a fixed delay from a given event.

Gate and trigger operations can be combined in a single counter/timer. Separate gate and trigger inputs (either hardware or software) can be specified, or one external input can be used as both a gate and a trigger. In the latter case, a low-to-high transition on the input acts as a trigger that starts counter/timer activity, and then counting or timing continues only as long as the input signal remains high. Again, either retriggerable or nonretriggerable modes are available. Figure 9-5 illustrates counter/timer operation in an application where counter mode is selected, one input is a count input, and the other input is used as both the trigger and gate.

9.4.3 Terminal Count Condition

During operation, the counter/timer counts down from a preset time constant value. The time constant value can range from 0 to 65535. The terminal count condition is reached with the transition from a count of 1 in the downcounter to a count of 0. The counter/timers can be programmed to interrupt the CPU and/or generate a counter/timer output signal when the terminal count is reached.

Another set of operating modes determines counter/timer activity upon reaching the terminal count. Whether in counter or timer mode, a counter/timer can be configured for single-cycle mode or continuous mode. In single-cycle mode, the counter/timer halts operation upon reaching terminal count; a new trigger is required to reload the time constant and initiate another countdown sequence. In continuous mode, the counter/timer is automatically reloaded with the time constant upon reaching terminal count; the downcounter is reloaded on the next count input after reaching terminal count. For example, a counter/timer in continuous mode with a 3 in its Time Constant register will be reloaded on every fourth count input.

An interrupt enable bit in the Counter/Timer Configuration register determines if an interrupt request is generated at the terminal count. This request will be processed by the CPU if the appropriate Interrupt Request Enable bit in the CPU's Master Status register is set to 1 (see Chapter 6).

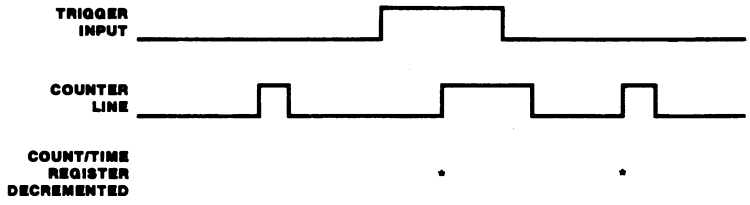


Figure 9-4. Counter Operation with Trigger Only

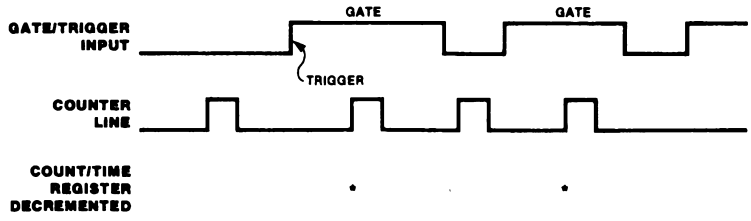


Figure 9-5. Counter Operation with Gate and Trigger

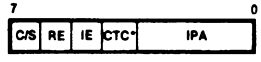
The CTIO pin can be configured as a counter/timer output signal. Reaching the terminal count condition causes a low-to-high transition on the CTIO pin; this signal remains high as long as the downcounter holds a value of zero (that is, until a non-zero time constant is loaded into the downcounter due to a trigger condition).

9.4.4 Counter/Timer Registers

Each counter/timer has two 8-bit command and status registers and two 16-bit count registers. The 8-bit Counter/Timer Configuration and Counter/Timer Command/Status registers determine the counter/timer's operating modes and provide status information about the current operation. If C/T 0 and C/T 1 are linked to form a 32-bit counter/timer, the functionality of these registers is affected, as described in section 9.4.5. The 16-bit Time Constant register holds the initialization value for the counter/timer, and the 16-bit Count-Time register contains the value of the current count in progress.

9.4.4.1 Counter/Timer Configuration Register

The Counter/Timer Configuration register, shown in Figure 9-6, specifies the counter/timer's mode of operation. The five fields in this register are described below.



*CTC is present on counter/timer 0 only.

Figure 9-6. Counter/Timer Configuration Register

Continuous/Single Cycle (C/S). While this bit is set to 1, the downcounter is automatically reloaded with the contents of the Time Constant register on the next count input signal after terminal count is reached, and the counting or timing operation continues. While this bit is cleared to 0, no automatic reloading occurs when terminal count is reached.

Retrigger Enable (RE). While this bit is set to 1, the value of the Time Constant register is loaded into the downcounter whenever a trigger input is received (retriggerable mode). While this bit is 0, trigger conditions do not cause reloading of the downcounter.

Interrupt Enable (IE). While this bit is set to 1, the counter/timer generates an interrupt request to the Z280 CPU upon reaching terminal count. While this bit is cleared to 0, no interrupt requests can be generated by the counter/timer.

Counter/Timer Cascade (CTC). For C/T 0, this is the enable bit for linking to C/T 1 in order to form a 32-bit counter/timer (see section 9.4.5). The state of this bit has no effect in C/T 1 and C/T 2.

Input Pin Assignments (IPA). The contents of this 4-bit field determine the operating mode of the counter/timer (counter or timer mode) and the functionality of the external pins associated with that counter/timer. The four bits in this field are associated with enabling the generation of an output pulse (EO), selecting the counter or timer mode (C/T), enabling the gating facility (G), and enabling the triggering facility (T). Table 9-1 shows the encoding of this field.

Table 9-1. Encoding of the IPA Field in the Counter/Timer Configuration Register

IPA Field				Pin Functionality		
EO	C/T	G	T	Counter/Timer I/O	Counter/Timer Input	Mode
0	0	0	0	Unused	Unused	Timer
0	0	0	1	Unused	Trigger	Timer
0	0	1	0	Gate	Unused	Timer
0	0	1	1	Gate	Trigger	Timer
0	1	0	0	Unused	Input	Counter
0	1	0	1	Trigger	Input	Counter
0	1	1	0	Gate	Input	Counter
0	1	1	1	Gate/Trigger	Input	Counter
1	0	0	0	Output	Unused	Timer
1	0	0	1	Output	Trigger	Timer
1	0	1	0	Output	Gate	Timer
1	0	1	1	Output	Gate/Trigger	Timer
1	1	0	0	Output	Input	Counter
1	1	0	1	Unused	Unused	Reserved
1	1	1	0	Unused	Unused	Reserved
1	1	1	1	Unused	Unused	Reserved

If a reserved encoding of the IPA field is specified for any counter/timer, counter/timer operation is unpredictable.

The Counter/Timer Configuration registers are cleared to all zeros by a reset.

9.4.4.2 Counter/Timer Command/Status Register

The Counter/Timer Command/Status register provides for software control of counter/timer operation and reflects the current status of the counter/timer. Three control bits and three status bits are included in the Command/Status register. The format for this register is illustrated in Figure 9-7.

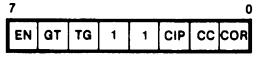


Figure 9-7. Counter/Timer Command/Status Register

Enable (EN). While this bit is set to 1, the counter/timer is enabled; operation begins on the first rising edge of the processor clock following the setting of this bit from a previously cleared state. Writing a 1 to this bit when its previous value was a 1 has no effect. While this bit is cleared to 0, the counter/timer is disabled and performs no counting or timing operations. While in the disabled state, the contents of the Time Constant register are continuously loaded into the downcounter.

Software Gate (GT). While the counter/timer is enabled (the EN bit is a 1), downcounter operation begins on the rising edge of the first scaled processor clock following the setting of this bit from a previously cleared value. Writing a 1 to this bit when the previous value was a 1 has no effect. While this bit is cleared to 0, the counting or timing sequence is halted.

Software Trigger (TR). While the counter/timer is enabled (the EN bit is a 1), the trigger condition is generated on the rising edge of the first scaled processor clock following the setting of this bit from a previously cleared value. If a previous trigger condition has not occurred, the contents of the Time Constant register are loaded into the downcounter and the counting or timing sequence begins. If a hardware or software trigger has already occurred and the Retrigger Enable bit is set to 1, the counter/timer will be retriggered. If a trigger has already occurred, the Retrigger Enable bit is cleared to 0, and a counting or timing operation is in progress (that is, the downcounter holds a count other than 0), then setting the TR bit has no effect on counter/timer operation. Clearing this bit to 0 also has no effect on counter/timer operation.

Count in Progress (CIP). This status bit indicates if a counting or timing operation is in progress. While this bit is a 1, the counter/timer has a time constant loaded and the downcounter holds a non-zero value. While this bit is a 0, the counter/timer is not operating. The state of this bit is determined by control logic in the counter/timer and cannot be altered by a write operation to this register.

End-of-Count Condition Has Been Reached (CC). This status bit is set to 1 by control logic in the counter/timer when the end-of-count condition is reached (that is, the downcounter has been decremented to zero in the single-cycle mode or the downcounter has been reloaded in the continuous mode). While this bit is a 0, the downcounter has not been decremented to 0 since the last time that this bit was cleared by software. This bit can be read or written under program control.

Count Overrun (COR). This status bit is set to 1 by control logic in the counter/timer if the end-of-count condition is reached while the CC bit is already set to 1, thereby indicating a count over-run condition. If this bit is a 0, the end-of-count condition has not been reached while the CC bit is a 1 since the last time the CC bit was cleared by software. This bit can be read or written under program control.

The Counter/Timer Command/Status register is cleared to all zeros by a reset. Bits 3 and 4 of this register are not used, and should always be written with zeros (however, when bits 3 and 4 are read back, they will be 1s regardless of whether they were written with zeros or ones).

9.4.4.3 Time Constant and Count-Time Registers

The 16-bit Time Constant register holds the value to be loaded into the downcounter when counter/timer operation begins. The downcounter is loaded with the contents of the Time Constant register when the counter/timer is initially triggered to begin counter/timer operation, each time the end-of-count condition is reached if the continuous mode is selected, and at the occurrence of each trigger condition if retriggerable mode is selected. By loading the Time Constant register, the user can specify counts ranging from 1 to 65536. The contents of the Time Constant register are continuously loaded into the downcounter while the counter/timer is disabled (the EN bit is 0).

The 16-bit Count-Time register holds the current value in the downcounter and can be read at any time without affecting counter/timer operation. Writes to this register have no effect.

Both the Time Constant and Count-Time registers hold unpredictable values after a reset.

Table 9-2 lists the I/O port addresses associated with each of the counter/timers' registers. The Counter/Timer Configuration register and Counter/Timer Command/Status register are accessed with byte I/O instructions and, with the exception of the read-only CIP bit, can be read or written. The Time Constant and Count-Time registers are accessed with word I/O instructions. The Time Constant register can be read or written; the Count-Time register is read-only.

Table 9-2. I/O Addresses of Counter/Timer Registers

Register	Counter/Timer C/T 0	C/T 1	C/T 2
Configuration	FExxE0	FExxE8	FExxF8
Command/Status	FExxE1	FExxE9	FExxF9
Time Constant	FExxE2	FExxEA	FExxFA
Count-Time	FExxE3	FExxEB	FExxFB

All addresses are in hexadecimal.
"x" means "don't care".

9.4.5 Linking Counter/Timers

Under software control, two Z280 MPU counter/timers can be linked to form a 32-bit counter/timer. C/T 0 can be linked with C/T 1. This linking function is controlled by the CTC bit in the Counter/Timer Configuration register in C/T 0. While the CTC bit in C/T 0's Configuration register is set to 1, C/T 0 and C/T 1 are linked together.

Linking the two counter/timers together affects the functionality of the counter/timers' registers. If C/T 0 and C/T 1 are linked to form a 32-bit counter, C/T 1's Time Constant register holds the upper 16 bits and C/T 0's Time Constant register holds the lower 16 bits of the 32-bit count to be loaded into the downcounter when a counter/timer operation begins. Similarly, C/T 1's Count-Time register holds the upper 16 bits and C/T 0's Count-Time register holds the lower 16 bits of the current count.

The effect of linking counter/timers on the Configuration and Command/Status registers is summarized in Table 9-3. The configuration of the 32-bit counter/timer is determined by the state of the C/S, RE, and IPA fields in the Configuration register of the more significant counter/timer (C/T 1). Any external connections specified in the IPA field of the C/T 1 Configuration register use the pins associated with C/T 1. The controls in the Configuration register for C/T 0 are ignored, with the exception of the CTC, IE, and EO bits. The CTC bit in C/T 0 is used to specify linking of the counter/timers. If the IE bit in the more significant counter/timer (C/T 1) is set to 1, an interrupt request is generated when the 32-bit counter reaches end-of-count, using the interrupt request signal from C/T 1; if the IE bit in the less significant counter/timer (C/T 0) is set to 1, an interrupt request is generated when the lower 16 bits of the 32-bit downcounter reach 0 (in other words, when C/T 0 reaches end-of-count), using the interrupt request signal from C/T 0. If the OE bit in C/T 0 is set, the C/T I/O signal associated with C/T 0 goes high whenever the lower half of the 32-bit down-counter holds a 0 (in other words, when C/T 0's downcounter holds a 0).

Similarly, the Command/Status register in the more significant counter/timer (C/T 1) contains the control and status bits for the linked 32-bit counter/timer. However, the status bits in the less significant counter/timer (C/T 0) hold valid status for the lower-half of the 32-bit counter/timer (that is, the status of C/T 0 itself).

9.4.6 Counter/Timer Sequence of Events

Before starting a counting or timing sequence, the counter/timer must be configured for the particular application by loading its Configuration register. Next, the starting value for the downcounter is specified by loading the Time Constant register; initial values ranging from 0 to 65535 can be specified for the downcounter. Lastly, the enable (EN) bit in the Command/Status register is set to 1 to enable counter/timer operation.

While the EN bit is cleared to 0, the counter/timer cannot be triggered, interrupt requests from the counter/timer cannot be generated, and the downcounter holds the value in the Time Constant register. However, clearing the EN bit does not clear any pending interrupt requests--it only prevents new interrupt requests from being generated.

Once the EN bit is set to 1, the countdown sequence begins when the counter/timer is triggered, causing the contents of the Time Constant register to be loaded into the down counter. The downcounter is loaded on the rising edge of the external trigger input (if an external trigger was specified in the Configuration register) or by writing a 1 into the TG bit of the Command/Status register. The EN and TG bits can both be set to 1 during the same write operation to the Command/Status register to both enable and trigger a counter/timer (assuming that the TG bit was a zero previously, so that a low-to-high

Table 9-3. Configuration and Command/Status Registers for Linked Counter/Timers

Bit	Active/Ignored	Comments
C/T 1 Configuration Register		
C/$\overline{S}$	Active	Specifies continuous or single-cycle mode for 32-bit counter/timer.
RE	Active	Specifies retriggerable or nonretriggerable mode for 32-bit counter/timer.
IE	Active	Interrupt enable for 32-bit counter/timer.
CTC	Ignored	
EO	Active	Enable output for 32-bit counter/timer; C/T 1's output pin is used.
C/T	Active	Specifies counter or timer mode for 32-bit counter/timer.
G	Active	Enable gate input for 32-bit counter/timer; C/T 1's input pin is used.
T	Active	Enable trigger input for 32-bit counter/timer; C/T 1's input pin is used.
C/T 0 Configuration Register		
C/$\overline{S}$	Ignored	
RE	Ignored	
IE	Active	Interrupt enable for lower half of 32-bit counter/timer.
CTC	Active	Set to 1 to link counter/timers.
EO	Active	Enable output for lower half of 32-bit counter/timer (C/T 0 only).
C/T	Ignored	
G	Ignored	
T	Ignored	
C/T 1 Command/Status Register		
EN	Active	Enable control for 32-bit counter/timer.
GT	Active	Software gate for 32-bit counter/timer.
TG	Active	Software trigger for 32-bit counter/timer.
CIP	Active	Count-in-Progress status bit for 32-bit counter/timer.
CC	Active	End-of-Count Has Been Reached status bit for 32-bit counter/timer.
COR	Active	Count Overrun status bit for 32-bit counter/timer.
C/T 0 Command/Status Register		
EN	Ignored	
GT	Ignored	
TG	Ignored	
CIP	Active	Count-in-Progress status bit for lower half of 32-bit counter/timer.
CC	Active	End-of-Count Has Been Reached status bit for lower half of 32-bit counter/timer.
COR	Active	Count Overrun status bit for lower half of 32-bit counter/timer.

transition on the trigger is detected). The trigger condition is a logical OR of the external trigger input (if specified) and the TG bit.

Once triggered, the rate at which the downcounter counts is determined by the mode of the counter/timer. In the timer mode, the downcounter is clocked internally by a signal that is one-fourth the frequency of the CPU clock (one-eighth the frequency of the external clock source). In the counter mode, the downcounter is clocked by a rising edge on the count input signal (this edge is internally synchronized with the scaled CPU clock).

In counter mode, the first low-to-high transition on the count input should occur a minimum of four internal CPU clock cycles after the trigger event. Count inputs occurring within four CPU clock cycles of the trigger may or may not be recognized by the downcounter.

Once the downcounter is loaded, the countdown sequence continues towards the terminal count condition as long as the counter/timer's gate input is high. The gate input to the counter/timer is the logical AND of the external gate input (if an external gate was specified in the Configuration register) and the GT bit in the

Command/Status register. If the gate input goes low, the countdown halts, and then resumes when the gate input goes high again. The gate function does not affect the trigger function.

The reaction to triggers during the countdown operation depends on the state of the RE bit in the Configuration register. If RE is a 0, retriggers are ignored and the countdown sequence continues normally. If RE is a 1, each occurrence of a trigger condition causes the downcounter to be reloaded from the Time Constant register and the countdown sequence starts over again.

The current state of the downcounter can be determined by polling the status bits in the Command/Status register and by reading the current count from the Count-Time register. Reading these registers does not affect the current countdown sequence.

The state of the C/$\overline{S}$ bit in the Configuration register controls the operation of the counter/timer upon reaching terminal count. If the C/$\overline{S}$ bit is a 1, specifying the continuous mode of operation, the downcounter is reloaded from the Time Constant register on the next count input after reaching terminal count, and a new countdown sequence begins. The Time Constant register can be programmably altered during counter/timer operation without affecting the current countdown sequence. If the C/$\overline{S}$ bit is 0, specifying single-cycle operation, the downcounter halts upon reaching terminal count until the next occurrence of a trigger condition reloads the downcounter.

If the IE bit in the Configuration register is a 1, an interrupt request is generated upon reaching the terminal count. If a counter/timer output signal is specified in the IPA field of the Configuration register, reaching terminal count causes a low-to-high transition on the output signal; this signal then remains high until the downcounter is reloaded with a non-zero value due to a trigger condition or disabling of the counter/timer with a non-zero value in the Time Constant register. Note that the counter/timer output line can be forced high by disabling the counter/timer with all zeros loaded into the Time Constant register.

9.5 DMA CHANNELS

The Z280 MPU has four on-chip Direct Memory Access (DMA) transfer controllers for high-bandwidth data transmissions within a Z280-based system. Each DMA channel is capable of controlling high speed memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-to-peripheral data transfers.

All four DMA channels, referred to as DMA0, DMA1, DMA2, and DMA3, are capable of controlling "flowthrough" type data transfers, wherein data is temporarily stored in the DMA device between reading from the source and writing to the destination. Two of the channels, DMA0 and DMA1, also support "flyby" mode data transfers, wherein the data is read from the source and written to the destination during a single bus transaction. Otherwise, the four DMA controllers are identical, although they have different priorities with respect to interrupt and bus requests.

Two external signals provide the interface between the DMA channels and external memory or peripheral devices. The READY ($\overline{RDY}$) input is used by an external device to request activity by a DMA channel. The DMA STROBE ($\overline{DMASTB}$) output is used to signal the I/O port when a flyby transaction is in progress; $\overline{DMASTB}$ is available only for DMA0 and DMA1.

Two 24-bit addresses are generated by the DMA for each flowthrough transaction, and one 24-bit address for each flyby transaction. These addresses can be physical memory addresses or I/O port addresses. The addresses are automatically generated for each transaction, and can be fixed, incrementing, or decrementing. Two readable registers, the Source Address register and Destination Address register, hold the current address of the source and destination ports.

During a DMA-controlled transaction, the DMA channel assumes control of the system's address and data buses. The on-chip DMA channels behave as if they were external bus requestors with respect to acquiring, using, and relinquishing the bus. The DMA channels are arranged in a priority daisy chain with the external Bus Request input signal being the "next lowest bus requestor" on the chain. Data can be transferred as bytes or words, using the same memory and I/O timing as the CPU for bus transactions (as determined by the contents of the Bus Timing and Initialization register).

Two DMA devices can be programmably linked, where one DMA channel is used to program the second DMA channel. DMA3 can be linked to DMA1 and DMA2 can be linked to DMA0 in this manner. DMA0 can also be programmably linked to the on-chip UART's receiver, and DMA1 can be linked to the on-chip UART's transmitter.

The DMA Master Control register specifies the general configuration of all four DMA channels, including the linking of DMA channels to the UART. Each DMA channel has its own Transaction Descriptor register that determines the operating

modes for that channel, Source Address and Destination Address registers that hold the addresses for the DMA transfers, and a Count register that controls the number of transfers to be performed. All DMA registers are accessed via I/O instructions.

9.5.1 Types of DMA Operations

The Z280 MPU's on-chip DMA channels are capable of two basic types of operations: flowthrough mode data transactions and flyby mode data transactions.

All four on-chip DMA channels support flowthrough mode data transactions. In flowthrough mode, each DMA-controlled data transfer involves two bus operations: a read cycle to obtain the data from the source and a write cycle to transfer the data to the destination. The data is temporarily stored in the DMA device between the read and write operations. Flowthrough mode transactions use the same address, data, and control signals as CPU-initiated transactions and, therefore, require no additional external logic in a Z280-based system. Memory-to-memory, memory-to-peripheral, peripheral-to-memory, or peripheral-to-peripheral transfers are possible using flowthrough mode.

Flyby mode data transactions are supported only by DMA0 and DMA1. In a flyby mode transaction, the data is read from the source and written to the destination in a single bus operation. There are two types of flyby transactions: memory-to-peripheral and peripheral-to-memory. For a memory-to-peripheral transaction, the DMA channel generates a memory read bus cycle and notifies the I/O device that a flyby transaction is in progress by activating the $\overline{\text{DMASTB}}$ output. The data must be written to the I/O device during the memory read operation. For a peripheral-to-memory flyby transaction, the DMA channel generates a memory write bus cycle while activating the $\overline{\text{DMASTB}}$ output; the data must be read from the I/O device during the memory write transaction. In other words, during flyby mode transactions, the DMA channel generates the bus signals needed to control the memory access, and $\overline{\text{DMASTB}}$ is used to notify the peripheral device when to read data from the bus (for memory-to-peripheral transfers) or when to put data onto the bus (for peripheral-to-memory transfers.) Thus, flyby mode transactions require additional external logic to activate the appropriate peripheral device when $\overline{\text{DMASTB}}$ is active. However, flyby mode transactions are faster than flowthrough mode transactions, since only one bus cycle is needed to complete a data transfer.

9.5.2 DMA Transfer Modes

When transferring data under DMA control (with either flowthrough or flyby transactions), one of three transfer modes can be selected: single transaction, burst, or continuous mode. Once DMA activity has been initiated, the transfer mode determines how many DMA-controlled data transfers are to occur before the DMA channel relinquishes the bus to the CPU or another DMA channel.

In the single transaction mode, the DMA controller transfers only one byte or word of data at a time. Control of the system bus is returned to the CPU between each DMA transfer; the DMA must make a new request for the bus before performing the next data transfer.

In the burst mode, once the DMA channel gains control of the bus, it continues to transfer data until the $\overline{\text{RDY}}$ input goes inactive. When the $\overline{\text{RDY}}$ line becomes inactive, the DMA releases the system bus; bus control then returns back to the CPU or to the next lower-priority DMA channel with a bus request pending.

In the continuous mode, the DMA channel retains control of the system bus until the entire block of data has been transferred. If the $\overline{\text{RDY}}$ line goes inactive before the entire data block is transferred, the DMA simply waits until $\overline{\text{RDY}}$ becomes active again, without releasing the bus. This mode is the fastest mode since it has the least response-time overhead when the $\overline{\text{RDY}}$ line momentarily goes inactive and returns active again. However, this mode does not allow any CPU activity for the duration of the transfer. Figure 9-8 summarizes the DMA transfer modes.

In any transfer mode, once a DMA-controlled data transfer begins, that transaction is completed in an orderly fashion, regardless of the state of the $\overline{\text{RDY}}$ input.

DMA0 and DMA1 include a software $\overline{\text{RDY}}$ signal in the DMA Master Control register. The $\overline{\text{RDY}}$ input to these DMA channels is the logical OR of the $\overline{\text{RDY}}$ pin and the software-controlled $\overline{\text{RDY}}$ signal.

A DMA channel can be programmed to perform data transfers on a byte (8-bit), word (16-bit), or long word (32-bit) basis. If a DMA's port address is a memory address that is auto-incremented or auto-decremented after each transfer, the size of the data transfer determines whether the memory address is incremented or decremented by a factor of 1, 2, or 4. For word and long word transfers to or from memory locations, the memory address must be even-valued (that is, the least significant bit of the memory address must be 0).

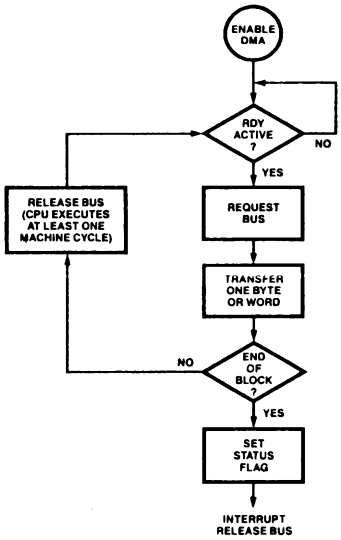


Figure 9-8a. Single Transaction Mode

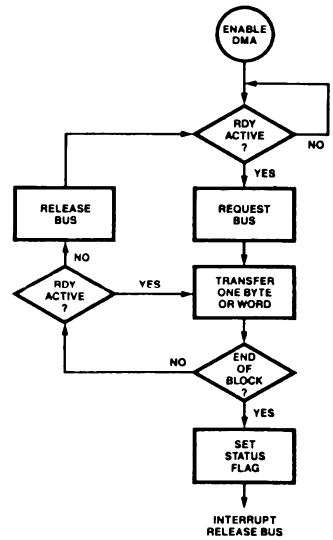


Figure 9-8b. Burst Mode

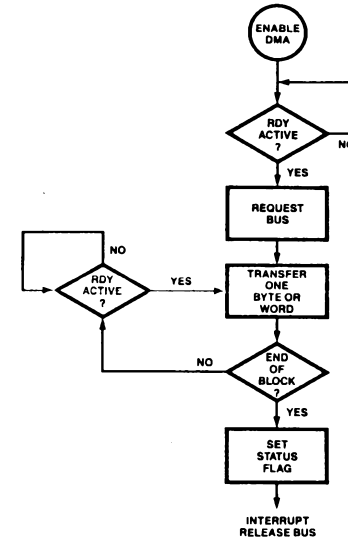


Figure 9-8c. Continuous Mode

Figure 9-8. Modes of Operation

Transfers of unaligned data on 16-bit buses can be accomplished via byte transfers only. Long word transfers are used in applications where the Z280 MPU is acting as a DMA controller for a system with a 32-bit bus, such as a Z80,000-based system. During long word transactions, the Z280 MPU's DMA channel provides only 24 bits of the address; the upper 8 bits of the 32-bit address have to be generated with external logic. Long word transfers are supported only in the flyby mode with the on-chip cache programmably disabled.

9.5.3 End-of-Process

An enable bit in the DMA Master Control register allows the Interrupt A input to be used as an end-of-process ($\overline{EOP}$) input during DMA transactions. When enabled, transfers by DMA channels can be prematurely terminated by a low on the $\overline{EOP}$ (Interrupt A) line. Recognition of the $\overline{EOP}$ signal is not affected by the state of the Interrupt Request Enable bit for Interrupt A in the CPU's Master Status register.

If the $\overline{EOP}$ signal goes active during the read portion of a flowthrough transaction, the DMA activity is aborted before the write portion of that transaction. If $\overline{EOP}$ becomes active during the write portion of a flowthrough transaction or during a flyby transaction, that transfer is completed before stopping the DMA operation.

When an active $\overline{EOP}$ signal terminates a DMA operation, the $\overline{EOP}$ Signaled (EPS) status bit in that channel's Transaction Descriptor register is automatically set to 1 and the Enable bit in that same register is cleared to 0. If that channel's Interrupt Enable bit is set to 1, an interrupt request to the CPU is generated.

The $\overline{EOP}$ signal is level-sensitive and shared by all four on-chip DMA channels. Thus, if an active $\overline{EOP}$ signal terminates the activity of one DMA channel and another DMA channel immediately requests the bus, the second DMA's activity is terminated before any transactions can be generated if $\overline{EOP}$ is still active. In other words, the second DMA channel also recognizes the $\overline{EOP}$ signal, and so on. Therefore, in order for the currently active DMA channel to be the only channel whose activity is terminated, $\overline{EOP}$ should be asserted for only one bus clock cycle in systems where the bus clock frequency is equal to or one-half of the processor clock frequency; $\overline{EOP}$ should be asserted for one-half of a bus clock cycle in systems where the bus clock frequency is one-fourth of the processor clock frequency.

If the end-of-process capability is enabled, a single input to the Z280 MPU can act as both the Interrupt A and the $\overline{EOP}$ signal; it acts as the Interrupt A Request line when the CPU controls the bus and as the $\overline{EOP}$ line when a DMA channel controls the bus. If an $\overline{EOP}$ signal terminates a

DMA operation, and that signal is still asserted when the CPU regains control of the bus, then the signal is interpreted as an interrupt request. Thus, a single signal can be used to stop DMA activity and generate an interrupt, if so desired. Note that the interrupt request generated by the DMA channel and the interrupt request generated by an active signal on the Interrupt A line are different interrupt requests, each with its own priority and its own enabling bit in the CPU's Master Status register.

9.5.4 Priority Resolution

Prioritization of the four on-chip DMA channels is implemented via an internal "service request" latch. A DMA channel generates a service request, indicating that the channel needs to gain control of the bus, if that channel's Enable bit in the Transaction Descriptor register is set to 1 and an active $\overline{RDY}$ signal is asserted. This service request signal is latched in the service request latch only if all preceding service requests have already been serviced (that is, there are no service requests active in the latch). Once a service request is latched, the service request latch is "closed" to all other service requests until the current requests are serviced; the latched requests are serviced in priority order, where DMA channel 0 has highest priority and DMA channel 3 has lowest priority. When all latched service requests have been serviced, the latch is "opened" so that new service requests can be latched.

This service request mechanism provides for non-preemptive prioritization of DMA activity. For example, if DMA channel 1 requires servicing while the other channels are quiescent (that is, not currently controlling the bus or making a service request), channel 1's service request is latched and the service request latch is closed. Thus, no other channel can preempt channel 1's activity. If channels 0 and 2 activate service requests while channel 1 is being serviced, both those requests will be latched after channel 1's activity is completed, and channel 0 will be serviced next, followed by channel 2. No new service requests are latched until both channels 0 and 2 have been serviced, and so on.

All service requests from the on-chip DMA channels have priority over bus requests made via the $\overline{BUSREQ}$ input by external DMA controllers.

9.5.5 DMA Linking

The Z280 MPU's on-chip DMA devices can be linked together to provide for DMA transfers to non-contiguous memory locations. In a linked configuration, one channel, called the master DMA, controls the actual data transfers to the memory and/or peripheral devices; the second channel, called the linked DMA, is used to load the master DMA's control registers from memory when the master DMA completes an operation. The master DMA signals the linked DMA when a transaction is completed via an internal "ready" input to the linked DMA. The linked DMA then initiates the transfers that load the master DMA's control registers from memory, allowing the master DMA to perform multiple data transfer operations without any CPU intervention.

Control bits in the DMA Master Control register allow DMA3 to be linked to DMA1, with DMA1 the master DMA and DMA3 the linked DMA, and DMA2 to be linked to DMA0, with DMA0 the master DMA and DMA2 the linked DMA.

When the linked DMA loads the master DMA's registers, the registers are written in the following order:

- Destination Address register (least significant word)
- Destination Address register (most significant word)
- Source Address register (least significant word)
- Source Address register (most significant word)
- Count register
- Transaction Descriptor register

After the six words have been written to the master DMA, the master DMA deasserts the ready signal to the linked DMA and begins the new transfer operation. For Z-BUS configurations of the Z280 MPU, the linked DMA uses six word transactions on the bus to program the master DMA; for Z80 Bus configurations, the linked DMA uses twelve byte transactions to program the master DMA, with the least significant byte of each word being transferred first.

Control bits in the DMA Master Control register also allow DMA0 to be programmably linked to the on-chip UART's receiver and DMA1 to be linked to the UART's transmitter. If so linked, an internal "ready" signal to DMA0 is automatically generated when the UART's receive buffer is full. Similarly, an internal "ready" signal to DMA1 is automatically generated when the UART's transmit buffer is empty. The external $\overline{RDY}$ inputs are ignored while in this configuration.

9.5.6 DMA Registers

DMA registers consist of a DMA Master Control register that specifies the general configuration of all four channels, and a Transaction Descriptor register, Source Address register, Destination Address register, and Count register for each DMA channel. All DMA registers are accessed using word I/O instructions.

9.5.6.1 DMA Master Control Register

The 16-bit DMA Master Control register is illustrated in Figure 9-9. The bit fields within this register are described below.

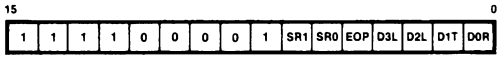


Figure 9-9. DMA Master Control Register

DMA0 to Receiver Link (D0R). While this bit is set to 1, DMA0 is linked to the on-chip UART's receiver.

DMA1 to Transmitter Link (D1T). While this bit is set to 1, DMA1 is linked to the on-chip UART's transmitter.

DMA2 Link (D2L). While this bit is set to 1, DMA2 is linked to DMA0.

DMA3 Link (D3L). While this bit is set to 1, DMA3 is linked to DMA1.

End-of-Process (EOP). While this bit is set to 1, the Interrupt A input acts as an End-of-Process input for the active DMA channel during DMA operations.

Software Ready for DMA0 (SR0). While this bit is set to 1, DMA0 requests use of the system bus if enabled.

Software Ready for DMA1 (SR1). While this bit is set to 1, DMA1 requests use of the system bus if enabled.

The DMA Master Control register is cleared to all zeros by a reset, unless bootstrap mode is enabled during the reset operation (see sections 3.2.1 and 9.7). Bits 7 through 15 of this register are not used.

9.5.6.2 DMA Transaction Descriptor Register

Each DMA channel has its own 16-bit Transaction Descriptor register. The Transaction Descriptor register (Figure 9-10) describes the type of DMA transfer to be performed and contains control and status information.

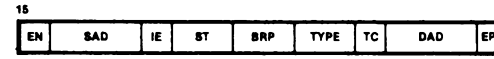


Figure 9-10. Transaction Descriptor Register

End-of-Process Signaled (EPS). This status bit is set to 1 automatically when an active End-of-Process signal prematurely terminates a DMA transfer. This bit can be set to 1 or cleared to 0 under software control.

Destination Address Descriptor (DAD). This 3-bit control field determines the type of location (memory or I/O) to be accessed as the destination port during DMA transfers, and whether the destination address is to be incremented, decremented, or left unchanged between transfers, as shown in Table 9-4. When memory addresses are auto-incremented or auto-decremented, the incrementing or decrementing value is determined by the size of the data transfer, as specified in the ST field. I/O port addresses are always auto-incremented and auto-decremented by 1.

Table 9-4. Encoding of DAD and SAD Fields in DMA Transaction Descriptor Register

Encoding	Address Modification Operation
000	Auto-increment memory location
001	Auto-decrement memory location
010	Memory address unmodified by transaction
011	Reserved
100	Auto-increment I/O location
101	Auto-decrement I/O location
110	I/O address unmodified by transaction
111	Reserved

Transfer Complete (TC). This status bit is set to 1 automatically when the Count register has reached zero. This bit can be set to 1 or cleared to 0 under software control.

Transaction Type (Type). This 2-bit control field specifies the type of DMA operation to be performed, as shown in Table 9-5.

Table 9-5. Encoding of Type Field in Transaction Descriptor Register

Encoding	DMA Operation
00	Flowthrough
01	Reserved
10	Flyby write (peripheral-to-memory)
11	Flyby read (memory-to-peripheral)

Bus Request Protocol (BRP). This 2-bit control field determines the transfer mode for the DMA operation, as shown in Table 9-6.

Table 9-6. Encoding of BRP Field in Transaction Descriptor Register

Encoding	DMA Transfer Mode
00	Single transaction
01	Burst
10	Continuous
11	Reserved

Size of Transfer (ST). This 2-bit control field specifies the size of the entity to be transferred during each DMA-controlled transaction, as shown in Table 9-7. If auto-increment or auto-decrement of a source or destination memory address is specified in the SAD or DAD fields, then the state of this field determines the size of the increment or decrement operation.

Table 9-7. Encoding of ST Field in Transaction Descriptor Register

Encoding	Size of Transfer	Number to Increment or Decrement By
00	Byte	1
01	Word	2
10	Long word	4
11	Reserved	

Interrupt Enable (IE). While this bit is set to 1, the DMA channel generates an interrupt request to the CPU either when the Count register goes to zero, indicating the completion of a DMA operation, or when an End-of-Process signal prematurely terminates a DMA operation. While this bit is cleared to 0, no interrupt request is generated.

Source Address Descriptor (SAD). This 3-bit control field determines the type of location (memory or I/O) to be accessed as the source port during DMA transfers, and whether the source address is to be incremented, decremented, or left unchanged between transfers, as shown in Table 9-4.

DMA Enable (EN). While this bit is set to 1, the DMA channel is enabled; while enabled, the DMA can request control of the system bus and, upon becoming bus master, initiate transactions on the bus. While this bit is a 0, the DMA channel is disabled and cannot request control of the bus. The DMA registers can be accessed regardless of the state of this bit.

For DMA0, a reset loads a 0100_H into the Transaction Descriptor register. For the remaining three channels, the EN, IE, TC, and EPS bits are all cleared to 0 by a reset, and the remaining fields are unaffected.

9.5.6.3 Count Register

Each channel has a 16-bit Count register that is programmed to contain the number of DMA transfers to be performed. When the contents of the Count register reach zero (terminal count), further requests on the $\overline{RDY}$ line are ignored, and, if the IE bit in the Transaction Descriptor register is set to 1, an interrupt request is generated.

A reset loads a 0100_H into DMA0's Count register; the other channels' Count registers are unaffected by a reset.

9.5.6.4 Source Address and Destination Address Registers

The 24-bit Source Address register and Destination Address register hold the port addresses used during DMA transfers. These are physical addresses that are not translated by the MMU. In flyby mode, only one of these registers is used to supply the address for the transaction, as determined by the Type field in the Transaction Descriptor register. The contents of these registers can be automatically incremented or decremented by each DMA transaction, as determined by the SAD and DAD field in the Transaction Descriptor register.

The entire 24-bit Source Address or Destination Address register is read and written via two word

accesses to the register. Twelve bits of the address are accessed by each word I/O operation; the format used when accessing these registers is shown in Figure 9-11.

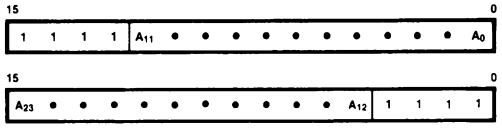


Figure 9-11. Source and Destination Address Registers Format

DMA0's Destination Address register is cleared to 0 by a reset; all other Source and Destination Address registers are unaffected by a reset.

All DMA registers are located in I/O page FF_H. The DMA Master Control register is accessed at I/O port address FFxx1F. Table 9-8 lists the I/O port addresses for the other DMA registers. All DMA registers can be read or written using word I/O instructions.

Table 9-8. I/O Addresses of DMA Registers

	DMA Channel			
Register	**DMA0**	**DMA1**	**DMA2**	**DMA3**
Destination Address (bits 0–11)	FFxx00	FFxx08	FFxx10	FFxx18
Destination Address (bits 12–23)	FFxx01	FFxx09	FFxx11	FFxx19
Source Address (bits 0–11)	FFxx02	FFxx0A	FFxx12	FFxx1A
Source Address (bits 12–23)	FFxx03	FFxx0B	FFxx13	FFxx1B
Count	FFxx04	FFxx0C	FFxx14	FFxx1C
Transaction Descriptor	FFxx05	FFxx0D	FFxx15	FFxx1D

All addresses are in hexadecimal.
"x" means "don't care".

No checking is performed by the hardware to determine if an invalid configuration is specified in the DMA registers, such as specifying word transactions on 8-bit data bus configuration of the Z280 MPU; in such cases, DMA behavior is unpredictable.

9.5.7 DMA Sequence of Events

This section describes a typical sequence of events when a DMA channel is used in flowthrough or flyby mode to control data transfers.

Before a DMA channel can begin operation, that DMA channel must be configured for the particular application by loading its Destination Address, Source Address, Count, and Transaction Descriptor registers. DMA operations cannot take place while the EN bit in the Transaction Descriptor register is cleared to 0. Thus, the EN bit should be cleared to zero while configuring the DMA channel, and set to 1 as the last step in the configuration process; the EN bit can be set at the same time that the other bit fields in the Transaction Descriptor register are specified.

Once the EN bit is set to 1, the DMA channel requests use of the system bus only after an active $\overline{RDY}$ signal is received. The $\overline{RDY}$ signal is sampled by the DMA on the rising edge of each processor clock cycle. For DMA0 and DMA1, the $\overline{RDY}$ signal is the logical OR of the external $\overline{RDY}$ input and the software $\overline{RDY}$ bit in the DMA Master Control register.

When the system bus is available for DMA transfers, the highest priority DMA channel with a request pending becomes the bus master. The priority of the on-chip DMA channels from highest to lowest is DMA0, DMA1, DMA2, and DMA3. The external Bus Request input has the next lowest priority after the on-chip DMA channels.

The number of data transfers performed by a DMA that has gained control of the bus is determined by the current transfer mode (single transaction, burst, or continuous) and the contents of the Count register. A DMA channel in single transaction mode relinquishes the bus after a single data transfer; a DMA channel in burst mode relinquishes the bus when $\overline{RDY}$ is deasserted or when terminal count is reached; a DMA channel in continuous mode relinquishes the bus when the terminal count is reached. Regardless of the transfer mode, a DMA channel will relinquish the bus if an $\overline{EOP}$ is signalled or the terminal count is reached.

If the destination for a DMA-controlled data transfer is a memory location that corresponds to an entry in the on-chip memory (in either the cache or fixed-address mode), the on-chip memory is updated to reflect the new contents of that memory location.

For each DMA-controlled data transfer on the bus, that DMA's Count register is decremented by 1, regardless of the size of the data transferred. The Destination Address and Source Address registers might also be incremented or decremented, as determined by the DAD, SAD, and ST fields in the Transaction Descriptor register. When a DMA operation reaches completion, either by assertion of an $\overline{EOP}$ signal or by reaching terminal count (a count of 0) in the Count register, the EN bit in the Transaction Descriptor register is automatically cleared to 0. If the IE bit is set to 1, an interrupt request to the CPU is generated. If the DMA operation terminated due to an active $\overline{EOP}$ signal, the EPS status bit is set to 1; if the DMA operation terminated due to reaching terminal count, the TC status bit is set to 1.

9.5.8 DMA Programming: Linked DMAs

When two DMA channels are linked together, the master DMA's registers are written via memory-to-peripheral data transfers initiated by the linked DMA. Thus, to begin DMA operations, the linked DMA must be programmed to load the master DMA. While the linked DMA is being configured, the master DMA must be prohibited from asserting a $\overline{RDY}$ signal to the linked DMA. The internal $\overline{RDY}$ signal from the master DMA to the linked DMA is controlled by the TC status bit of the master DMA; therefore, before configuring the linked DMA, the TC bit of the master DMA's Transaction Descriptor register should be written with a 0. Then, the linked DMA is configured by writing to its registers. Finally, the TC bit in the master DMA should be set to 1; this causes the internal $\overline{RDY}$ signal to the linked DMA to go active, which in turn causes the linked DMA to request the bus and, upon acknowledgement of that request, initiates the transactions that program the master DMA.

The linked DMA must be configured for flowthrough-type data transfers. The transfer size must match the size of the external data bus (that is, byte for Z80 bus configurations and word for Z-BUS configurations). The Source Address register is loaded with the starting address of the memory block that holds the data to be written to the master DMA's registers; for the Z-BUS, this starting address must be even-valued (A0=0). The SAD field of the Transaction Descriptor register should specify an auto-increment or auto-decrement of the memory address. The Destination Address register must be set to $FFxx00_H$ when DMA2 is the linked DMA, or $FFxx08_H$ when DMA3 is the linked DMA ("x" means don't care). The DAD field in the linked DMA's Transaction Descriptor register should be set to 100_H (auto-increment I/O address). Burst mode transactions must be specified. The contents of the Count register vary depending on the number of times that the linked DMA is required to reconfigure the master DMA.

When the master DMA has completed a transaction (terminal count is reached), an internal $\overline{RDY}$ signal to the linked DMA is activated. If the linked DMA is enabled, the linked DMA will generate the transactions that program the master DMA's registers. (The linked DMA's external $\overline{RDY}$ input is ignored when DMA linking is specified.)

When the linked DMA loads the master DMA's registers, the registers are written in the following order:

- Destination Address register (least significant word)
- Destination Address register (most significant word)
- Source Address register (least significant word)
- Source Address register (most significant word)
- Count register
- Transaction Descriptor register

After the six words have been written to the master DMA, the master DMA deasserts the ready signal to the linked DMA and begins the new transfer operation. For Z-BUS configurations of the Z280 MPU, the linked DMA uses six word transactions on the bus to program the master DMA; for Z80 Bus configurations, the linked DMA uses twelve byte transactions to program the master DMA, with the least significant byte of each word being transferred first.

Both the master and linked DMAs can be programmed to generate an interrupt request to signal the end of DMA activity. If the IE bit of the master DMA is set, an interrupt request is generated when the master DMA reaches terminal count and the linked DMA's TC bit is set (that is, when the last block has been transferred), or if $\overline{EOP}$ is asserted. If the IE bit in the linked DMA is set, an interrupt request is generated when the linked DMA reaches terminal count (that is, when the last block transfer has been programmed into the master DMA), or if $\overline{EOP}$ is asserted.

9.5.9 DMA Programming: DMAs Linked to UART

The DOR and DIT bits of the DMA Master Control register specify whether DMA0 is linked to the UART receiver and DMA1 is linked to the UART transmitter, respectively.

When DMA0 is linked to the UART receiver, the state of the Source Address register and the SAD field in the Transaction Descriptor register do not affect DMA operation. The Destination Address register is programmed with the starting address of the memory area or the address of the I/O device that will be used to store the received data; if the destination port is a memory block, the DAD field should specify an auto-increment or auto-decrement of the memory address. Flowthrough-type transactions and the byte transfer size must be specified. Single, burst, or continuous mode operation can be used.

When DMA1 is linked to the UART transmitter, the Source Address register is programmed with the starting address of the memory area or the address of the I/O device that holds the data to be transmitted; if the source is a memory area, the SAD field should specify an auto-increment or auto-decrement of the memory address. The Destination Address register must be set to $xxxx18_H$, and the DAD field to a 110_H. Flowthrough type transactions and the byte transfer size must be specified. Single, burst, or continuous mode operation can be used.

9.6 UART

The on-chip universal asynchronous receiver/transmitter (UART) provides the Z280 MPU with serial I/O capability. The full-duplex UART transmits and receives serial data using any common asynchronous data communication protocol.

Figure 9-12 illustrates the general format for an asynchronous transmission using the Z280 MPU's UART. Characters can contain five, six, seven, or eight bits, plus an optional even or odd parity bit. The transmitter can supply one or two stop bits per character. Break outputs can be produced by the transmitter at any time under program control; the receiver can detect breaks as well as parity errors, framing errors, and overrun errors. Transmission and reception are performed independently.

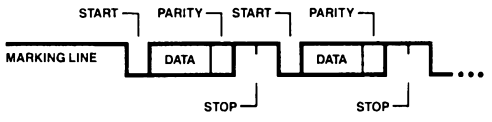


Figure 9-12. General Format for an Asynchronous Transmission

The UART uses the same clock frequency for both the transmitter and the receiver. The UART's clock input can be generated externally or internally. For externally generated clocks, Counter/Timer 1's input line is used as the source of the UART's clock in addition to being an input to the counter/timer. The maximum external clock frequency is the CPU clock divided by 4. Alternately, the UART's clock can be provided by the output pulse from Counter/Timer 1, allowing the internal processor clock to be used for bit rate generation. The UART's clock input is further scaled by a factor of 1, 16, 32, or 64 for clocking the transmitter and receiver.

The UART can be used in an interrupt-driven or polled environment. If enabled, separate transmit and receive interrupt requests are generated by the UART. Transmit interrupts occur when the transmitter's data buffer is emptied, and receive interrupts occur when an entire character is received or an error is detected. In polled environments, status bits in UART registers can be read to determine if the transmit buffer is empty or receive buffer is full. As described in section 9.5.9, DMA channel 0 can be linked to the receiver and DMA channel 1 to the transmitter to provide for DMA-controlled transfers between the UART and memory.

The UART uses two external pins, Transmit (Tx) and Receive (Rx). Data that is to be transmitted is placed serially on the Transmit pin and data that is to be received is read from the Receive pin.

The UART contains five registers. UART operation is controlled by three registers: the UART Configuration register, which contains controls for both the transmitter and receiver, the Transmitter Control/Status register, and the Receiver Control/Status register. Received data is read from the Receive Data register, and data to be transmitted is written to the Transmit Data register.

9.6.1 Transmitter Operation

Transmit operations are performed only when the Transmitter Enable bit in the Transmitter Control/Status register is set to 1. In order to transmit data, the data character is written to the Transmit Data register. The UART automatically adds the start bit, the programmed parity bit (if so specified), and the programmed number of stop bits to the data character to be transmitted. The number of bits per character, the number of stop bits per character, and the type of

parity (even, odd, or none) is determined by the contents of the UART Configuration register. When the transmit character size is five, six, or seven bits, the unused most significant bits in the Transmit Data register are ignored by the UART.

Serial data is shifted out of the transmitter on the Tx pin at a rate equal to 1, 1/16th, 1/32nd, or 1/64th of the clock signal supplied to the UART, as determined by the contents of the UART Configuration register. Serial data is shifted on the falling edge of the clock input.

The Tx output line is held high (marking) when the transmitter has no data to send or is disabled. If transmit interrupts are enabled, an interrupt request is generated when the Transmit Data register is emptied. Under program control, break conditions can be generated, wherein the Tx line is held low (spacing) until the break command is cleared.

9.6.2 Receiver Operation

Receive operations are performed only when the Receiver Enable bit in the Receiver Control/Status register is set to 1. A low (spacing) condition on the Receive input line indicates a start bit; if the low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is sampled at mid-bit times until the entire character is assembled. Thus, reception is protected from transients on the input line by checking for a valid start bit one-half bit time after detecting a high-to-low transition on the Receive input; if the low does not persist (as with a transient), the character assembly process is not started. If the bit time is one clock period (the x1 clock mode), bit synchronization must be accomplished externally; received data is sampled on the rising edge of the clock.

Received characters are read from the Receive Data register. If parity is enabled, the parity bit is assembled as part of the character for character lengths other than eight bits. If the resulting character is still less than eight bits, 1's are appended in the unused high-order bit positions. For example, Figure 9-13 illustrates how the character is assembled in the Receive Data register when receiving 5-bit characters with parity.

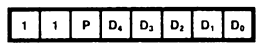


Figure 9-13. Byte Assembled by Receiver for 5-bit Character with Parity

For each character assembled by the receiver, error flags in the Receiver Control/Status register indicate if an error condition was detected. These flags are loaded when the character assembly process is completed--that is, when the character is loaded into the Receive Data register from the receiver's shift register. The receiver checks for parity errors, framing errors, and overrun errors for each received character.

A parity error occurs when the parity bit of the received character does not match the programmed parity, as determined by the contents of the UART Configuration register.

A framing error occurs if a character is assembled without any stop bits (that is, if a low level is detected for the stop bit). A built-in checking process prevents a framing error from being interpreted as a new start bit; detection of a framing error results in the addition of one-half of a bit time to the point at which the search for a new start bit is begun.

An overrun error occurs if a new character is assembled and loaded into the Receive Data register before the previous character has been read from that register. Since the receiver is buffered by the Receive Data register in addition to the receiver shift register, ample time is available for responding to a receiver interrupt and accepting a received character before the next character is assembled by the receiver.

9.6.3 UART Registers

UART operation is controlled by three 8-bit registers: the UART Configuration register, Transmitter Control/Status register, and Receiver Control/Status register. Data to be transmitted is written to an 8-bit Transmit Data register, and received data is read from an 8-bit Receive Data register. All UART registers are accessed using byte I/O instructions.

9.6.3.1 UART Configuration Register

The 8-bit UART Configuration register (Figure 9-14) contains control information for both the receiver and transmitter. The control fields within this register are described below.

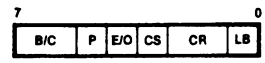


Figure 9-14. UART Configuration Register

Loop Back Enable (LB). When set to 1, the UART is in local loopback mode; in this mode, the internal transmit data line is tied to the internal receiver input line and the external receiver input pin is ignored. Thus, all transmitted data is automatically received. When this bit is cleared to 0, the transmitter and receiver operate independently.

Clock Rate (CR). This 2-bit field determines the multiplier between the UART clock and data rates (that is, the number of clocks per bit time), as specified in Table 9-9. The same data rate is used by both the transmitter and receiver. If the X1 clock rate is selected, bit synchronization must be accomplished externally. In the X1 mode, the transmitter sends data on the falling edge of the clock and the receiver samples data on the rising edge of the clock.

Table 9-9. CR Field of UART Configuration Register

CR Field	UART Clock Rate
00	X1
01	X16
10	X32
11	X64

Clock Select (CS). The state of this bit specifies the clock input for the UART. When this bit is set to 1, counter/timer 1's output pulse supplies the UART clock. When this bit is cleared to 0, counter/timer 1's clock input pin provides the UART clock signal, thus allowing the use of an externally-generated clock. The content of the IPA field of C/T 1's Configuration register does not affect these UART clocking modes.

Parity (P). When set to 1, an additional bit position (in addition to the number of bits per character specified in the BC field) is added to each transmitted character and expected in each received character; this additional bit is the parity bit. Parity bits in received characters are assembled as part of the character for character lengths of less than 8 bits.

Parity Even/Odd (E/O). If parity is specified (P = 1), this bit determines whether an odd or even parity bit is added to transmitted characters and whether odd or even parity is checked for in received characters. E/O = 1 specifies even parity and E/O = 0 specifies odd parity. If P = 0, then this bit is ignored.

Bits per Character (B/C). This 2-bit field determines the number of bits per character in both the transmitter and receiver, as specified in Table 9-10. If this field is changed while a character is being transmitted or received, the results are unpredictable.

Table 9-10. BC Field of UART Control Register

BC Field	Bits per Character
00	5
01	6
10	7
11	8

A reset clears the UART Configuration register to all zeros, unless bootstrap mode is selected (see section 9.7).

9.6.3.2 Transmitter Control/Status Register

The 8-bit Transmitter Control/Status register, shown in Figure 9-15, specifies the operation of the UART transmitter, as described below.

Figure 9-15. Transmitter Control/Status Register

Transmitter Buffer Empty (BE). This status bit is automatically set to 1 whenever the Transmit Data register becomes empty and cleared to 0 whenever a character is loaded into the Transmit Data register. The BE bit is controlled by the UART circuitry; it can be read via an I/O read but is unaffected by an I/O write to this register. A reset loads a 1 into this bit.

Value (VAL). This bit determines the value of the bits transmitted by the UART when the FRC bit is set to 1 and "dummy" characters are loaded into the Transmit Data register. When the VAL bit is set to 1, a mark character (all 1s) is transmitted; when the VAL bit is cleared to 0, a break character (all 0s) is transmitted.

Force Character (FRC). When this bit is set to 1, writing a character to the Transmit Data register causes the transmitter output to be held high or low (depending on the state of the VAL bit) for the length of time required to transmit the character. Note that characters written to the Transmit Data register are not themselves transmitted while FRC is set to 1. When FRC is cleared to 0, the transmitter operates normally, sending characters that are written to the Transmit Data register.

Send Break (BRK). When this bit is set to 1, the transmitter is forced into the spacing condition, wherein the transmit data output is forced to 0. When this bit is cleared to 0, normal transmitter operation resumes.

Stop Bits (SB). The state of this bit determines the number of stop bits appended to each character by the transmitter. Setting this bit to 1 specifies two stop bits per character; clearing this bit to 0 specifies one stop bit per character.

Transmitter Interrupt Enable (IE). When this bit is set to 1, an interrupt request is generated whenever the Transmit Data register is emptied. When this bit is cleared to 0, no transmsit interrupts are generated.

Transmitter Enable (EN). When this bit is cleared to 0, the transmitter is disabled and the transmitter output line is held high (marking). When this bit is set to 1, the transmitter is enabled and operates as specified by the UART Configuration register and the Transmitter Control/Status register. If this bit is cleared while a character is in the process of being transmitted, transmission of that character is completed.

A reset sets the Transmitter Control/Status register to a 01_H. Bit 5 of this register is not used.

9.6.3.3 Receiver Control/Status Register

The 8-bit Receiver Control/Status register, shown in Figure 9-16, specifies the operation of the UART receiver, as described below.

Figure 9-16. Receiver Control/Status Register

Receiver Error (ERR). This bit is the logical OR of the PE, OVE, and FE bits.

Framing Error (FE). This bit is automatically set to 1 if the receiver detects a framing error when assembling the received character. Detection of a framing error adds an additional one-half bit time to the character to ensure that the framing error is not interpreted as a new start bit. This bit is not latched; once set, it remains set only until a new character is assembled and shifted into the Receive Data register.

Parity Error (PE). When parity is enabled (P = 1 in the UART Configuration register) this bit is automatically set to 1 if a character is received without the specified parity. This bit is latched; once set, it remains set until cleared via software.

Receiver Overrun Error (OVE). This bit is automatically set to 1 if a new character is assembled and loaded into the Receive Data register before the previous character has been read from that register. Only the most recently received character is flagged with this error, but once this character is read, the OVE bit remains latched until cleared via software.

Receiver Character Available (CA). This bit is automatically set to 1 when a received character is available in the Receive Data register and automatically cleared to 0 when the Receive Data register is read. This bit is controlled by UART circuitry; it can be read via an I/O read but cannot be altered by an I/O write to this register.

Receiver Interrupt Enable (IE). When this bit is set to 1, an interrupt request is generated whenever the receiver has a character available in the Receive Data register or when a receiver error is detected.

Receiver Enable (EN). When set to 1, receiver operation is enabled. This bit should be set after programming the UART Configuration register.

The Receiver Control/Status register is cleared to all zeros by a reset, unless bootstrap mode is selected (see section 9.7). Bit 5 of this register is not used.

All UART registers are in I/O page FE and are accessed via byte I/O instructions. Table 9-11 lists the I/O port addresses for the UART registers.

Table 9-11. I/O Addresses of UART Registers

Register	I/O Port Address
UART Configuration Register	FExx10
Transmitter Control/Status Register	FExx12
Receiver Control/Status Register	FExx14
Receive Data Register	FExx16
Transmit Data Register	FExx18

All addresses are in hexadecimal.
"x" means "don't care".

9.6.4 UART Operation

Operation of the UART's transmitter and receiver are enabled by the Transmitter Enable and Receiver Enable control bits in their respective control/status registers. Before enabling the UART by setting one of those bits, the UART's configuration must be determined by programming the UART Configuration register. If the UART Configuration register is to be altered during system operation, the transmitter and receiver should be disabled before writing to the Configuration register, and then re-enabled afterwards.

Once enabled, the UART can be used in an interrupt-driven or polled environment. Separate transmit and receive interrupts are controlled by the interrupt enable bits in the control/status registers. Receive interrupts are generated whenever a new character is available in the Receive Data register or when an error is detected. Transmit interrupts are generated whenever the Transmit Data register is emptied.

For polled environments, the Character Available bit in the Receiver Control/Status register must be monitored to determine when a character is to be read from the Receive Data register; this bit is automatically cleared when the received data is read. For transmitting characters, the Transmit Buffer Empty flag should be checked before writing to the Transmit Data register to prevent the overwriting of transmitted data.

The error flags in the Receiver Control/Status register are loaded at the same time that the received data character is moved from the receiver's shift register to the Receive Data register. Since the parity and receiver overrun error flags are latched, the error status reflects any errors in the current character in the Receive Data register plus any parity or overrun errors that have been detected since the last write to the Receiver Control/Status register. To maintain correspondence between the state of the error flags and the data in the Receive Data register, the flags in the Receiver Control/Status register should be read before the data.

Once the transmitter has been enabled, there are two ways to produce a break output on the transmit data line. Setting the BRK bit in the Transmitter Control/Status register forces a break condition on the transmit data output until that bit is cleared. Alternatively, setting the FRC bit to 1 and clearing the VAL bit to 0 causes a break condition on the transmit data output each time a character is loaded into the Transmit Data register; this break output persists for the same amount of time that it would have taken to transmit the data written to the Transmit Data register had the FRC bit been 0. Note that the characters written to the Transmit Data register while the FRC bit is set to 1 are not actually transmitted.

9.7 UART BOOTSTRAPPING OPTION

The on-chip UART and DMA Channel 0 can be used to automatically initialize the Z280 MPU's memory with values received by the UART following a reset. This system bootstrapping capability permits ROMless system configurations, where memory is initialized using a serial link prior to the first Z280 MPU instruction fetch after the reset.

As described in Section 3.2.1 and Chapter 11, bootstrap mode is selected by driving $\overline{WAIT}$ low and AD_6 high while $\overline{RESET}$ is asserted. The appropriate UART and DMA registers are automatically programmed as shown in Table 9-12 as a result of selecting bootstrap mode. The UART is initialized to receive data in 8-bit characters with odd parity, an external clock source, and a x16 clock rate. DMA Channel 0 is initialized with the link to the UART receiver and end-of-process capability enabled, and set up for flowthrough byte transfers in continuous mode. The destination address starts at memory location 0, with an autoincrement after each transfer, and a transfer count of 256 (100_H).

Table 9-12. Reset Value of UART and DMA Registers When Bootstrap Mode is Selected

Register	Initial Hex Value
UART Registers	
UART Configuration register	E2
Receiver Control/Status register	80
DMA Registers	
DMA Master Control register	0011
Channel 0 Transaction Descriptor register	8100
Channel 0 Destination Address register	000000
Channel 0 Source Address register	Undefined
Channel 0 Count register	0100

If bootstrap mode is specified, the Z280 CPU automatically enters an idle state when $\overline{\text{RESET}}$ is deasserted. A minimum of 15 processor clock cycles must elapse after $\overline{\text{RESET}}$ is deasserted before tranmission of data to the UART receiver begins. DMA Channel 0 is then used to transfer characters received by the UART into memory. The data received is placed in memory starting at physical address 0. If an error is detected by the UART receiver, the Transmit Output (Tx) line is driven low; external circuitry can use this signal to restart the initialization procedure, if so desired. After 256 bytes of data have been received and transferred to memory, the Z280 CPU automatically begins execution with an instruction fetch from memory location 0.

Chapter 10.
Multiprocessor Configurations

10.1 INTRODUCTION

The Z280 MPU architecture provides support for four types of multiprocessor configurations (Figure 10-1): slave processors, tightly coupled multiple CPUs, loosely coupled multiple CPUs, and coprocessors.

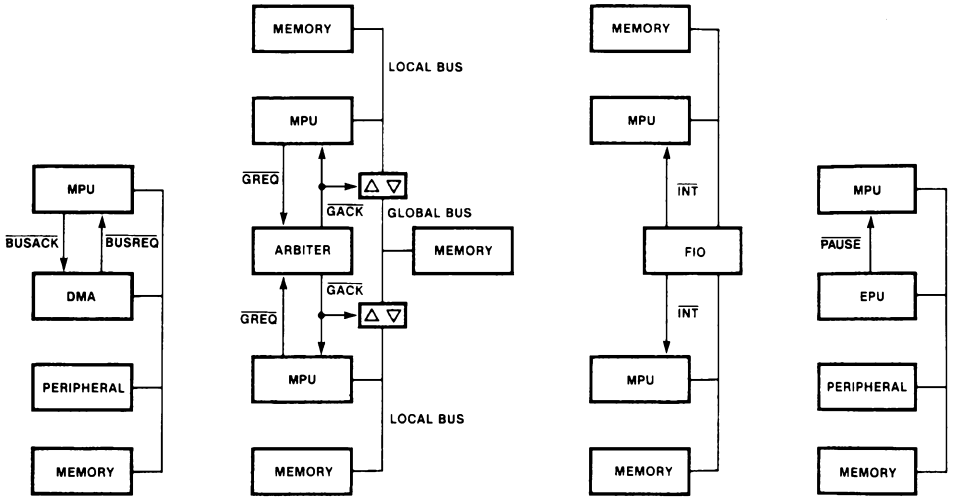


Figure 10-1. Multiprocessor Configurations

10.2 SLAVE PROCESSORS

Slave processors, such as the Z8016 DMA Transfer Controller or other DMA devices, perform dedicated functions asynchronously to the CPU. The CPU and slave processors share a local bus, where the CPU is the default bus master. In order for a slave processor to use the bus, it must request control of the bus from the CPU and receive an acknowledgement of that request.

Two Z280 MPU signals are provided for supporting slave processors: $\overline{\text{BUSREQ}}$ and $\overline{\text{BUSACK}}$. A bus request is initiated by pulling the $\overline{\text{BUSREQ}}$ input low. Several bus requestors may be wire-ORed to the $\overline{\text{BUSREQ}}$ pin; priorities are resolved external to the MPU, usually by a priority daisy chain. The external $\overline{\text{BUSREQ}}$ signal generates an internal, synchronous $\overline{\text{BUSREQ}}$. If this signal is active at the beginning of any bus cycle, the Z280 MPU will relinquish the bus at the end of that bus cycle (with the exception of the TSET instruction, where the read-modify-write cycle is atomic). The MPU suspends execution of the current instruction and gives up control of the bus by 3-stating all address, address/data, bus timing, and bus status output pins. The $\overline{\text{BUSACK}}$ output is then asserted, signaling that the bus request has been accepted and the bus is free for use by the slave processor. The Z280 MPU remains in the bus disconnect state until $\overline{\text{BUSREQ}}$ is deasserted.

The $\overline{\text{BUSREQ}}$ input is sampled during each processor clock period by the external bus interface logic of the Z280 MPU. If $\overline{\text{BUSREQ}}$ is sampled active low while the Z280 MPU is involved in an internal operation, the external bus is relinquished to the bus requestor immediately. Internal processing can continue until a transaction involving the external bus is required; the MPU then suspends activity until regaining control of the bus. If $\overline{\text{BUSREQ}}$ is sampled active during a CPU-generated

transaction on the external bus, the bus is not relinquished nor CPU activity suspended until the current transaction is completed.

The Z280 MPU regains control of the bus after $\overline{BUSREQ}$ rises, continuing execution from the point at which it was suspended. Any bus requestor desiring control of the bus must wait at least two bus cycles after $\overline{BUSREQ}$ has risen before asserting $\overline{BUSREQ}$ again.

In the case of simultaneous bus requests from multiple sources, the on-chip DMA channels have higher priority than external slave processors in Z280 MPU systems. After reset, the Z280 MPU acknowledges an active $\overline{BUSREQ}$ signal before performing any transactions.

10.3 TIGHTLY COUPLED MULTIPLE PROCESSORS

Tightly coupled multiple CPUs execute independent instruction streams from their own (local) memory locations and communicate through shared memory locations on a common (global) bus. Each CPU is the default master of its local bus, but the global bus master is chosen by an external arbiter.

The Z280 MPU's multiprocessor mode of operation supports tightly coupled multiple CPU configurations. This mode is also useful when configuring the Z280 MPU as an I/O processor in a distributed processing system. Multiprocessor mode is selected by setting the Multiprocessor Configuration Enable (MP) bit in the Z280 CPU's Bus Timing and Initialization register (see Section 3.2.1). While in the multiprocessor mode, the Z280 MPU is able to support both a local bus and a global bus. The Z280 CPU is the default bus master of the local bus, but must make a request and receive an acknowledgement before performing transactions on the global bus. Only memory transactions can be performed on the global bus; I/O transactions always use the local bus. The range of memory addresses dedicated to the global and local buses is determined by the contents of the CPU's Local Address register.

While in the multiprocessor mode, Counter/Timer 0's I/O and IN pins are used as global bus request ($\overline{GREQ}$) and global bus acknowledge ($\overline{GACK}$) signals, respectively. $\overline{GREQ}$ is a three-state output; an active low signal on this line requests use of the global bus. An active low level on the $\overline{GACK}$ input acknowledges a global bus request.

10.3.1 The Local Address Register

During each memory transaction while in multiprocessor mode, the Z280 CPU uses the Local Address register to determine if that transaction is to occur on the local or global bus. The Local Address register includes a 4-bit Base field and a 4-bit Match Enable field (Figure 10-2). For each bus transaction, the four most-significant bits of the physical address (address bits A_{20} through A_{23}) are compared with the 4-bit Base field; the Match Enable field specifies which bits are going to be used during this comparison. If all the corresponding address bits match the Base field in the bit positions specified by the Match Enable field, then the bus transaction can proceed on the local bus without requesting the global bus. If there is a mismatch in at least one specified bit position, then the global bus is requested and the bus transaction does not proceed until the global bus acknowledge signal is asserted. (See section 3.2.3.)

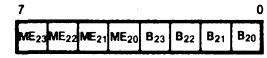


Figure 10-2. Local Address Register

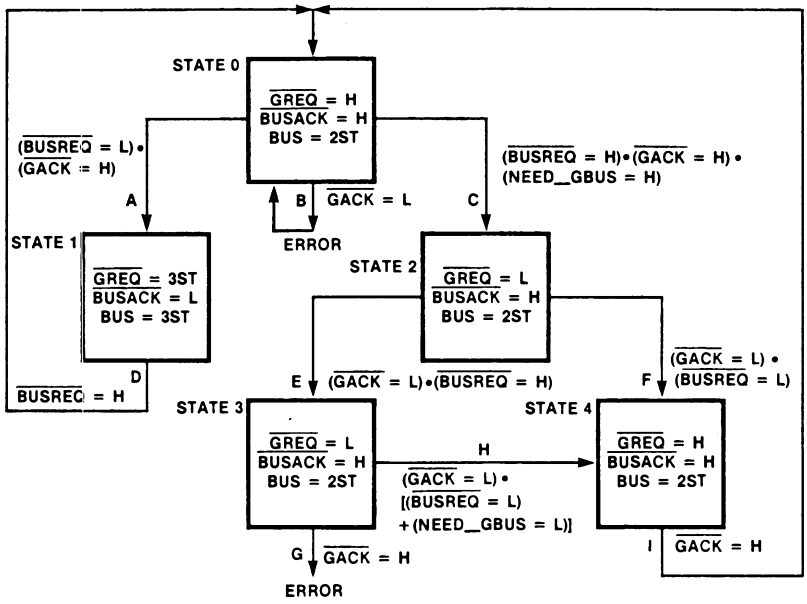
10.3.2 Bus Request Protocols

While in the multiprocessor mode, the $\overline{BUSREQ}$ and $\overline{BUSACK}$ signals control use of the local bus in the same manner as described in section 10.2. When a local bus request is granted, as indicated by an active $\overline{BUSACK}$ signal, the CPU places all output signals, including $\overline{GREQ}$, in the high-impedance state.

When in control of its local bus, a Z280 CPU can initiate transactions with devices on the global bus that are shared with other CPUs. At any one time, only one CPU can control transactions on the global bus. Control of the global bus is arbitrated by external circuitry. Before initiating a transaction on the global bus, the CPU requests control of the global bus from the external arbiter circuitry by asserting $\overline{GREQ}$ and waiting for an active $\overline{GACK}$ in response. (The timing diagrams for global bus requests are shown in Figures 12-15 and 13-19.) The $\overline{GACK}$ input is asynchronous to the CPU clock; the Z280 CPU synchronizes $\overline{GACK}$ internally. Once $\overline{GACK}$ is asserted, the CPU performs the transaction on the global bus. The CPU then deasserts $\overline{GREQ}$ and waits

for the arbiter circuit to deassert $\overline{GACK}$. The CPU always relinquishes the global bus by deasserting $\overline{GREQ}$ after each global transaction is completed, except during execution of a Test and Set (TSET) instruction (both the data read and write are completed before relinquishing the global bus) or during a burst-mode memory transfer (the entire sequence of burst-mode memory reads is completed before relinquishing the global bus).

A state diagram of the bus request protocol is shown in Figure 10-3.



NOTES: Interface signals are High (H), Low (L), High or Low (2ST), or 3-stated (3ST).

NEED_GBUS is an active High signal internal to the CPU.

Transition Legend	
A	A local bus request occurs.
B	The global bus arbiter grants control of the global bus when no global bus request is pending. This is an error. The CPU remains in State 0.
C	The CPU requests the global bus in response to the internally generated signal NEED_GBUS.
D	The local bus master relinquishes the bus.
E	The global bus arbiter grants the global bus to the CPU while no local bus request is pending.
F	The global bus arbiter grants the global bus to the CPU while a local bus request is pending. The local bus request has preempted the CPU.
G	The global bus arbiter reclaims the global bus before the CPU relinquishes the global bus. This is an error. The CPU's response to this error is undefined.
H	The CPU relinquishes control of the global bus when it no longer needs the global bus or in response to a local bus request.
I	The global bus arbiter reclaims the global bus.

State Legend	
State 0	The CPU controls the local bus and is neither requesting nor controlling the global bus. The CPU can perform transactions on the local bus.
State 1	The CPU has granted the local bus. The CPU cannot perform transactions.
State 2	The CPU controls the local bus and is requesting the global bus. The CPU cannot perform transactions.
State 3	The CPU controls the local and global buses. The CPU can perform transactions on the global bus.
State 4	The CPU controls the local bus and is relinquishing control of the global bus. The CPU cannot perform transactions.

Figure 10-3. State Diagram for CPU Bus Request Protocol

While a Z280 CPU is asserting $\overline{GREQ}$ and waiting for an active $\overline{GACK}$, if $\overline{BUSREQ}$ is asserted before $\overline{GACK}$, the CPU releases the global bus request after $\overline{GACK}$ is asserted without performing any transactions.

The on-chip DMA channels may also initiate transactions on the global bus. During each DMA-controlled transaction, memory addresses generated by a DMA channel are compared to the contents of the Local Address register to determine if the global bus is to be requested, in the same manner as CPU-controlled bus transactions.

If the automatic memory refresh mechanism is enabled, refresh cycles are inhibited while either the CPU or a DMA channel has requested the global bus but not yet received the global bus acknowledge. No refresh transactions are ever performed on the global bus.

10.3.3 Examples of the Use of the Global Bus

The Z280 MPU's multiprocessor mode of operation facilitates the development of tightly coupled multiprocessor systems and systems using the Z280 MPU as a front-end I/O processor.

Figure 10-4 is a block diagram illustrating the use of multiple Z280 MPUs as tightly-coupled processors. Access to the global memory via the global bus is controlled by a centralized bus arbitration circuit. The $\overline{GACK}$ circuit controls the buffers that connect or isolate the global bus from each MPU's local bus. Each Z280 MPU can access its local memory independent of the other MPU's activity. Only one MPU at a time can access the shared global memory. Note that memory-mapped I/O devices could also be shared using the global bus.

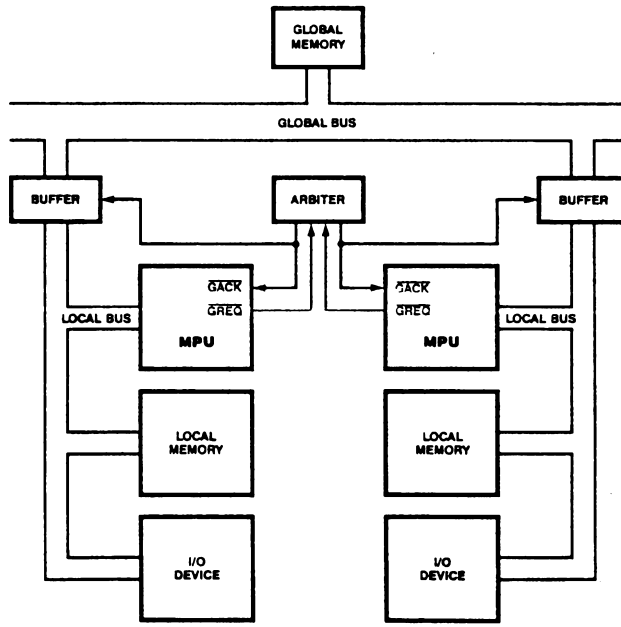


Figure 10-4. Tightly Coupled Processors with Shared Global Memory

Figure 10-5 shows a tightly coupled multiple Z280 MPU system without a global memory, where each processor can directly access the local memory of the other processor. For this system, priority resolution logic would control both the local and global bus requests. A global bus request from one processor is used to generate a local bus request to the other processor. When one processor generates a global bus request, an active $\overline{GACK}$ signal is not returned to that processor until the other processor's local bus is available, as indicated by $\overline{BUSACK}$.

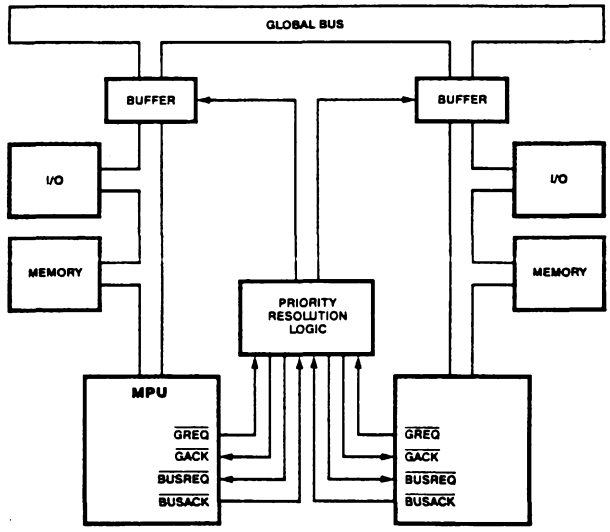


Figure 10-5. Tightly Coupled Processors without Global Memory

Although both Figure 10-4 and 10-5 show only two tightly coupled processors, more processors could be added to these systems in a similar manner.

Figure 10-6 illustrates the use of a Z280 MPU as an I/O processor in a Z8000-based system. The Z280 MPU's $\overline{\text{GREQ}}$ signal is used as the bus request signal to the Z8000 CPU; the Z8000 CPU's $\overline{\text{BUSACK}}$ signal is input directly to the Z280 MPU's $\overline{\text{GACK}}$, as well as controlling the buffers that normally isolate the Z280 MPU's local bus from the Z8000 CPU's bus.

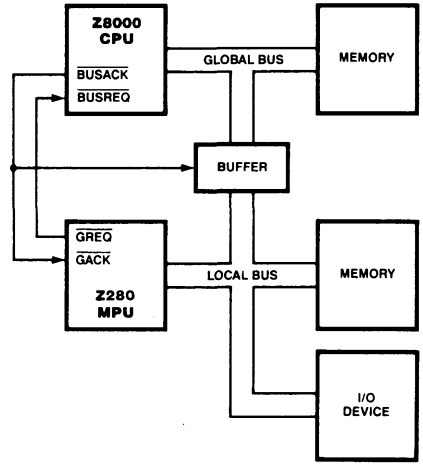


Figure 10-6. Z280 MPU as an I/O Processor

10.4 LOOSELY COUPLED MULTIPLE CPUS

Loosely coupled multiple CPUs generally communicate through a multiple-port peripheral, such as the Z8038 FIO (FIFO buffer I/O unit). The Z280 MPU's I/O and interrupt facilities and the on-chip DMA channels support loosely coupled multiprocessing with the Z280 MPU.

10.5 COPROCESSORS AND THE EXTENDED PROCESSING ARCHITECTURE

The Zilog Extended Processing Architecture (EPA) provides a flexible and modular approach to expanding the capabilities of the Z280 MPU through the use of coprocessors called Extended Processing Units (EPUs). The Extended Processing Architecture is available on the Z-BUS configurations of the Z280 MPU, but not the Z80 Bus configurations. Up to four EPUs can be connected to a single Z280 MPU.

An Extended Processing Unit is a coprocessor that can be used to execute complex, time-consuming tasks in order to unburden the CPU. EPUs connect directly to the Z-BUS; no extra external logic is required to interface an EPU to a Z280-based system (Figure 10-7). As the Z280 CPU fetches and executes instructions, the EPU continuously monitors the instruction stream on the bus. A special group of instructions, called extended instructions, are processed by EPUs. When the Z280 CPU encounters an extended instruction, it performs any specified data transactions, but otherwise assumes that the instruction will be recognized and handled by an EPU. (In systems without EPUs, extended instructions can be used to generate a trap condition.) Thus, when EPUs are added to a system, the instruction set is expanded to include the extended instructions applicable to those EPUs, thereby boosting the processing power of the whole system. The Z280 CPU and EPUs work together like a single central processor; a system with EPUs can be thought of as a system whose central processor consists of 1 + N separate devices, where N is the number of EPUs in the system.

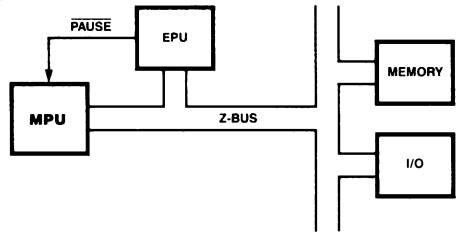


Figure 10-7. EPU Connection in Z280 MPU System

The underlying philosophy of the Extended Processing Architecture is that the CPU is an instruction processor; that is, the CPU fetches an instruction, fetches data associated with that instruction, performs the specified operation, and stores the result. Extending the number of operations that can be performed does not affect the instruction fetch and address calculation portion of the CPU activity. The extended instructions exploit this feature. The CPU is responsible for fetching instructions, performing address calculations, and generating the timing signals for bus transactions; however, the actual data manipulation for extended instructions is handled by an EPU. Both the CPU and EPUs are, therefore, controlled via a single instruction stream, eliminating many significant system software and bus contention problems that can occur with other multiprocessing configurations.

10.5.1 Extended Instructions

Extended Processing Units connect directly to the Z-BUS and continuously monitor the instruction stream. When the template portion of an extended instruction is fetched from memory, the appropriate EPU will detect that the instruction is meant for it and respond to the instruction. The CPU is always responsible for fetching instructions and delivering operands to the EPUs. The EPUs recognize the extended instruction templates and execute them, using data supplied with the template and/or data already within internal EPU registers.

There are four types of extended instructions in the Z280 instruction set: data transfers from memory to an EPU, data transfers from an EPU to memory, data transfers from an EPU to the CPU's

accumulator register, and EPU internal operations. Twenty-two instruction opcodes are used to implement these operations. Each extended instruction opcode includes two parts: a two- or four-byte instruction opcode used by the Z280 CPU to determine its activity and the address of the memory operand, and a four-byte instruction "template" that specifies the EPU activity. Six operand addressing modes are supported by the instructions that specify transfers between EPU registers and main memory: Direct Address, Indirect Register, Indexed, Stack Pointer Relative, Program Counter Relative, and Base Index. (See section 5.4.10 for a description of the extended instructions.)

In addition to the hardware-implemented capabilities of the EPA, there is an extended instruction trap mechanism that permits software simulation of EPU functions. The state of the EPU Enable bit in the CPU's Trap Control register indicates whether EPUs are present in the system (see section 3.3.5). If the EPU Enable bit is cleared to 0, indicating that there are not EPUs in the system, the CPU will execute an Extended Instruction trap if an extended instruction is encountered in the instruction stream. The service routine for this trap could perform a software simulation of an EPU's functions. This trap mechanism facilitates the design of systems in which EPUs are not present but may be added later. Initially, the "extended" function is executed as the Extended Instruction trap service routine; when EPUs are added to the system, the trap routine is eliminated and the EPU Enable bit is set to 1. This change would be transparent to applications programs. (The Extended Instruction trap is described in section 6.3.1.)

10.5.2 Extended Instruction Execution Sequence

The CPU and EPU instruction execution sequence is diagrammed in Figure 10-8. When the CPU fetches an extended instruction, the EPU Enable bit in the Trap Control register is examined. If the EPU Enable bit is a 0, an Extended Instruction trap is executed. If the EPU Enable bit is a 1, indicating that there is an EPU in the system, then the CPU fetches the four-byte instruction template from memory. The fetching of the template is indicated by the ST_3-ST_0 status lines from the CPU. EPUs must continuously monitor the address/data bus and ST_3-ST_0 status lines for its templates. A 2-bit identification field in the template can select one of up to four EPUs for execution of a given extended instruction. If the extended instruction calls for the transfer of data between the CPU and EPU or between the EPU and memory, the CPU generates the appropriate bus transaction cycles. These transactions are identified by unique encodings of the ST_3-ST_0 status lines. The EPU monitors the status and

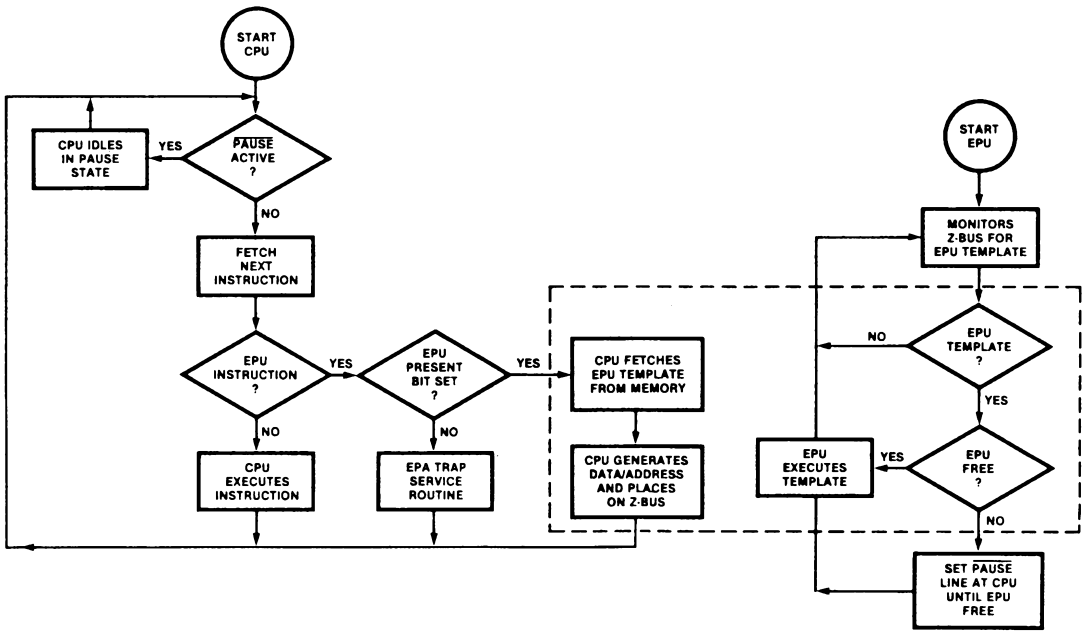


Figure 10-8. CPU-EPU Instruction Execution Sequence

timing signals output by the CPU to determine when to participate in the data transaction; the EPU supplies or captures the data when $\overline{DS}$ is active. For transactions between an EPU and memory, the CPU 3-states its address/data lines while $\overline{DS}$ is active so that the EPU or memory can supply the data. (See section 13.5.5 for a description of the bus transaction timing.)

The number and type of bus cycles required to fetch the extended instruction template depends on whether the template is aligned on an even address boundary. The four-byte long template can be fetched with two word transactions if the template begins on an even memory address or with one byte and two word transactions if the template begins at an odd memory address, as described in Table 10-1. (In the case of an odd starting address for the template, the EPU captures only the upper byte from the bus during the second word transaction.) The template is always fetched from memory using the CPU's external bus interface, regardless of the current state of the on-chip cache memory.

Table 10-1. Bus Transactions Involved in Fetch of Extended Instruction Template

Address at Template Start	Bus Cycle	Address from Z280	Byte/Word	ST_3-ST_0
Even	1	n	Word	1101
	2	n+2	Word	1100
Odd	1	n	Byte	1101
	2	n+1	Word	1100
	3	n+3	Word	1100

If the extended instruction specifies an internal EPU operation, the Z280 CPU can proceed to fetch and execute subsequent instructions. Thus, the CPU and EPUs may be processing in parallel. The $\overline{PAUSE}$ signal is used to synchronize CPU-EPU activity in the case of overlapping extended instructions. If the CPU fetches another extended instruction template intended for an EPU that is still executing a previous instruction, the EPU activates the $\overline{PAUSE}$ input to the CPU to halt further CPU activity until the EPU can finish the original operation. While $\overline{PAUSE}$ is asserted, all CPU activity is suspended except responses to refresh requests, bus requests, and resets.

CPU activity following the fetch of the extended instruction template is governed by the type of extended instruction being processed. In the case of an EPU internal operation, no further bus transactions are required by the extended instruction, so the CPU will proceed to fetch the next instruction. However, the CPU will still honor an active $\overline{PAUSE}$ input and suspend execution until $\overline{PAUSE}$ is released.

In the case of an EPU-to-CPU transfer instruction, the next non-refresh transaction following the fetch of the template (and after an active $\overline{PAUSE}$ signal is deasserted) will be the EPU-to-CPU bus transaction. EPU-to-CPU bus transactions are identified by a 1110 status code on the ST_3-ST_0 status lines and are word transactions. The address emitted by the CPU during this cycle is the memory address of the previous transaction (that is, the address used during the last fetch of the instruction template).

In the case of EPU-to-memory or memory-to-EPU transfer instructions, the next one to sixteen non-refresh transactions following the fetch of the template (and after an active $\overline{PAUSE}$ signal is deasserted) will be the appropriate data transfer cycles. Up to 16 bytes of data may be transferred as the result of a single extended instruction; the number of data transfers to be performed is encoded in the instruction template. The 1010 status code on the ST_3-ST_0 status lines identifies bus cycles that transfer data between an EPU and memory. The EPU must supply the data for write operations or capture the data for read operations during each transaction, just as if it were part of the CPU. The number and type of transactions generated also depends on whether the starting memory address of the data block to be moved is an even-valued address, as defined in Table 10-2. The case where only one byte is transferred is degenerate and shown separately in Table 10-2 for clarity. These transfers are always performed on the Z280 MPU's external bus, regardless of the current state of the on-chip cache memory.

Table 10-2. Sequence of Transactions for Data Transfers between an EPU and Memory

Starting Memory Address	Number of Bytes (n)	Byte/Word Status of Transfers	Type of Addresses	Total Number of Transactions
Even	Even	word, word....,word	All even	n/2
Even	Odd	word, word....,word, byte	All even	(n+1)/2
Even	One	byte	Even	1
Odd	Even	byte, word,...,word, byte	First odd, others even	(n+2)/2
Odd	Odd	byte, word,...,word, word	First odd, others even	(n+1)/2
Odd	One	byte	Odd	1

Chapter 11.
Reset

Hardware resets are asserted by an active $\overline{\text{RESET}}$ input and place the Z280 MPU in a known state. Optionally, the Bus Timing and Initialization register can be initialized to a system specifiable value during a reset. The $\overline{\text{RESET}}$ input is internally synchronized to the clock and then sampled at the end of every processor clock cycle. When an active $\overline{\text{RESET}}$ line is detected, the current bus transaction is allowed to be completed before starting the reset process. A reset overrides all other operations, including interrupts, traps, and bus requests. A hardware reset must be used to initialize the Z280 MPU as part of the power-up sequence.

The $\overline{\text{RESET}}$ input must be asserted for a minimum of 128 processor clock cycles. Within this time the Z280 MPU lines assume their reset values: the address and address/data lines are 3-stated and all control lines are forced High. While $\overline{\text{RESET}}$ is asserted, the clock output line (CLK) is the processor clock frequency divided by four.

When $\overline{\text{RESET}}$ is sampled high (deasserted), the state of the $\overline{\text{WAIT}}$ input is sampled. If $\overline{\text{WAIT}}$ is asserted, the contents of the AD_0-AD_7 lines are sampled on the falling edge of the processor clock and loaded into the Bus Timing and Initialization register; if this method of initialization is chosen, AD_7 must be a 1 and AD_4 must be a 0 when the bus is sampled, and the state of the AD_6 line determines whether the bootstrap mode option is selected. $\overline{\text{WAIT}}$ must be asserted for at least two processor clock cycles after $\overline{\text{RESET}}$ is deasserted in order for the Bus Timing and Initialization register, thereby specifying a bus clock frequency of one-half the processor clock, no automatic wait states when accessing the lower 8M bytes of memory, and disabling the multiprocessor mode of operation.

Table 11-1 delineates the effect of a reset on other CPU registers. A reset places the CPU in interrupt mode 0; thus, the IM field in the Interrupt Status register will be a 0. The Interrupt Vector Enable bits in the Interrupt Status register also are cleared to 0 by a reset, and the Interrupt Pending bits will reflect the current status of the interrupt requests. All other CPU and MMU registers, including the remaining registers in the CPU register file, the MMU page descriptor registers, and the Interrupt/Trap Vector Table Pointer are unaffected by a reset.

The effect of a reset on the on-chip peripherals' programmable registers is shown in Table 11-2. The on-chip counter/timers are always disabled by a reset. The on-chip DMA channels and UART are also disabled by a reset, unless bootstrap mode is selected (see Section 9.7). The counter/timers' Time Constant and Count-Time registers are unaffected by a reset. The DMA channels' Destination Address, Source Address, and Count registers also are unaffected by a reset, except for DMA Channel 0's Destination Address and Count registers.

In a multiprocessing system employing multiple Z280 MPUs with a shared bus, the internal processor clocks for the Z280 MPUs need to be synchronized. The processor clock is generated by dividing the XTAL1 input by two. The falling edge of $\overline{\text{RESET}}$ is used internally to synchronize the processor clock, and can be used to synchronize processor clocks in a multiprocessing system. If all the Z280 MPUs in the system have identical XTAL1 and $\overline{\text{RESET}}$ input signals, their internal processor clocks will be initialized in the same manner by a reset.

If an active bus request is detected on the rising edge of $\overline{\text{RESET}}$, the Z280 MPU grants the bus before fetching the first instruction from location 0. Thus, an external DMA device can initialize RAM memory before execution begins. If bus request is not asserted, the CPU begins execution with a fetch from location 0 unless bootstrap mode is in effect.

Table 11-1. Effect of a Reset on Z280 MPU and MMU Registers

Register	Value Loaded on Reset (Hexadecimal)	Comments
Program Counter	0000	
System Stack Pointer	0000	
I	00	
R	00	
Master Status	0000	System mode, Single-Step disabled, Breakpoint-on-Halt disabled All maskable interrupts disabled
Bus Timing and Control	30	No automatic wait states for I/O, upper 8M bytes of memory, or interrupt acknowledges
Bus Timing and Initialization	80	CLK output 2 × processor clock period, no automatic wait states for lower 8M bytes of memory, bootstrap mode disabled
I/O Page	00	I/O Page 0 in use
Cache Control	20	Cache enabled for instructions All valid bits cleared to 0 Burst mode disabled
Trap Control	00	EPA trap disabled, I/O not privileged
System Stack Limit	0000	System Stack Overflow Warning trap disabled
Local Address	00	All memory transactions are made to local bus
Interrupt Status	00xx	Interrupt mode 0, nonvectored interrupts, current state of interrupt requests (indicated by xx)
Interrupt/Trap Vector Table Pointer		Unaffected
CPU Registers AF, BC, DE, HL, IX, IY, AF′, BC′, DE′, HL′		Unaffected
User Stack Pointer		Unaffected
MMU Master Control	0000	MMU disabled
MMU Page Descriptor Register, Page Descriptor Register Pointer		Unaffected

Table 11-2. Effect of a Reset on Z280 On-Chip Peripheral Registers

Register	Value Loaded on Reset (Hexadecimal)	Comments
Refresh	88	Refresh enabled, rate = 32
Counter/Timers:		
Configuration	00	Timer mode, single-cycle mode
Command/Status	00	Timer disabled
DMA Channels:		
Master Control	0000*	No DMA linking, EOP disabled, Software Ready disabled
DMA0 Transaction Descriptor	0100*	DMA0 disabled, continuous mode
DMA1/2/3 Transaction Descriptor	—	EN, IE, TC, and EPS fields cleared, other fields unaffected
DMA0 Destination Address	000000	
DMA0 Count	0100	
UART:		
Configuration	00*	5 bits/character, parity disabled, external clock, × 1 clock rate, loop back disabled
Transmitter Control/Status	01	Transmitter disabled, transmit buffer empty
Receiver Control/Status	00*	Receiver disabled

*Unless bootstrap mode is selected.

Chapter 12.
Z280 Bus External Interface

12.1 INTRODUCTION

The Z280 MPU is typically only one component in a system that may include memory, peripherals, slave processors, coprocessors, and other CPUs, all connected via a system bus. Two different component-interconnect bus schemes are available with the Z280 MPU: the Z80 Bus and the Z-BUS.

This chapter describes the external manifestations (that is, the activity on the pins) that result from CPU or on-chip peripheral activity for the Z80 Bus configurations of the Z280 MPU. (The Z-BUS external interface is described in Chapter 13.) Since the pins are connected to the system bus, most of this discussion will center on the bus and bus operations.

The condition of the OPT signal pin determines the configuration of the bus interface for the Z280 MPU; the Z80 Bus configuration is selected by applying a logical 0 (ground) level on the OPT pin.

The Z80 Bus on the Z280 MPU includes a 24-bit address bus, 8-bit data bus, and associated status and control signals. The data bus is multiplexed with the low-order 8 bits of the address bus. Figure 12-1a shows the pin functions for the Z80 Bus configuration of the Z280 MPU. The Z80 bus described here is compatible with Zilog's Z8400 and Z8500 families of peripheral devices.

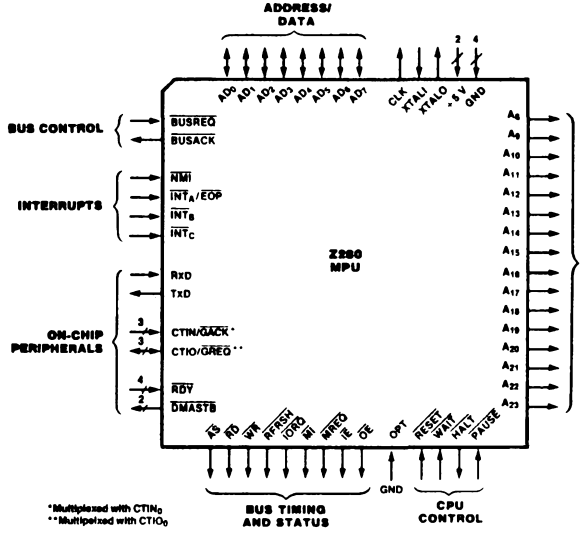


Figure 12-1a. Pin Functions

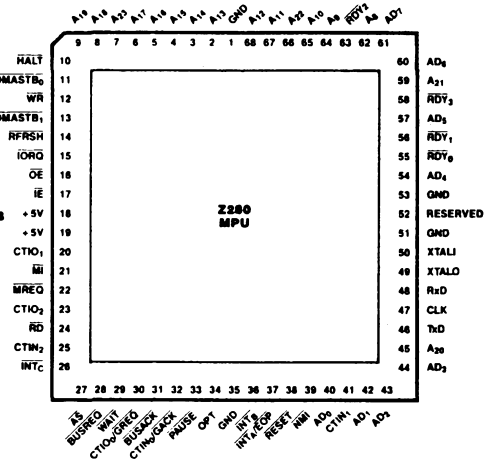


Figure 12-1b. Pin Assignments

Figure 12-1. Z80 Bus Configuration (Input OPT tied to GND)

12.2 BUS OPERATIONS

Two kinds of operations can occur on the Z80 Bus: transactions and requests. At any given time only one device (either the CPU or a bus requestor such as a DMA channel) can be in control of the bus; this device is called the bus master. Transactions are always initiated by the bus master and are responded to by some other device on the bus. Only one transaction can proceed at a time. Requests can be initiated by a device that does not have control of the bus.

Seven types of transactions can occur on the Z80 Bus, as described below:

Memory transaction. CPU- or DMA-controlled transfer of data to or from a memory location.

RETI transaction. CPU-initiated transaction used in conjunction with the interrupt logic of Z8400 family peripherals.

Halt transaction. Transaction indicating that the CPU is entering the Halt state due to the execution of a HALT instruction or a fatal sequence of traps.

Refresh. Transaction that refreshes dynamic memory; refresh transactions do not involve a transfer of data.

I/O transaction. CPU- or DMA-controlled transfer of data to or from a peripheral device.

Interrupt Acknowledge. CPU-controlled transaction used to acknowledge an interrupt and read data from the interrupting device.

DMA Flyby transaction. A DMA-controlled transaction that transfers data between a memory location and a peripheral device.

Two types of requests can occur on the Z80 Bus, as described below:

Interrupt request. A request initiated by a peripheral device to gain the attention of the CPU.

Bus request. A request by an external device (typically a DMA channel) to gain control of the bus in order to initiate transactions.

A request is answered by the CPU according to its type: for interrupt requests, an interrupt acknowledge sequence is generated; for bus requests, the CPU relinquishes the bus and activates an acknowledge signal.

12.3 PIN DESCRIPTIONS

The pin functions for the Z80 Bus configuration of the Z280 MPU are illustrated in Figure 12-1a. The pin assignments are shown in Figure 12-1b. A functional description of each pin is given below:

A_8-A_{23}. *Address* (output, active High, 3-state). These address lines carry I/O addresses and memory addresses during bus transactions.

AD_0-AD_7. *Address/Data* (bidirectional, active High, 3-state). These eight multiplexed Data and Address lines carry I/O addresses, memory addresses, and data during bus transactions.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of $\overline{AS}$ indicates the beginning of a transaction and shows that the address is valid.

$\overline{BUSACK}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates that the CPU has relinquished control of the bus in response to a bus request.

$\overline{BUSREQ}$. *Bus Request* (input, active Low). A Low on this line indicates that an external bus requester has obtained or is trying to obtain control of the bus.

CLK. *Clock Output* (output). The frequency of the processor timing clock is derived from the oscillator input (external oscillator) or crystal frequency (internal oscillator) by dividing the crystal or external oscillator input by two. The processor clock is further divided by one, two, or four (as programmed) and then output on this line.

CTIN. *Counter/Timer Input* (input, active High). These lines receive signals from external devices for the counter/timers.

CTIO. *Counter/Timer I/O* (bidirectional, active High, 3-state). These I/O lines transfer signals between the counter/timers and external devices.

$\overline{DMASTB}$. *DMA Flyby Strobe* (output, active Low). These lines select peripheral devices for flyby transfers.

$\overline{EOP}$. *End of Process* (input, active Low). An external source can terminate a DMA operation in progress by driving $\overline{EOP}$ Low. $\overline{EOP}$ always applies to the active channel; if no channel is active, $\overline{EOP}$ is ignored.

$\overline{GACK}$. *Global Acknowledge* (input, active Low). A Low on this line indicates the CPU has been granted control of a global bus.

$\overline{GREQ}$. *Global Request* (output, active Low, 3-state). A Low on this line indicates the CPU has obtained or is trying to obtain control of a global bus.

GND. *Ground.* Ground reference.

$\overline{HALT}$. *Halt* (output, active Low, 3-state). This signal indicates that the CPU is in the Halt state and is awaiting an interrupt before operation can resume.

$\overline{IE}$. *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the MPU.

$\overline{INT}$. *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt.

$\overline{IORQ}$. *Input/Output Request* (output, active Low, 3-state). This signal indicates that AD_0-AD_7 and A_{16}-A_{23} of the address bus hold a valid I/O address for an I/O read or write operation. An $\overline{IORQ}$ signal is also generated with an $\overline{M1}$ signal when an interrupt is being acknowledged, to indicate that an interrupt response vector can be placed on the data bus.

$\overline{M1}$. *Machine Cycle One* (output, active Low, 3-state). This signal indicates that the current transaction is the opcode fetch cycle of a RETI instruction execution. $\overline{M1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

$\overline{MREQ}$. *Memory Request* (output, active Low, 3-state). This signal indicates that the address bus holds a valid address for a memory read or write operation.

$\overline{NMI}$. *Nonmaskable Interrupt* (input, falling-edge activated). A High-to-Low transition on this line requests a nonmaskable interrupt.

$\overline{OE}$. *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

OPT. *Bus Option* (input). This signal establishes the bus option during reset.

OPT	Bus Interface
0	Z80 Bus, 8-bit
1	Z-BUS, 16-bit

$\overline{PAUSE}$. *MPU Pause* (input, active Low). While this line is Low the MPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

$\overline{RD}$. *Read* (output, active Low, 3-state). This signal indicates that the CPU or DMA peripheral is reading data from memory or an I/O device.

$\overline{RDY}$. *DMA Ready* (input, active Low). These lines are monitored by the DMAs to determine when a peripheral device associated with a DMA port is ready for a read or write operation. When a DMA port is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

$\overline{RESET}$. *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

$\overline{RFSH}$. *Refresh* (output, active Low, 3-state). This signal indicates that the lower ten bits of the Address bus contain a refresh address for dynamic memories and the current $\overline{MREQ}$ signal should be used to perform a refresh to all dynamic memories.

RxD. *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

TxD. *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

$\overline{WAIT}$. *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

$\overline{WR}$. *Write* (output, active Low, 3-state). This signal indicates that the bus holds valid data to be stored at the addressed memory or I/O location.

XTALI. *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip oscillator.

XTALO. *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip oscillator.

+5V. *Power Supply Voltage.* (+5 nominal).

12.4 BUS CONFIGURATION AND TIMING

Four Z280 CPU control registers specify certain characteristics of the Z280 MPU's external interface and determine bus timing: the Bus Timing and Initialization register, Bus Timing and Control register, Local Address register, and Cache Control register.

Bus timing is determined by the frequency of the Z280 MPU's external clock source or crystal and the contents of the Bus Timing and Initialization register, which receives its initial values as part of the reset process (see section 3.2.1). The frequency of the processor clock is one-half of the frequency of the external clock source or crystal. The processor clock can be further divided by a factor of 1, 2, or 4 to provide the bus timing clock, as specified by the contents of the Clock Scaling field in the Bus Timing and Initialization register. The bus timing clock is output by the MPU as the CLK signal. In the logical timing diagrams that follow, signal transitions on the bus are shown in relation to the bus clock, CLK.

The number of automatic wait states included in a given transaction is determined by the contents of the Bus Timing and Initialization and Bus Timing and Control registers. The physical memory address space is divided into two sections based on the most significant physical address bit, A_{23}. Up to three automatic wait states can be added to transactions to the lower half of memory (addresses where $A_{23} = 0$); similarly, up to three automatic wait states can be added to transactions to the upper half of memory ($A_{23} = 1$), to all I/O transactions, and to interrupt acknowledge transactions.

The state of the Multiprocessor Configuration Enable bit in the Bus Timing and Initialization register and the contents of the Local Address register determine which memory transactions require use of a global bus, as described in section 10.3. The contents of the Cache Control register and the state of the address tags and valid bits in the cache memory determine which transactions employ the cache memory and which transactions use the external bus interface, as described in Chapter 8.

12.5 TRANSACTIONS

At any given time, one device (either the CPU or a bus requester) has control of the bus and is known as the bus master. A transaction is initiated by the bus master and is responded to by some other device on the bus. Information transfers (both instructions and data) to and from the Z280 MPU are accomplished through the use of transactions. All transactions start when Address Strobe ($\overline{AS}$) is driven low and then raised high.

If the transaction requires an address, the address is valid on the rising edge of $\overline{AS}$. $\overline{AS}$ can be used to latch Z280 MPU addresses to demultiplex the Z280 Address/Data lines. If an address is generated, the Output Enable ($\overline{OE}$) line is activated coincident with $\overline{AS}$ assertion.

The Read ($\overline{RD}$) and Write ($\overline{WR}$) lines are used to time the data transfers. For transactions that do not involve the transfer of data (Refresh and Halt transactions), neither $\overline{RD}$ nor $\overline{WR}$ is activated. For write operations, a low on $\overline{WR}$ indicates that valid data from the bus master is on the AD lines. The Output Enable line continues to be asserted until $\overline{WR}$ is deasserted. For read operations, the bus master drives the $\overline{RD}$ line low when the addressed device is to put its data on the bus. Coincident with the assertion of $\overline{RD}$, the AD lines are 3 stated by the bus master and $\overline{OE}$ is deasserted; Input Enable ($\overline{IE}$) is asserted one-half clock cycle later. The bus master samples the data on the falling clock edge just before deasserting $\overline{RD}$ and $\overline{IE}$. The rising edge of $\overline{RD}$ or $\overline{WR}$ marks the end of the transaction.

The Z280 MPU's $\overline{WAIT}$ input provides a mechanism whereby the timing of a particular transaction can be extended to accommodate a memory or peripheral device with a long access time. The $\overline{WAIT}$ line is sampled on the falling clock edge when data is to be sampled (i.e. just before $\overline{RD}$ or $\overline{WR}$ rises) during a transaction. If the $\overline{WAIT}$ line is low, another bus clock cycle is added to the transaction before data is sampled ($\overline{RD}$ or $\overline{WR}$ rises). In this added cycle, and all subsequent cycles added due to $\overline{WAIT}$ being low, the $\overline{WAIT}$ line is sampled on the falling edge of the clock and, if it is low, another cycle is added to the transaction before data is sampled. In this way, the transaction can be extended by external logic to an arbitrary length, in increments of one bus clock cycle.

The $\overline{WAIT}$ input is synchronous, and must meet the specified setup and hold times in order for the Z280 MPU to function correctly. This requires asynchronously generated $\overline{WAIT}$ signals to be synchronized to the CLK output before they are input into the Z280 MPU. Automatic wait states can also be generated by programming the Bus Timing and Control register and Bus Timing and Initialization register; these are inserted in the transaction before the external $\overline{WAIT}$ signal is sampled.

12.5.1 Memory Transactions

Memory transactions move instructions or data to or from memory when a bus master makes a memory access. Thus, they are generated during program execution to fetch instructions from memory and to fetch and store memory data. They are also generated to store old program status and fetch new program status during interrupt and trap handling, and to transfer information during DMA-controlled memory accesses. A memory transaction is three bus cycles long unless extended with hardware- and/or software-generated wait states, as explained previously.

Memory transaction timing is illustrated in Figures 12-2 and 12-3. During the first bus cycle, $\overline{AS}$ is asserted to indicate the beginning of a transaction; Output Enable ($\overline{OE}$) is also asserted at this time. The $\overline{MREQ}$ signal goes active during the second half of this bus cycle, which indicates a memory transaction. For a Read operation (Figure 12-2), $\overline{RD}$ is activated during the first half of the second bus cycle, after the bus master has 3-stated the AD lines; $\overline{OE}$ is deasserted at the beginning of the second cycle and Input Enable ($\overline{IE}$) is asserted during the second half of the second cycle. The bus master samples the information returned from memory on the Address/Data bus on the falling edge of the clock during the third bus cycle; after the data is sampled, $\overline{RD}$, $\overline{MREQ}$, and $\overline{IE}$ are deasserted. For a Write operation (Figure 12-3), the $\overline{WR}$ line is asserted during the second half of the second cycle, after the bus master has placed the data to be written on the AD lines, and $\overline{OE}$ stays active throughout the transaction.

The $\overline{WAIT}$ input is also sampled on the falling edge of the clock during the third clock cycle; if $\overline{WAIT}$ is low, another bus clock cycle is added before sampling the data. Wait states can also be added through programming of the Bus Timing and Initialization register and Bus Timing and Control register. For example, Figures 12-4, 12-5, and 12-6 illustrate memory transactions with one wait state.

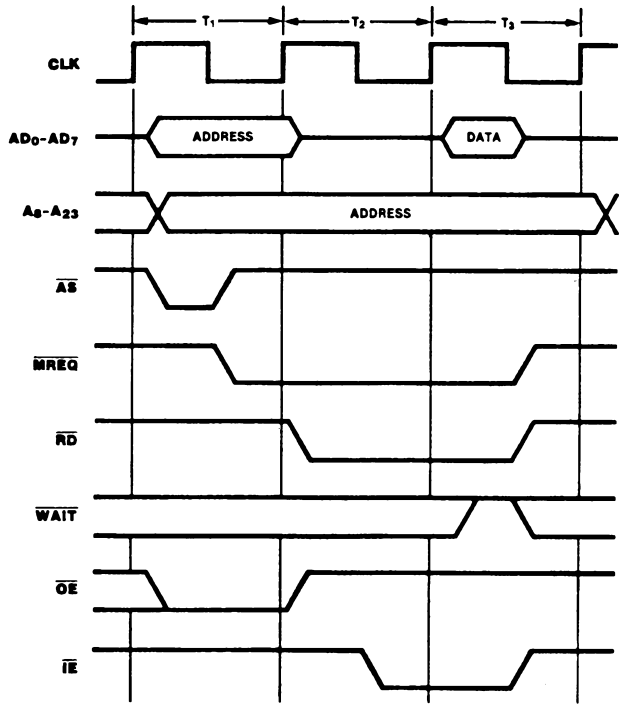


Figure 12-2. Memory Read Timing

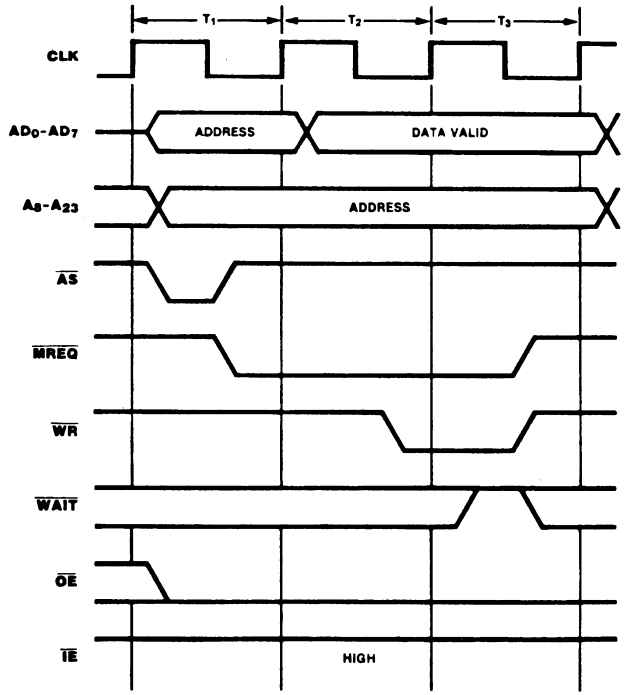


Figure 12-3. Memory Write Timing

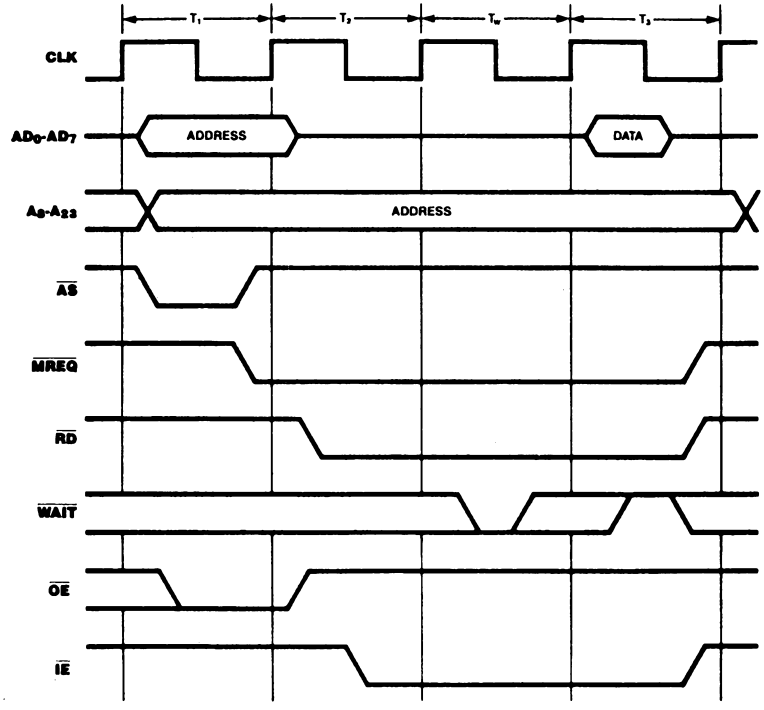


Figure 12-4. Memory Read Timing with One External Wait State

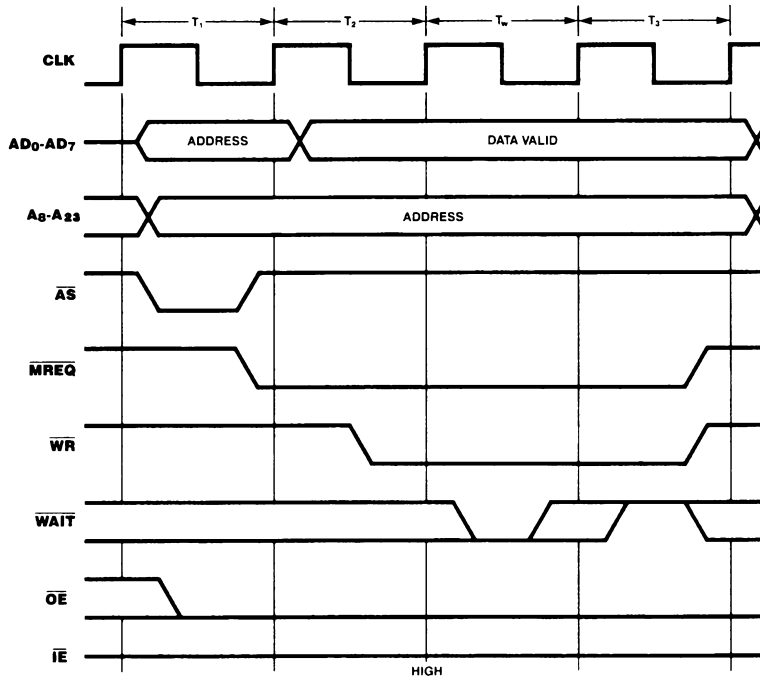


Figure 12-5. Memory Write Timing with One External Wait State

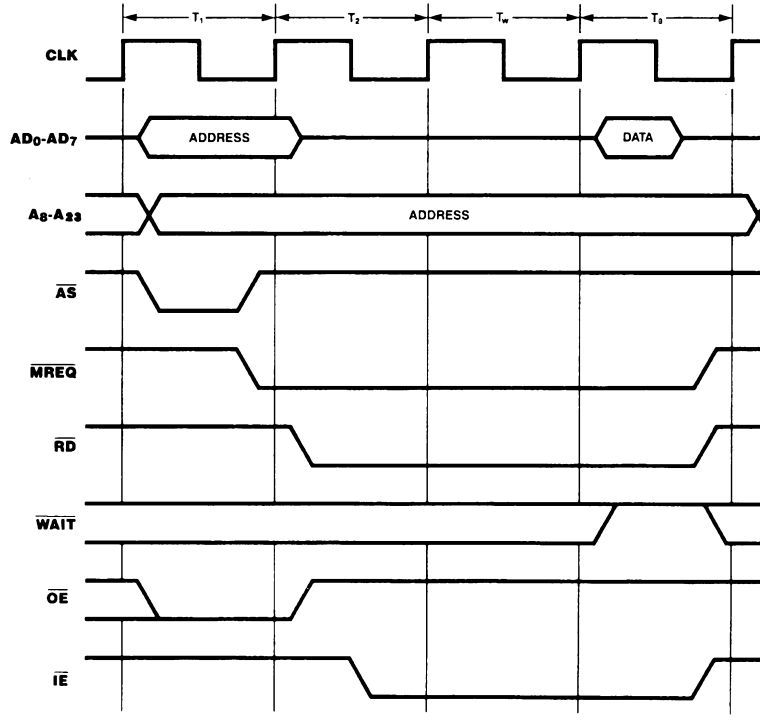


Figure 12-6. Memory Read Timing with One Internal Wait State

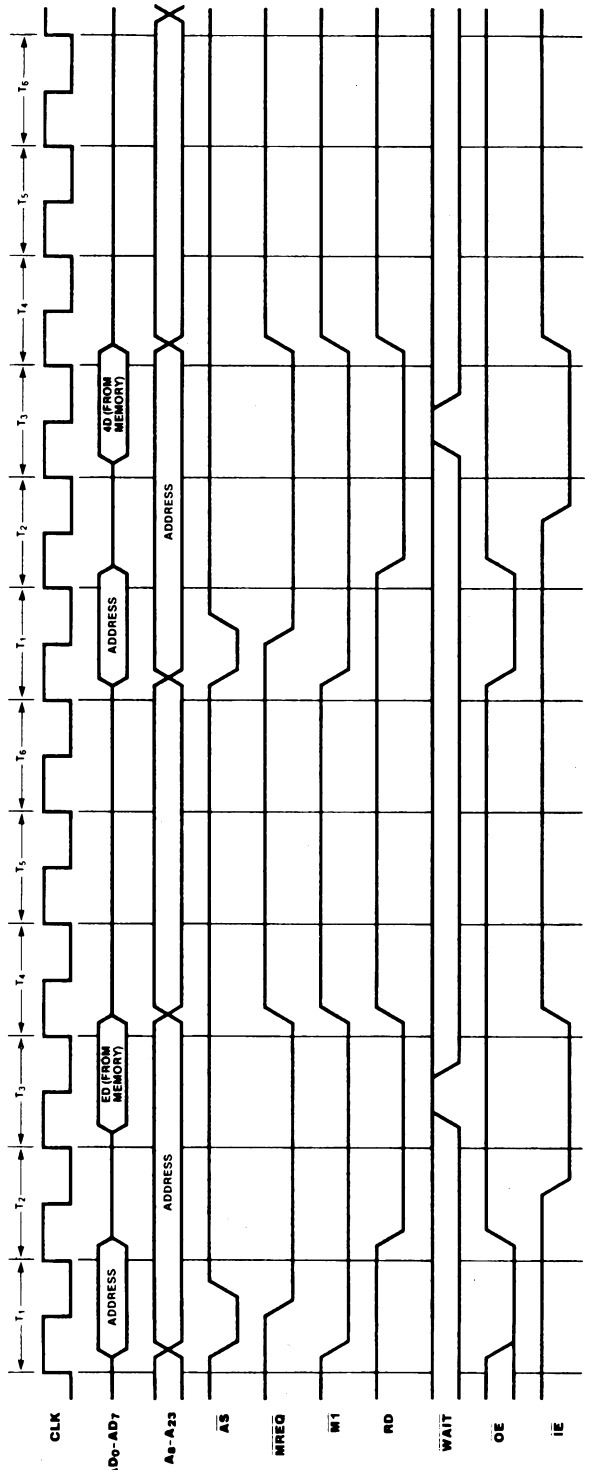


Figure 12-7. RETI Read Timing

12.5.2 RETI Transactions

RETI transactions (Figure 12-7) are similar to memory read transactions with two exceptions: $\overline{M1}$ is asserted throughout each read transaction, falling early in the first bus cycle, and $\overline{MREQ}$, $\overline{M1}$, $\overline{RD}$, and $\overline{IE}$ are deasserted on the rising edge of the clock following the third cycle. Each of the read transactions is followed by a minimum of three bus cycles of inactivity. These transactions are invoked whenever an RETI instruction is encountered in the instruction stream; they are used to re-fetch the instruction from external memory so that interrupt logic within Z8400 family peripherals that monitor the bus for this instruction will function correctly.

12.5.3 Halt and Refresh Transactions

There are two types of bus transactions that do not transfer data: Halt and Refresh transactions. These transactions are similar to memory transactions, except that $\overline{RD}$ and $\overline{WR}$ remain high, the $\overline{WAIT}$ input is not sampled, and no data is transferred.

Halt transactions (Figure 12-8) are identical to memory read transactions except that $\overline{HALT}$ is asserted throughout the transaction, falling during the second half of the first bus cycle, and remains asserted after the transaction is completed. This transaction is invoked when a $\overline{HALT}$ instruction is executed or a fatal sequence of traps occurs. For Halt transactions generated by the $\overline{HALT}$ instruction, once the Halt transaction is completed, all subsequent CPU activity is suspended until an active interrupt request or reset is detected. After Halt transactions generated due to a fatal condition, all CPU activity is suspended until an active reset is detected (see section 6.6). The $\overline{HALT}$ line remains asserted until the interrupt request is acknowledged or the reset is received. Refresh transactions or DMA transfers may occur while $\overline{HALT}$ is asserted; also, the bus can be granted. The address put out during the address phase of the Halt transaction is the address of the Halt instruction or the instruction that initiated the fatal sequence of traps.

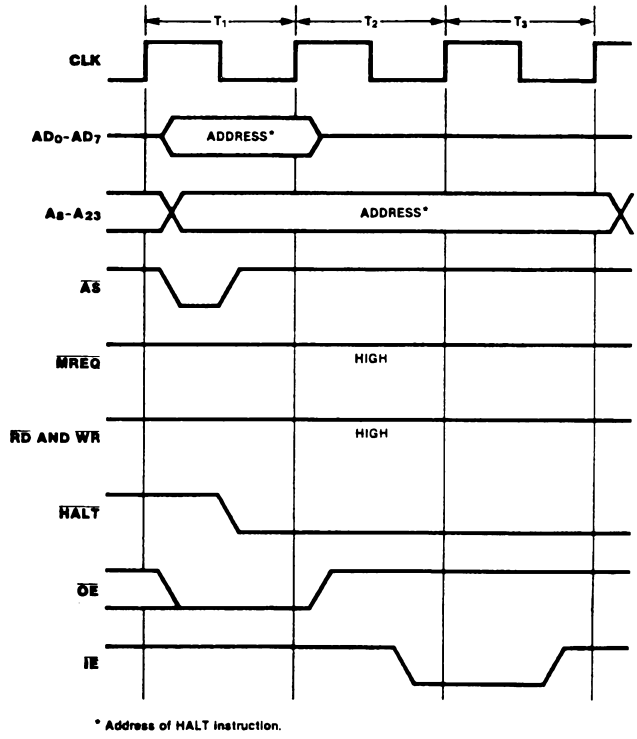


Figure 12-8. Halt Timing

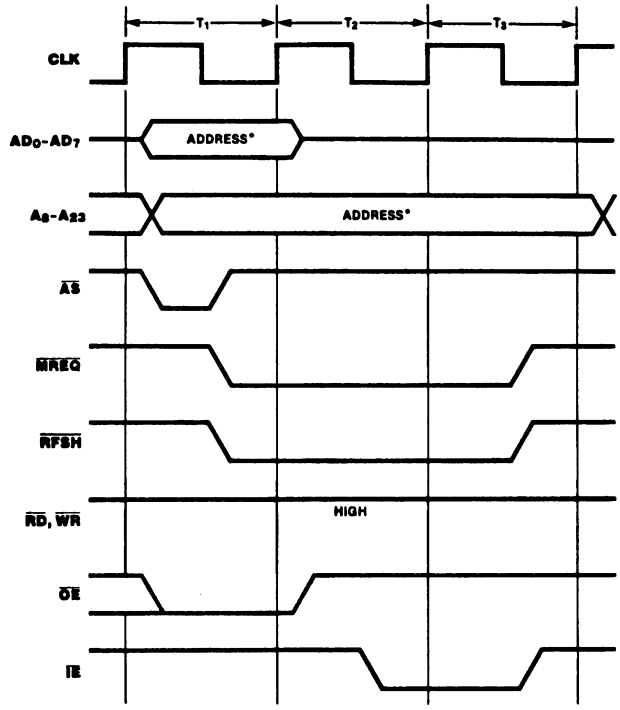
A memory refresh transaction (Figure 12-9) is generated by the Z280 MPU refresh mechanism and can occur immediately after the final clock cycle of any other transaction. The memory refresh counter's 10-bit address is output on AD_0-AD_7, A_8, and A_9 when $\overline{AS}$ is asserted; the remaining address lines are undefined. The $\overline{RFSH}$ line is activated concurrent with $\overline{MREQ}$. This transaction can be used to generate refreshes for dynamic RAMs. Refreshes may occur while the CPU is in the Halt state.

12.5.4 I/O Transactions

I/O transactions move data to or from peripherals and are generated during the execution of I/O instructions or during DMA-controlled transfers. I/O transactions to devices in I/O pages FE_H and FF_H do not generate external bus transactions.

Figures 12-10 and 12-11 illustrate I/O transaction timing. I/O transactions are four clock cycles long at a minimum, and, like memory transactions, may be lengthened by the addition of wait cycles. I/O transaction timing is similar to memory transaction timing with one automatic wait state.

The $\overline{IORQ}$ line indicates that an I/O transaction is taking place. The I/O address is found on AD_0-AD_7 and A_8-A_{23} when $\overline{AS}$ rises. For read operations, $\overline{RD}$ and $\overline{IE}$ are asserted during the second clock cycle, and input data from the peripheral is sampled by the bus master during the fourth cycle (unless additional wait states are inserted in the transaction). For write operations, $\overline{WR}$ is asserted during the second cycle with $\overline{OE}$ remaining asserted; output data to the peripheral is placed on the bus at this time.



*10 least significant bits are Refresh address, the rest are undefined.

Figure 12-9. Memory Refresh Timing

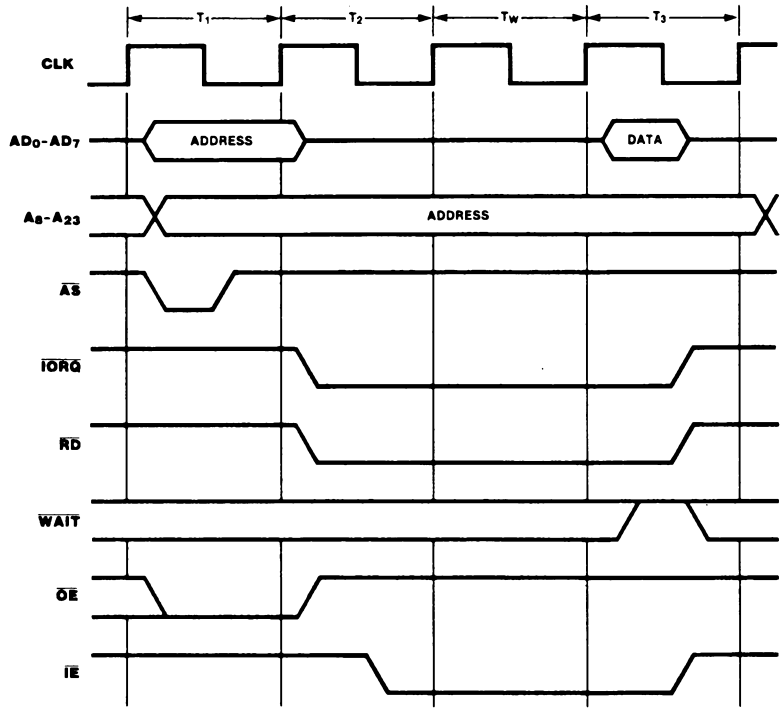


Figure 12-10. I/O Read Timing

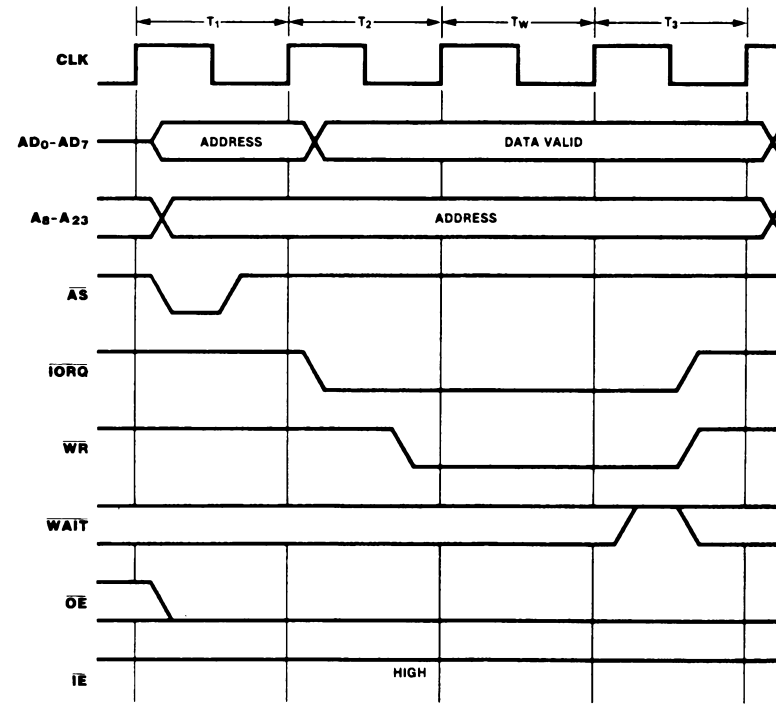


Figure 12-11. I/O Write Timing

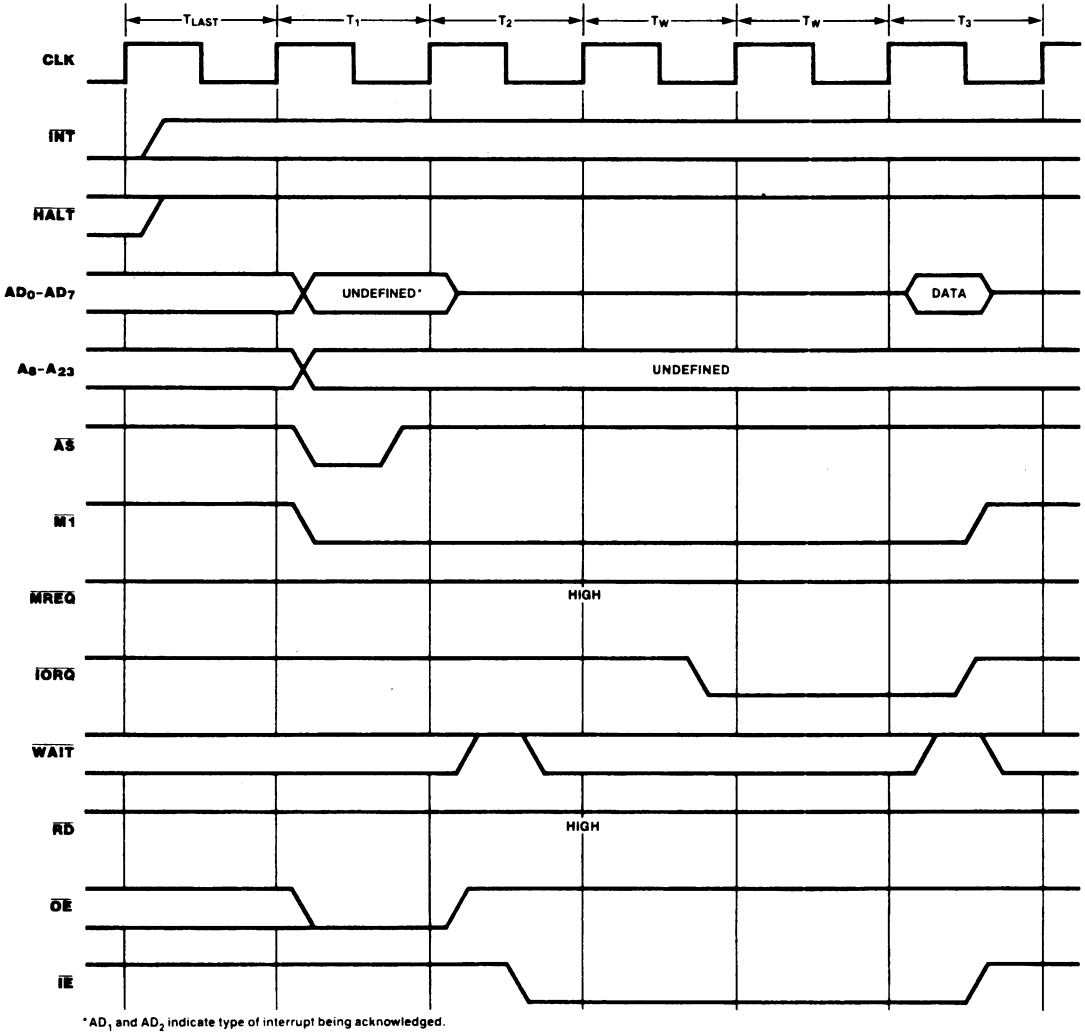
12.5.5 Interrupt Acknowledge Transactions

Interrupt acknowledge transactions acknowledge an interrupt and read information from the device that generated the interrupt. These transactions are generated automatically by the CPU when an interrupt request is detected.

Interrupt acknowledge transactions are five cycles long at a minimum, with two automatic wait cycles (Figure 12-12). The wait cycles are used to give the interrupt priority daisy chain (or other priority resolution devices) time to settle before the identifier or vector is read. Additional automatic wait states can be generated by programming the Bus Timing and Control register.

The interrupt acknowledge transaction is indicated by an $\overline{M1}$ assertion without $\overline{MREQ}$ during the first cycle. The AD_1 and AD_2 address lines indicate the type of interrupt being acknowledged when $\overline{AS}$ is asserted (see Table 6-4); the remaining address lines are undefined. The $\overline{IORQ}$ signal becomes active during the third cycle to indicate that the interrupting device can place an 8-bit identifier or vector on the bus. It is captured from the AD lines on the falling clock edge before $\overline{IORQ}$ is raised high.

There are two places where the $\overline{WAIT}$ line is sampled and, thus, where wait states can be inserted by external circuitry. The first, during T2, serves to delay the falling edge of $\overline{IORQ}$ to



*AD_1 and AD_2 indicate type of interrupt being acknowledged.

Figure 12-12. Interrupt Acknowledge Sequence

allow the daisy chain a longer time to settle; the second, during T3, serves to delay the point at which the identifier or vector is read. Software-generated wait states can also be added at either time via programming of the DC and I/O fields in the Bus Timing and Control register. As always, software-generated wait states are inserted into the transaction before the external $\overline{WAIT}$ signal is sampled.

12.5.6 DMA Flyby Transactions

On-chip DMA channels 0 and 1 can transfer data between memory and peripheral devices using flyby type transfers; external DMA controllers in Z280 MPU systems may also have this capability. The timing of flyby transactions is identical to memory transaction timing, with the exception that the DMA Flyby Strobe ($\overline{DMASTB}$) signal is activated; the $\overline{DMASTB}$ signal is used to select the participating I/O device that must capture or supply the data during the memory access transaction.

Flyby transactions controlled by the on-chip DMA channels always include one automatic wait state (Figures 12-13 and 12-14). As with all memory transactions, other hardware- and software-generated wait states can be added to the transaction. The external $\overline{WAIT}$ signal is sampled at two different times: during the automatic wait state and during T3.

For flyby transactions that read from memory and write to a peripheral (Figure 12-13), $\overline{DMASTB}$ is asserted during the automatic wait state and any subsequent wait states added by an active $\overline{WAIT}$ signal sampled during the automatic wait state. Thus, if the $\overline{WAIT}$ input is asserted during the automatic wait state, the additional wait states extend the width of the $\overline{DMASTB}$ pulse. Wait states added via the assertion of $\overline{WAIT}$ during T3 (after $\overline{DMASTB}$ is deasserted) stretch the $\overline{RD}$ signal without affecting $\overline{DMASTB}$.

For flyby transactions that read from a peripheral and write to memory (Figure 12-14), $\overline{DMASTB}$ is asserted at the beginning of T2 and remains asserted until the second half of T3. The $\overline{WR}$ signal is asserted only during the automatic wait state and any subsequent wait states added by sampling $\overline{WAIT}$ during the automatic wait state. Wait states added via the assertion of $\overline{WAIT}$ during T3 stretch the $\overline{DMASTB}$ signal without affecting $\overline{WR}$.

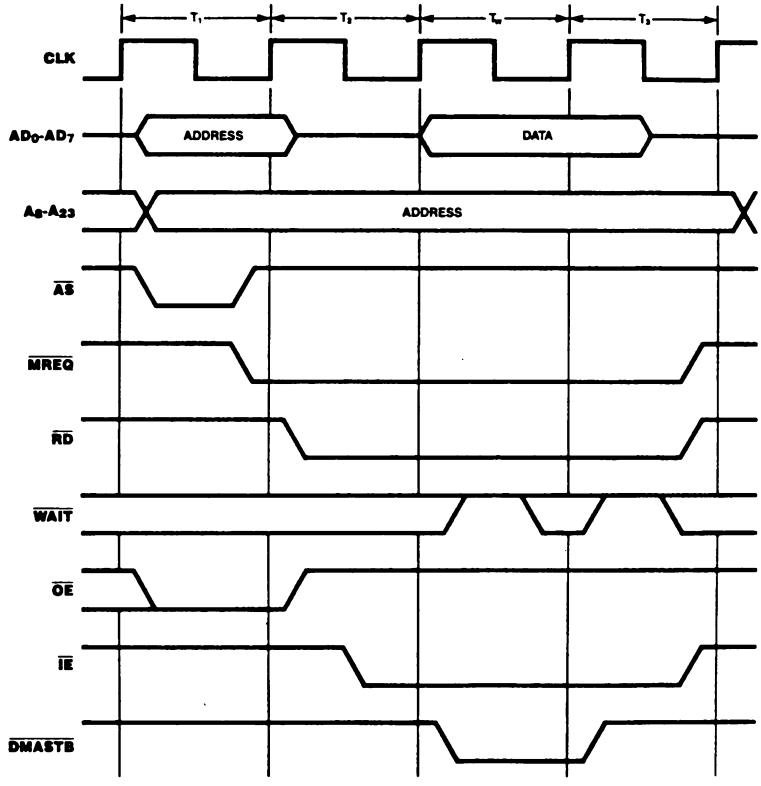


Figure 12-13. On-Chip DMA Channel Flyby Memory Read Transaction

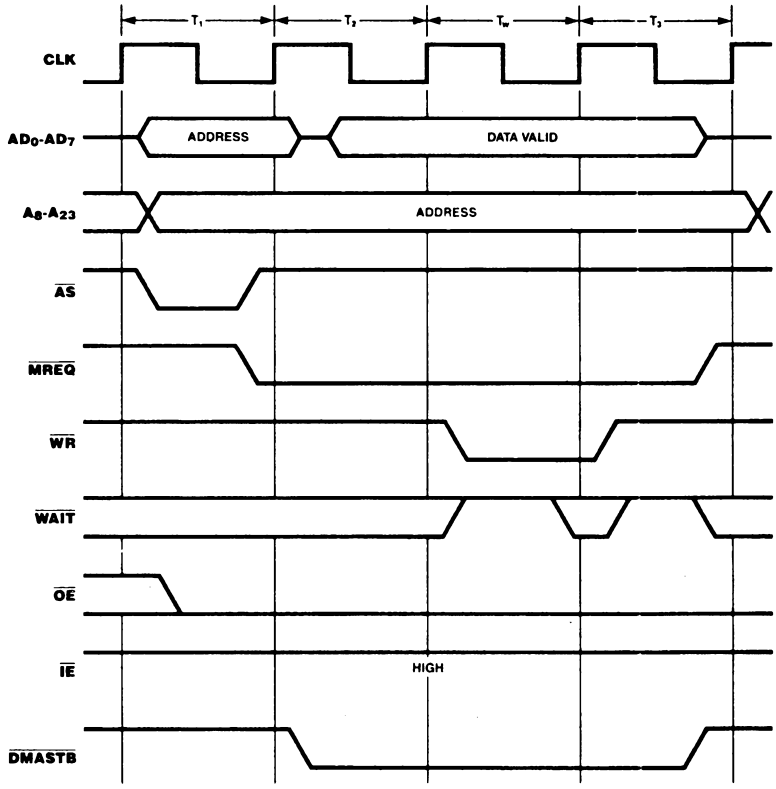


Figure 12-14. On-Chip DMA Channel Flyby Memory Write Transaction

12.6 REQUESTS

The Z280 MPU supports three types of request signals: interrupt requests, local bus requests, and global bus requests. A request is answered according to its type. Interrupt requests are generated by peripheral devices; the Z280 MPU responds with an Interrupt Acknowledge transaction. Local bus requests are initiated by an external potential bus master; the Z280 MPU responds by relinquishing the bus and generating an active Bus Acknowledge signal. Global bus requests are generated by the Z280 CPU or an on-chip DMA channel to access a global bus; the Z280 MPU receives a Global Bus Acknowledge signal in response to the request.

12.6.1 Interrupt Requests

The Z280 CPU supports two types of interrupts, maskable $\overline{INT}$ and nonmaskable ($\overline{NMI}$). The interrupt request line from a device capable of generating interrupts can be tied to the Z280 MPU's $\overline{INT}$ or $\overline{NMI}$ inputs; several devices can be connected to one interrupt request input, with interrupt priorities established via external logic or a priority daisy chain. However, all Z8400 family peripherals in a Z280-based system will respond to the RETI transaction. Therefore, either all Z8400 family peripherals should use the same interrupt request line or, alternatively, no nesting of interrupts should be allowed among the Z8400 peripherals using different interrupt request lines.

Nonmaskable interrupt requests are edge-triggered, but maskable interrupts are level-triggered. Any high-to-low transition on the $\overline{NMI}$ input is asynchronously edge-detected, and an internal $\overline{NMI}$ latch is set. At the beginning of the last clock cycle during execution of an instruction, the maskable interrupt inputs are sampled along with the state of the internal $\overline{NMI}$ latch. If an interrupt is detected, and that interrupt is enabled in the Master Status register, interrupt processing proceeds in accordance with the current interrupt mode, as described in Chapter 6.

12.6.2 Local Bus Requests

To generate transactions on the bus, a potential bus master (such as a DMA controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling $\overline{BUSREQ}$ low; the Z280 MPU responds by 3-stating its address, data, bus control, and bus status outputs and asserting an active $\overline{BUSACK}$, as described in section 10.2. The CPU regains control of the bus after $\overline{BUSREQ}$ rises. The on-chip DMA channels have higher priority than external devices requesting the bus via $\overline{BUSREQ}$.

12.6.3 Global Bus Requests

If the multiprocessor mode is specified in the Bus Timing and Initialization register, then the contents of the Local Address register determine the range of memory addresses dedicated to the shared global bus. Before accessing an address on the global bus, the Z280 MPU must issue a Global Bus Request ($\overline{GREQ}$) and receive an active Global Bus Acknowledge ($\overline{GACK}$) signal, as described in Section 10.3.

Figure 12-15 illustrates the timing of the global bus request/acknowledge sequence. When the Z280 MPU needs to access a location on the global bus, $\overline{GREQ}$ is asserted in order to request use of the global bus. $\overline{GACK}$ is then sampled on each successive rising edge of the clock; when $\overline{GACK}$ becomes active (and if $\overline{BUSREQ}$ is not asserted), the memory transaction proceeds as described in section 12.5.1. $\overline{GREQ}$ is deasserted in the bus clock cycle immediately following the end of the memory transaction (except when executing the Test and Set instruction, where both the memory read and write operations are executed before deasserting $\overline{GREQ}$).

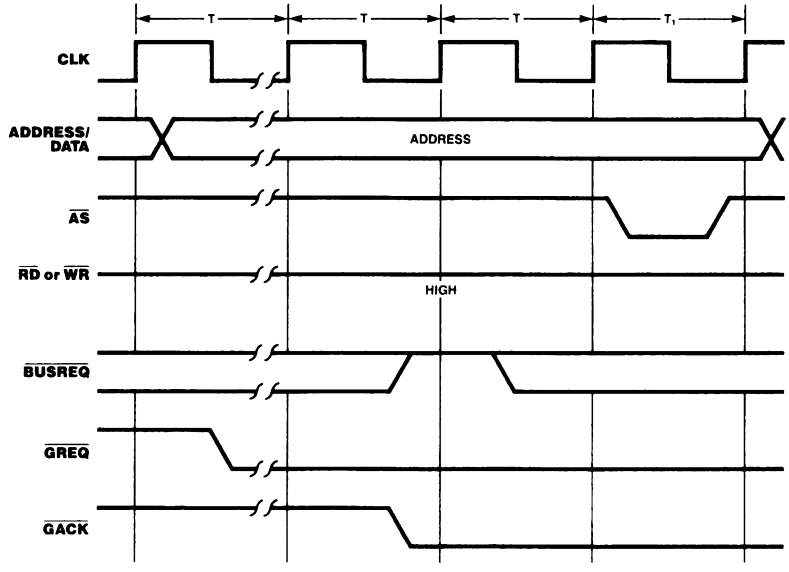


Figure 12-15. Multiprocessor Mode Timing

Chapter 13.
Z-BUS External Interface

13.1 INTRODUCTION

The Z280 MPU is typically only one component in a system that may include memory, peripherals, slave processors, coprocessors, and other CPUs, all connected via a system bus. Two different component-interconnect bus schemes are available with the Z280 MPU: the Z80 Bus and the Z-BUS.

This chapter describes the external manifestations (that is, the activity on the pins) that result from CPU or on-chip peripheral activity for the Z-BUS configurations of the Z280 MPU. (The Z80 Bus external interface is described in Chapter 12.) Since the pins are connected to the system bus, most of this discussion will center on the bus and bus operations.

The condition of the OPT pin determines the configuration of the bus interface for the Z280 MPU; the Z-BUS configuration is selected either by applying a logical 1 (V_{CC}) level on the OPT pin or by leaving the OPT pin disconnected.

The Z-BUS on the Z280 MPU includes a 24-bit address bus, 16-bit data bus, and associated status and control signals. The data bus is multiplexed with the low-order 16 bits of the address bus. The Z-BUS configuration of the Z280 MPU supports the use of Extended Processing Units and burst-mode memories. Figure 13-1 shows the pin functions and pin assignments for the Z-BUS configuration of the Z280 MPU. The Z-BUS described here is compatible with Zilog's Z8000 family of peripheral devices. Other Z-BUS compatible components include the Z8000 family of CPUs. Refer to Zilog's Component Data Book for a complete description of the Z-BUS Component Interconnect convention.

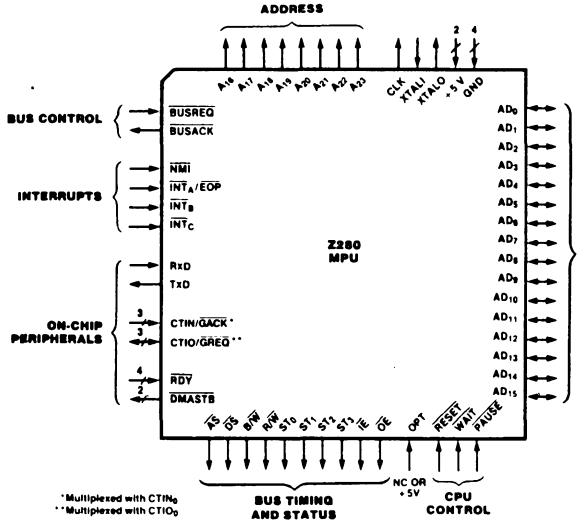


Figure 13-1a. Pin Functions

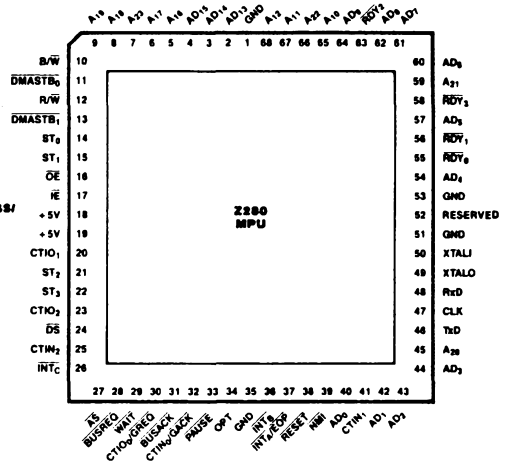


Figure 13-1b. Pin Assignments

Figure 13-1. Z-BUS Configuration (Input OPT tied to + 5V or not connected)

13.2 BUS OPERATIONS

Two kinds of operations can occur on the Z-BUS: transactions and requests. At any given time only one device (either the CPU or a bus requestor such as a DMA channel) can be in control of the bus; this device is called the bus master. Transactions are always initiated by the bus master and are responded to by some other device on the bus. Only one transaction can proceed at a time. Requests can be initiated by a device that does not have control of the bus.

Seven types of transactions can occur on the Z-BUS, as described below:

Memory transaction. CPU- or DMA-controlled transfer of data to or from a memory location.

Halt transaction. Transaction indicating that the CPU is entering the Halt state due to execution of a HALT instruction or a fatal sequence of traps.

Refresh. Transaction that refreshes dynamic memory; refresh transactions do not involve a transfer of data.

I/O transaction. CPU- or DMA-controlled transfer of data to or from a peripheral device.

Interrupt Acknowledge. CPU-controlled transaction used to acknowledge an interrupt and read data from the interrupting device.

EPU transaction. A transfer of data from an Extended Processing Unit (EPU) to the CPU.

DMA Flyby transaction. A DMA-controlled transaction that transfers data between a memory location and a peripheral device.

Two types of requests can occur on the Z-BUS, as described below:

Interrupt request. A request initiated by a peripheral device to gain the attention of the CPU.

Bus request. A request by an external device (typically a DMA channel) to gain control of the bus in order to initiate transactions.

A request is answered by the CPU according to its type: for interrupt requests, an interrupt acknowledge sequence is generated; for bus requests, the CPU relinquishes the bus and activates an acknowledge signal.

13.3 PIN DESCRIPTIONS

The pin functions and assignments for the Z-BUS configuration of the Z280 MPU are illustrated in Figure 13-1. A functional description of each pin is given below:

A_{16}-A_{23}. *Address* (output, active High, 3-state). These address lines carry I/O addresses and memory addresses during bus transactions.

AD_0-AD_{15}. *Address/Data* (bidirectional, active High, 3-state). These 16 multiplexed address and data lines carry I/O addresses, memory addresses, and data during bus transactions.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of Address Strobe indicates the beginning of a transaction and shows that the address, status, R/$\overline{W}$, and B/$\overline{W}$ signals are valid.

$\overline{BUSACK}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates that the CPU has relinquished control of the bus in response to a bus request.

$\overline{BUSREQ}$. *Bus Request* (input, active Low). A Low on this line indicates that an external bus requester has obtained or is trying to obtain control of the bus.

B/$\overline{W}$. *Byte/Word* (output, Low = Word, 3-state). This signal indicates whether a byte or a word of data is to be transmitted during a transaction.

CLK. *Clock Output* (output). The frequency of the processor timing clock is derived from the oscillator input (external oscillator) or crystal frequency (internal oscillator) by dividing the crystal or external oscillator input by two. The processor clock is further divided by one, two, or four (as programmed), and then output on this line.

CTIN. *Counter/Timer Input* (input, active High). These lines receive signals from external devices for the counter/timers.

CTIO. *Counter/Timer I/O* (bidirectional, active High, 3-state). These I/O lines transfer signals between the counter/timers and external devices.

$\overline{DMASTB}$. *DMA Flyby Strobe* (output, active Low). These lines select peripheral devices for DMA flyby transfers.

$\overline{DS}$. *Data Strobe* (output, active Low, 3-state). This signal provides timing for data movement to or from the bus master.

$\overline{EOP}$. *End of Process* (input, active Low). An external source can terminate a DMA operation in progress by driving $\overline{EOP}$ Low. $\overline{EOP}$ always applies to the active channel; if no channel is active, $\overline{EOP}$ is ignored.

$\overline{GACK}$. *Global Acknowledge* (input, active Low). A Low on this line indicates the CPU has been granted control of a global bus.

$\overline{\text{GREQ}}$. *Global Request* (output, active Low, 3-state). A Low on this line indicates the CPU has obtained or is trying to obtain control of a global bus.

$\overline{\text{IE}}$. *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the CPU.

$\overline{\text{INT}}$. *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt.

$\overline{\text{NMI}}$. *Nonmaskable Interrupt* (input, falling-edge activated). A High-to Low transition on this line requests a nonmaskable interrupt.

$\overline{\text{OE}}$. *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

OPT. *Bus Option* (input). This signal establishes the bus option during reset as follows:

OPT	Bus Interface
0	Z80-Bus, 8-bit
1	Z-BUS, 16-bit

$\overline{\text{PAUSE}}$. *CPU Pause* (input, active Low). While this line is Low the CPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

$\overline{\text{RDY}}$. *DMA Ready* (input, active Low). These lines are monitored by the DMA channels to determine when a peripheral device associated with a DMA channel is ready for a read or write operation. When a DMA channel is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

$\overline{\text{RESET}}$. *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

R/$\overline{\text{W}}$. *Read/Write* (output, Low = Write, 3-state). This signal determines the direction of data transfer for memory, I/O, or EPU transfer transactions.

RxD. *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

ST_0-ST_3. *Status* (output, active High, 3-state). These four lines indicate the type of transaction occurring on the bus and give additional information about the transaction.

TxD. *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

$\overline{\text{WAIT}}$. *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

XTALI. *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip clock oscillator.

XTALO. *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip clock oscillator.

+5V. *Power Supply Voltage.* (+5 nominal).

GND. *Ground.* Ground reference.

13.4 BUS CONFIGURATION AND TIMING

Four Z280 CPU control registers specify certain characteristics of the Z280 MPU's external interface and determine bus timing: the Bus Timing and Initialization register, Bus Timing and Control register, Local Address register, and Cache Control register.

Bus timing is determined by the frequency of the Z280 MPU's external clock source or crystal and the contents of the Bus Timing and Initialization register, which receives its initial values as part of the reset process (see section 3.2.1).

The frequency of the processor clock is one-half of the frequency of the external clock source or crystal. The processor clock can be further divided by a factor of 1, 2, or 4 to provide the bus timing clock, as specified by the contents of the Clock Scaling field in the Bus Timing and Initialization register. The bus timing clock is output by the MPU as the CLK signal. In the logical timing diagrams that follow, signal transitions on the bus are shown in relation to the bus clock, CLK.

The number of automatic wait states included in a given transaction is determined by the contents of the Bus Timing and Initialization and Bus Timing and Control registers. The physical memory address space is divided into two sections based on the most significant physical address bit, A_{23}. Up to three automatic wait states can be added to transactions to the lower half of memory (addresses where $A_{23} = 0$); similarly, up to three automatic wait states can be added to transactions to the upper half of memory ($A_{23} = 1$), to all I/O transactions, and to interrupt acknowledge transactions.

The state of the Multiprocessor Configuration Enable bit in the Bus Timing and Initialization register and the contents of the Local Address register determine which memory transactions require use of a global bus, as described in section 10.3. The contents of the Cache Control register and the state of the address tags and valid bits in the cache memory determine which transactions employ the cache memory and which transactions use the external bus interface, as described in Chapter 8.

13.5 TRANSACTIONS

At any given time, one device (either the CPU or a bus requester) has control of the bus and is known as the bus master. A transaction is initiated by the bus master and is responded to by some other device on the bus. Information transfers (both instructions and data) to and from the Z280 MPU are accomplished through the use of transactions. All transactions start when Address Strobe ($\overline{AS}$) is driven low and then raised high.

On the rising edge of $\overline{AS}$, the bus status signals (ST_0-ST_3, R/$\overline{W}$, and B/$\overline{W}$) are valid. The ST_0-ST_3 status lines indicate the type of transaction being performed (Table 13-1). Typically, these signals are decoded and used to enable the appropriate buffers, drivers, and chip select logic necessary for proper completion of the data transfer.

Table 13-1. ST Status Line Decode

Status Lines 3••0	Type of Transaction
0000	Reserved
0001	Refresh
0010	I/O transaction
0011	Halt
0100	Interrupt acknowledge line A
0101	$\overline{NMI}$ acknowledge
0110	Interrupt acknowledge line B
0111	Interrupt acknowledge line C
1000	Transfer between CPU and memory, cacheable
1001	Transfer between CPU and memory, non-cacheable
1010	Data transfer between EPU and memory
1011	Reserved
1100	EPU Instruction fetch, template, subsequent words
1101	EPU Instruction fetch, template, first word
1110	Data transfer between EPU and CPU
1111	Test and Set (data transfers)

If the transaction requires an address, the address is valid on the rising edge of $\overline{AS}$. Thus, $\overline{AS}$ can be used to latch Z280 MPU addresses to de-multiplex the Address/Data lines. No address is required for EPU-CPU or Interrupt Acknowledge transactions; the contents of the A and AD lines are undefined while $\overline{AS}$ is asserted during these transactions. If an address is generated for a transaction, the Output Enable ($\overline{OE}$) signal is activated coincident with $\overline{AS}$ assertion.

The Z-BUS MPUs use Data Strobe ($\overline{DS}$) to time the transfer of data. For transactions that do not involve the transfer of data (Refresh and Halt transactions), $\overline{DS}$ is not activated. During write operations (R/$\overline{W}$ = low), a low on $\overline{DS}$ indicates that valid data from the bus master is on the Address/Data lines. The Output Enable line continues to be asserted until $\overline{DS}$ is deasserted. For Read Operations (R/$\overline{W}$ = high), the bus master drives $\overline{DS}$ low when the addressed device is to put its data on the bus. Coincident with the assertion of $\overline{DS}$ during a read operation, the AD lines are 3-stated by the bus master, $\overline{OE}$ is deasserted, and Input Enable ($\overline{IE}$) is asserted. The bus master samples the data on the falling clock edge just before deasserting $\overline{DS}$ and $\overline{IE}$.

The Z280 MPU's $\overline{WAIT}$ input provides a mechanism whereby the timing of a particular transaction can be extended to accommodate a memory or peripheral device with a long access time. The $\overline{WAIT}$ line is sampled on the falling clock edge when data is to be sampled (i.e. just before $\overline{DS}$ rises) during a transaction. If the $\overline{WAIT}$ line is low, another bus clock cycle is added to the transaction before data is sampled and $\overline{DS}$ rises. In this added cycle, and all subsequent cycles added due to $\overline{WAIT}$ being low, the $\overline{WAIT}$ line is sampled on the falling edge of the clock and, if it is low, another cycle is added to the transaction. In this way, the transaction can be extended by external logic to an arbitrary length, in increments of one bus clock cycle.

The $\overline{WAIT}$ input is synchronous, and must meet the specified setup and hold times in order for the Z280 MPU to function correctly. This requires asynchronously-generated $\overline{WAIT}$ signals to be synchronized to the CLK output before they are input into the Z280 MPU. Automatic wait states can also be generated by programming the Bus Timing and Control register and Bus Timing and Initialization register; these are inserted in the transaction before the external $\overline{WAIT}$ signal is sampled.

13.5.1 Memory Transactions

Memory transactions move instructions or data to or from memory when a bus master makes a memory access. Thus, they are generated during program execution to fetch instructions from memory and to fetch and store memory data. They are also generated to store old program status and fetch new program status during interrupt and trap handling, and to transfer information during DMA-controlled memory accesses. A memory transaction is three bus cycles long unless extended with hardware- and/or software-generated wait states, as explained previously.

During memory transactions, the ST_3-ST_0 status lines indicate that a memory transaction is occurring and provide the following information:

- Whether the memory access is cacheable (ST_3-ST_0 = 1000) or noncacheable (ST_3-ST_0 = 1001).
- Whether the memory access is a fetch of an extended instruction's template intended for an EPU (ST_3-ST_0 = 1100 or 1101).
- Whether the data is supplied or captured by an Extended Processor Unit while executing an extended instruction (ST_3-ST_0 = 1010).
- Whether the memory access is part of an atomic read-modify-write operation during the execution of a Test and Set instruction (ST_3-ST_0 = 1111).

A memory read is distinguished from a memory write via the R/$\overline{W}$ signal.

13.5.1.1 Byte/Word Organization

The byte is the basic addressable memory element in Z280 MPU systems. However, although memory is addressed as bytes, the Z-BUS configuration of the Z280 MPU has a 16-bit data path, and memory transactions can be byte or word transfers. Each 16-bit word in memory is made up of two 8-bit bytes, where the least-significant byte preceeds the most-significant byte of the word, as in the Z80 CPU architecture. For example, the word at memory location 5000_H has its low-order byte at location 5000_H and its high-order byte at location 5001_H.

Bytes transferred to or from odd memory locations (address bit 0 = 1) are always transmitted on lines AD_0-AD_7. Bytes transferred to or from even memory locations (address bit 0 = 0) are always transmitted on lines AD_8-AD_{15}. For byte reads (B/$\overline{W}$ = high, R/$\overline{W}$ = high), the CPU or on-chip DMA channel uses only the byte whose address it put out on the bus. In other words, for a byte read with an odd address, the CPU or DMA channel will only read the lower half of the bus; for a byte read with an even address, the CPU or DMA channel will only read the upper half of the bus. For byte writes (B/$\overline{W}$ = high, R/$\overline{W}$ = low), the CPU or on-chip DMA channel (flowthrough mode) places the byte to be written on both halves of the bus, and the proper byte must be selected in the memory control logic by testing address bit 0.

For word transfers (B/$\overline{W}$ = low), all 16 bits are captured by the CPU or DMA channnel during reads (R/$\overline{W}$ = high) or stored by the memory during writes

($R/\overline{W}$ = low). The most-significant byte of the word is transferred on AD_0-AD_7 and the least-significant byte on AD_8-AD_{15}; thus, the bytes of data will appear swapped on the bus, with the most significant byte on the lower half of the bus and the least significant byte on the upper half of the bus. Word transfers always use even-valued addresses (address bit 0 = 0) and result in an access to the byte at the even address and the next consecutive byte at the following odd address. For example, a word access to location 5000_H would access the byte at location 5000H (transferred on AD_8-AD_{15}) and the byte at location 5001_H (transferred on AD_0-AD_7).

Instruction fetches are always executed as word transactions. However, instruction opcodes need not be aligned on even-address boundaries; the CPU will use only one byte of the fetched word if appropriate.

Data accesses may be byte or word accesses. Data words aligned at even-address memory boundaries are accessed via one word transaction. Data words on odd-address boundaries are accessed via two consecutive byte transactions.

13.5.1.2 Memory Transaction Timing

Memory transaction timing is illustrated in Figures 13-2 and 13-3. During the first bus cycle, $\overline{AS}$ is asserted to indicate the beginning of a transaction; Output Enable ($\overline{OE}$) is also asserted at this time. All address and status information is guaranteed valid on the rising edge of $\overline{AS}$. The S_0-S_3 status lines indicate that a memory transaction is occurring. For a read operation (Figure 13-2), $\overline{DS}$ is activated during the first half of the second bus cycle, after the bus master has 3-stated the AD lines; $\overline{OE}$ is deasserted at the beginning of the second cycle and Input Enable ($\overline{IE}$) is asserted during the second half of the second cycle. The bus master samples the information returned from memory on the Address/Data bus on the falling edge of the clock during the third bus cycle; after the data is sampled, $\overline{DS}$ and $\overline{IE}$ are deasserted. For a write operation (Figure 13-3), $\overline{DS}$ is asserted during the second half of the second cycle, after the bus master has placed the data to be written on the AD lines, and $\overline{OE}$ stays active throughout the transaction.

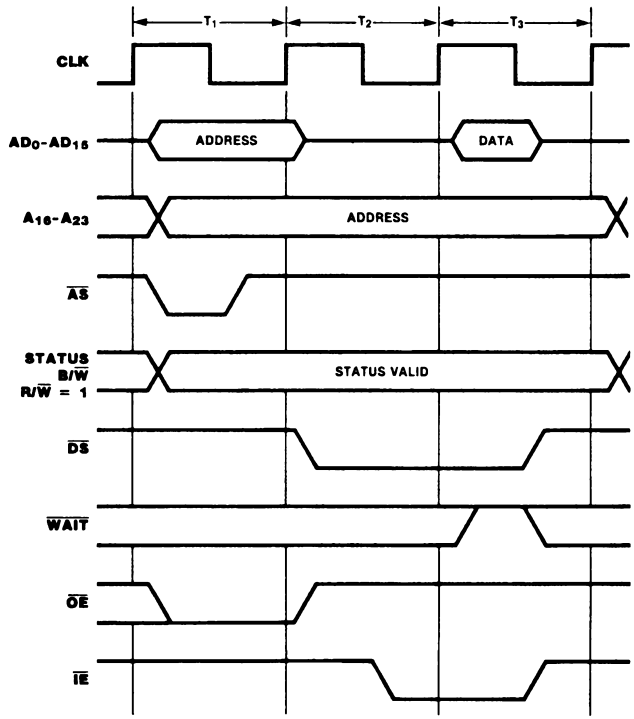


Figure 13-2. Memory Read Timing

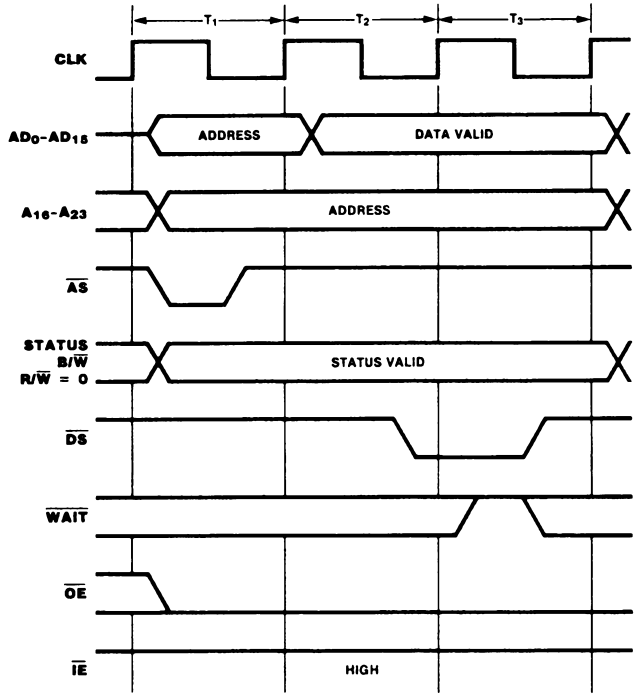


Figure 13-3. Memory Write Timing

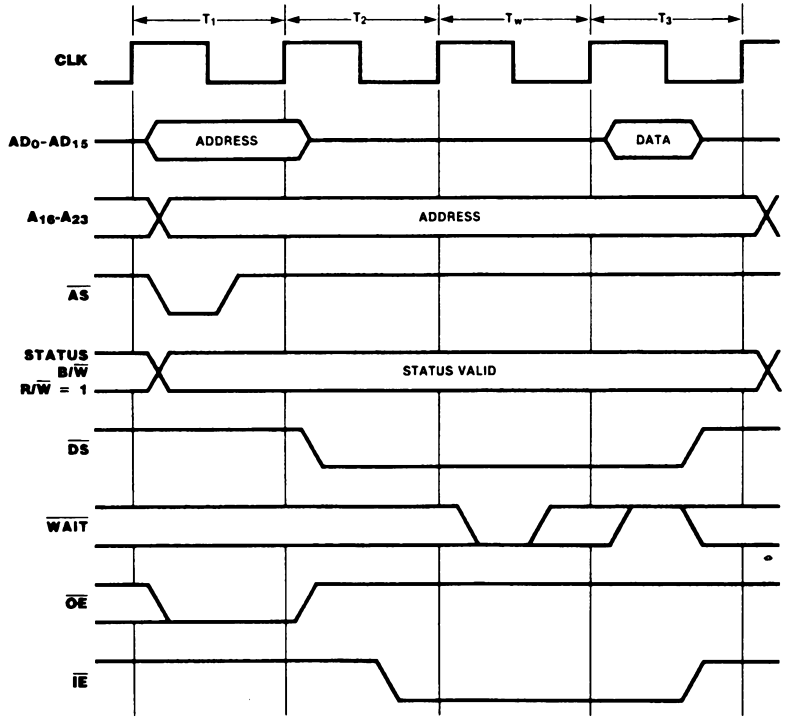


Figure 13-4. Memory Read Timing with External Wait Cycle

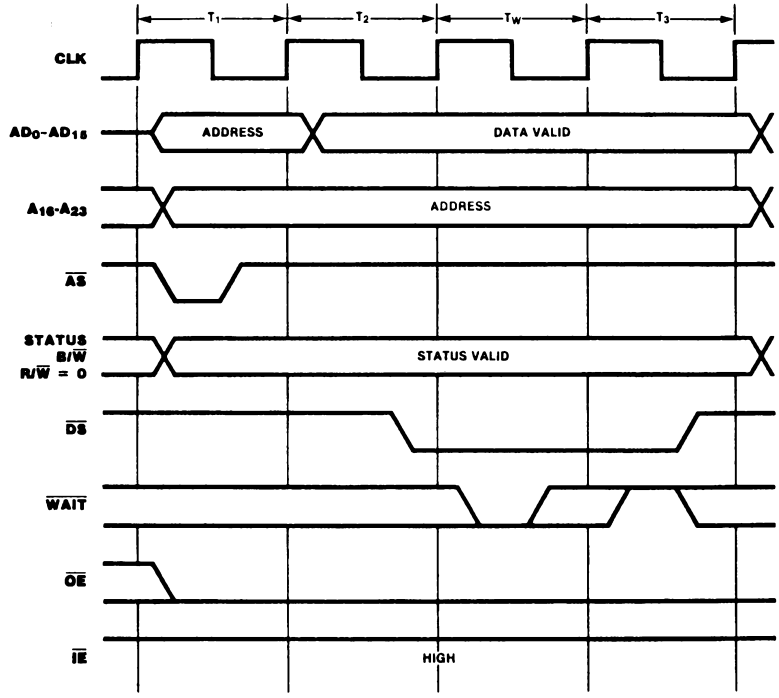


Figure 13-5. Memory Write Timing with External Wait Cycle

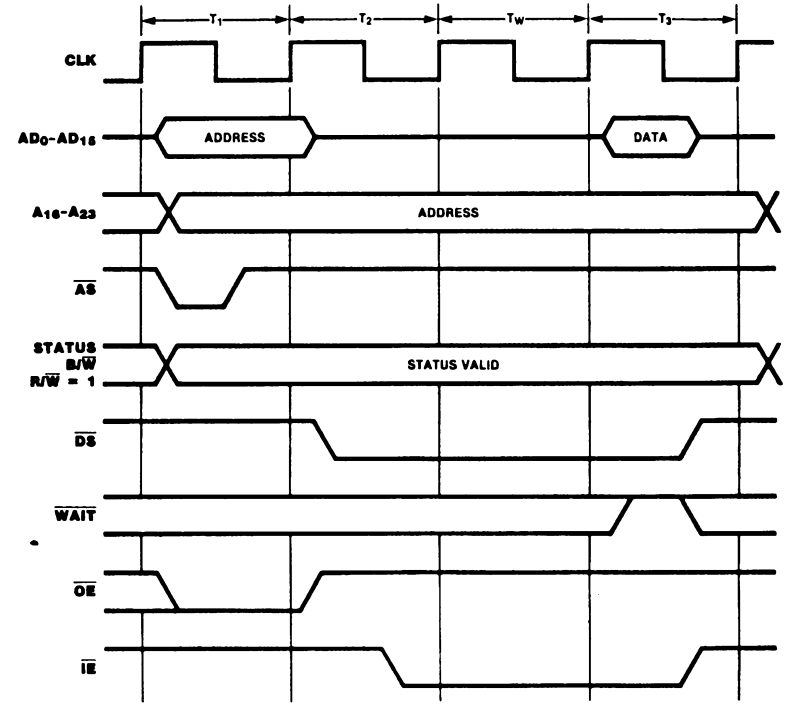


Figure 13-6. Memory Read Timing with Internal Wait Cycle

The $\overline{WAIT}$ input is also sampled on the falling edge of the clock during the third clock cycle; if $\overline{WAIT}$ is low, another bus clock cycle is added before sampling the data. Wait states can also be added through programming of the Bus Timing and Initialization register and Bus Timing and Control register. For example, Figures 13-4, 13-5, and 13-6 illustrate memory transactions with one wait state.

13.5.1.3 Burst Memory Transactions

The Z-BUS configuration of the Z280 MPU supports a special kind of memory transaction called a "burst memory transaction" for use in systems employing burst-mode memory devices. Control bits in the Cache Control register indicate whether portions of the memory system can support burst transactions; burst mode can be specified for either the upper half of memory (A_{23} = 1), the lower half of memory (A_{23} = 0), or both.

Burst memory transactions are used only during instruction fetches to "prefetch" instructions into the on-chip cache. In a burst memory read, four consecutive words of memory are read. If a byte is to be read from a portion of external memory that supports burst transactions, and that read operation is cacheable, the CPU reads the four words that contain the desired byte of the instruction with a single burst transaction. The address of the first word read during a burst transaction has zeros in the three least significant bits. The CPU reads a total of eight bytes via four word transfers, where the last byte read has all ones in the three least significant bits of its address. This effectively increases the bus bandwidth by prefetching a cache block on a cache miss. Burst transactions are not used when fetching templates in extended instructions.

The timing of a burst transaction is illustrated in Figure 13-7. During burst transactions, four Data Strobes are generated with a single Address Strobe. Timing for the first data transfer is identical to that for a single memory read, including the insertion of automatic wait states.

This first transfer is immediately followed by three more transfers in the next three bus clock cycles. The $\overline{WAIT}$ input is sampled during each transfer and any resulting wait states, thereby allowing wait states to be added before any of the transfers. However, automatic wait states are added only before the first transfer.

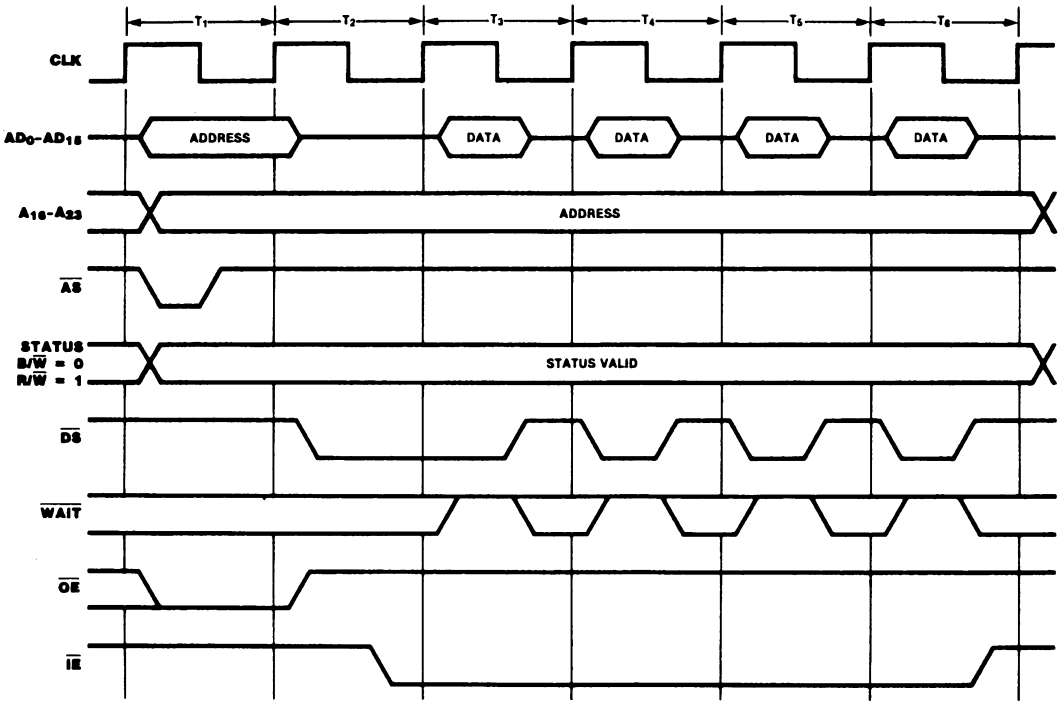


Figure 13-7. Burst Memory Read Timing

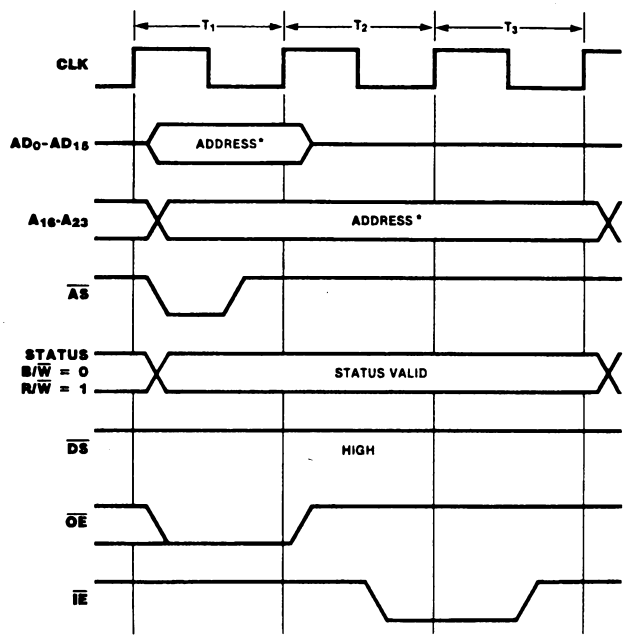
13.5.1.4 Test and Set Memory Transactions

The Test and Set (TSET) instruction provides a locking mechanism that can be used to synchronize software processes in a multitasking or multiprocessor system where exclusive access to certain resources is required. TSET tests and sets semaphores that control access to shared resources. Execution of TSET involves a memory read followed immediately by a memory write; the memory read followed by the memory write is one indivisible operation. The testing and setting of a semaphore requires the semaphore to be read from memory, modified, then written back into the same memory location. During the first of these two memory operations, the "1111" status code is placed on the ST_3-ST_0 status lines. This is particularly useful in a multiple microprocessor environment with semaphores in a shared memory area. The Test and Set status code can be used to control external circuitry that precludes memory access by another processor during the Test and Set semaphore operation. Furthermore, the $\overline{BUSREQ}$ input is disabled during a Test and Set operation to ensure that the semaphore is tested and set without any intervening accesses.

13.5.2 Halt and Refresh Transactions

There are two kinds of bus transactions that do not transfer data: Halt and Refresh transactions. These transactions are similar to memory transactions, except that $\overline{DS}$ remains high, the $\overline{WAIT}$ input is not sampled, and no data is transferred.

The Halt transaction (Figure 13-8) is generated when a HALT instruction is encountered or a fatal sequence of traps occurs. The "0011" status code on the ST_3-ST_0 lines identifies the Halt transaction. For Halt transactions generated by the HALT instruction, once the Halt transaction is executed, all subsequent CPU activity is suspended until an active interrupt request or reset is detected. After Halt transactions generated due to a fatal condition, all CPU activity is suspended until an active reset is detected (see section 6.6). However, Refresh transactions or DMA transfers may occur while the CPU is in the Halt state; also, the bus can be granted. The address emitted during the address phase of the Halt transaction is the address of the Halt instruction or the instruction that initiated the fatal sequence of traps.



*Address of Halt Instruction.

Figure 13-8. Halt Timing

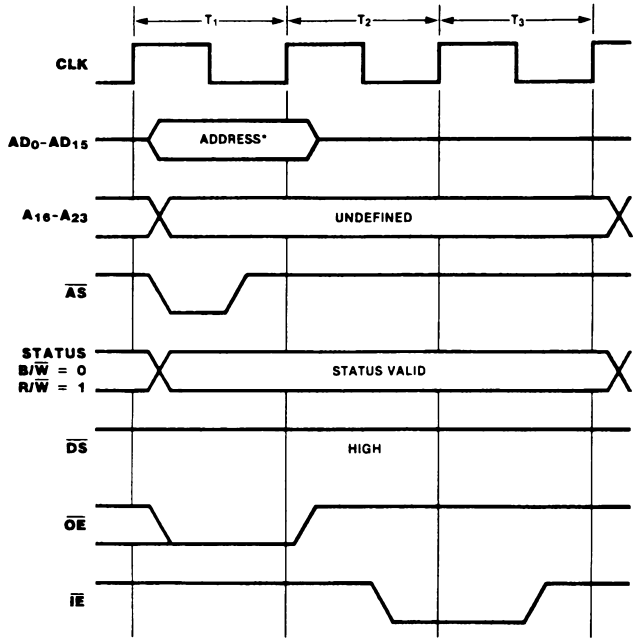
A memory refresh transaction (Figure 13-9) is generated by the Z280 MPU refresh mechanism and can occur immediately after the final clock cycle of any other transaction. The memory refresh counter's 10-bit address is emitted on AD_0-AD_9 when $\overline{AS}$ is asserted; the contents of the remaining address lines are undefined. The "0001" status code on the ST_3-ST_0 lines identifies the Refresh transaction. This transaction can be used to generate refreshes for dynamic RAMs. Refreshes may occur while the CPU is in the Halt state.

13.5.3 I/O Transactions

I/O Transactions move data to or from peripherals and are generated during the execution of I/O instructions or during DMA-controlled transfers. I/O transactions to devices in I/O pages FE_H and FF_H do not generate external bus transactions.

Figures 13-10 and 13-11 illustrate I/O transaction timing. I/O transactions are four clock cycles long at a minimum, and, like memory transactions, may be lengthened by the addition of wait cycles. I/O transaction timing is similar to memory transaction timing with one automatic wait state. The "0010" status code on the ST_3-ST_0 lines indicates that an I/O transaction is taking place, and the R/$\overline{W}$ line indicates the direction of the data transfer. The I/O address is found on AD_0-AD_{15} and A_{16}-A_{23} when $\overline{AS}$ rises. For read operations, $\overline{DS}$ and $\overline{IE}$ are asserted during the second clock cycle, and input data from the peripheral is sampled by the bus master during the fourth cycle (unless additional wait states are inserted in the transaction). Note that $\overline{DS}$ falls near the middle of T2 for I/O read transactions (as opposed to the beginning of T2 for memory reads); this provides peripheral control logic with additional time for address decoding. For write operations, $\overline{DS}$ is asserted during the second cycle with $\overline{OE}$ remaining asserted; output data to the peripheral is placed on the bus at this time.

For byte I/O operations (B/$\overline{W}$ = high), the byte of data is always transferred on the AD_0-AD_7 bus lines, regardless of the address of the peripheral device. For word I/O operations, the most significant byte of data is transferred on AD_0-AD_7 and the least significant byte on AD_8-AD_{15}, as with word memory transactions.



*10 least-significant bits are Refresh address.

Figure 13-9. Memory Refresh Timing

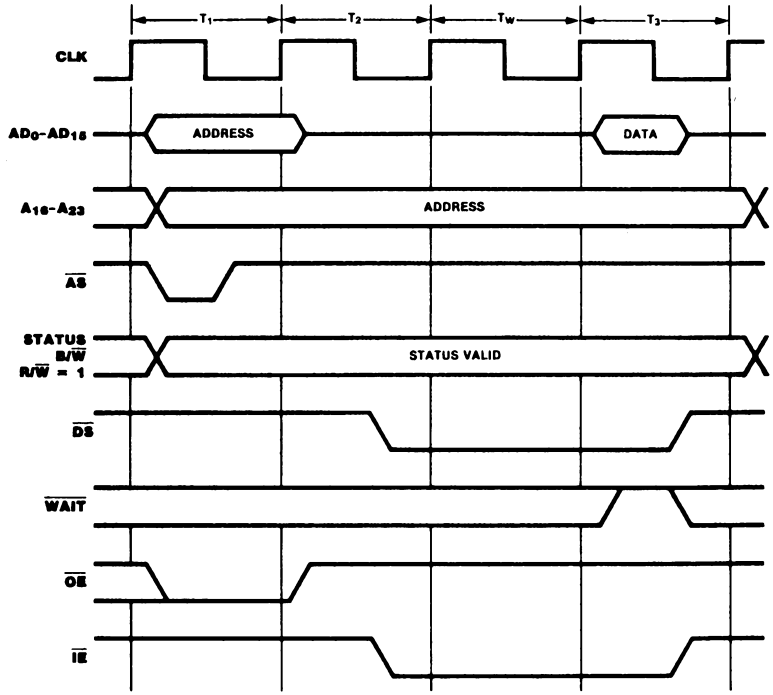


Figure 13-10. I/O Read Timing

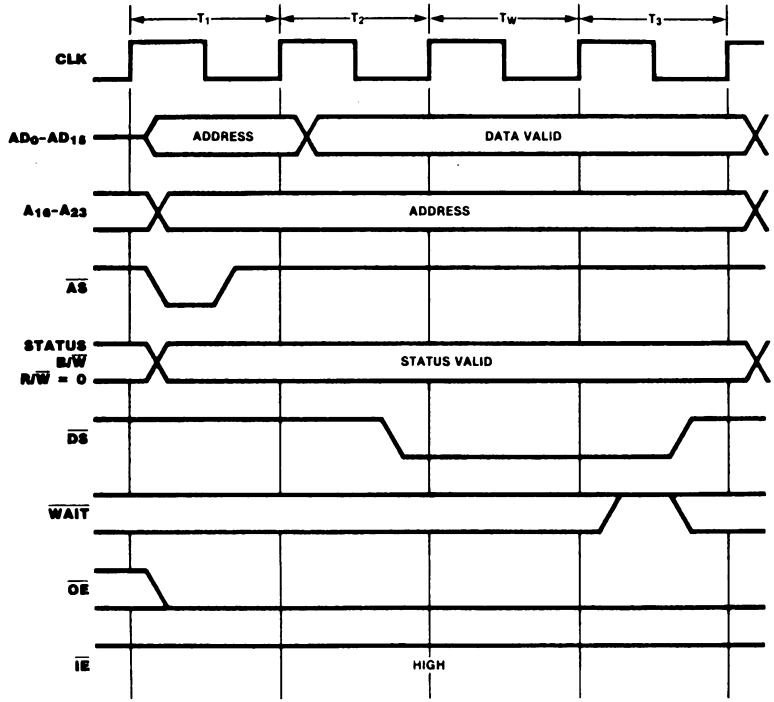


Figure 13-11. I/O Write Timing

13.5.4 Interrupt Acknowledge Transactions

Interrupt Acknowledge transactions acknowledge an interrupt and read an identifier from the device that generated the interrupt. These transactions are generated automatically by the CPU when an interrupt request is detected.

Interrupt Acknowledge transactions are five cycles long at a mimimum, with two automatic wait cycles (Figure 13-12). The wait cycles are used to give the interrupt priority daisy chain (or other priority resolution devices) time to settle before the identifier is read. Additional automatic wait states can be generated by programming the Bus Timing and Control register.

The ST_3-ST_0 status lines indicate the type of interrupt being acknowledged. No address is generated, so the contents of the address bus are undefined when $\overline{AS}$ is asserted. The R/$\overline{W}$ line indicates read (high), and the B/$\overline{W}$ line indicates word (low). The identifier is sampled by the CPU on the $\overline{AD}$ lines at the falling clock edge before $\overline{DS}$ is raised high.

There are two places where the $\overline{WAIT}$ line is sampled and, thus, where wait states can be inserted by external circuitry. The first, during T2, serves to delay the falling edge of $\overline{DS}$ to allow the daisy chain a longer time to settle; the second, during T3, serves to delay the point at which the identifier is read. Software-generated wait states can also be added at either time via programming of the DC and I/O fields in the Bus Timing and Control register. As always, software-generated wait states are inserted into the transaction before the external $\overline{WAIT}$ signal is sampled.

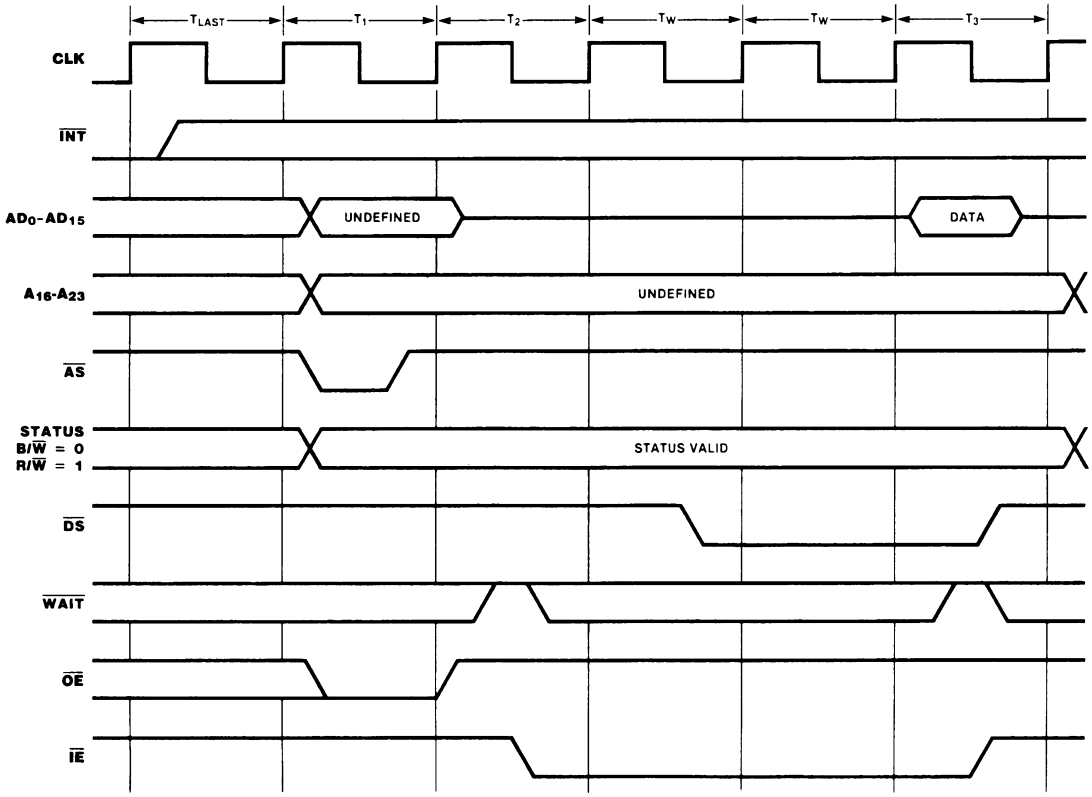


Figure 13-12. Interrupt Acknowledge Timing

13.5.5 Extended Processing Unit (EPU) Transactions

Z280 MPUs in the Z-BUS configuration can operate in conjunction with one or more Extended Processing Units (EPUs). Functioning as a coprocessor, the EPU monitors the status and timing signals output by the CPU so that it knows when to participate in a transaction. The Z280 MPU provides the address, status, and timing signals while the EPU supplies or captures data. Each of the four possible types of transactions that require EPU participation are signalled by the Z280 MPU ST_3-ST_0 outputs. CPU and EPU interaction is fully described in section 10.5.

13.5.5.1 EPU Instruction Fetch

When the Z280 CPU encounters an extended instruction, the state of the EPU Enable bit in the Trap Control register is examined. If the EPU Enable bit is zero, the Z280 generates an Extended Instruction trap. If the EPU Enable bit is set to 1, then the four-byte EPU template is fetched from memory using memory transactions and captured by both the CPU and EPU. The "1101" status code on the ST_3-ST_0 lines indicates when the first word of the template is fetched, and the "1100" status code indicates fetches of the subsequent template word or words, depending on the alignment. The CPU fetches the template from external memory using two word transactions if the template is aligned (that is, starts on an even address) or a byte transaction followed by two word transactions if the template is unaligned. The opcode and addressing mode portion of the extended instruction may be executed from cache, but the template will always be fetched from external memory.

In a multiple EPU system, the EPU that is to participate in the execution of an extended instruction is selected implicitly by an identification code in the instruction template. Thus, there is no indication on the bus as to which EPU is cooperating with the CPU at any given time.

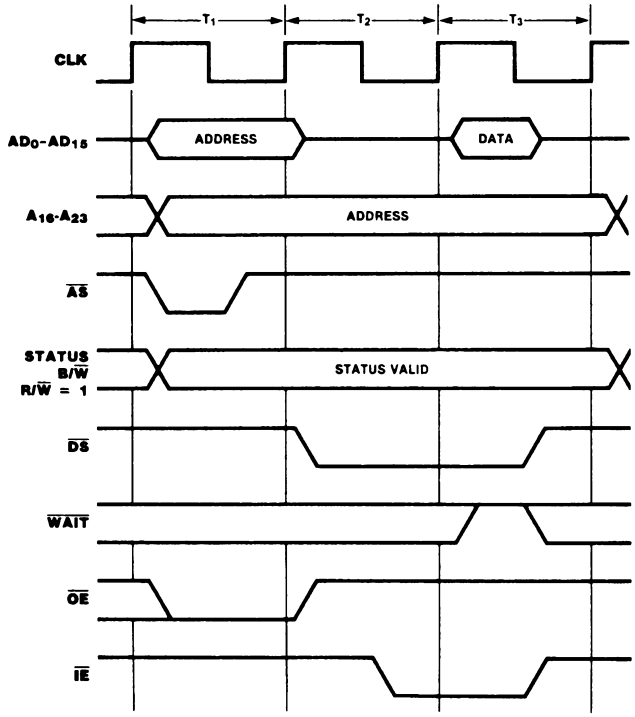


Figure 13-13. Memory to EPU Timing

13.5.5.2 Memory-EPU Transactions

If an extended instruction involves a read or write to memory, then the transfers of data between memory and the EPU are the next non-refresh transactions performed by the CPU following the fetch of the template. The timing of memory-EPU data transfers is shown in Figures 13-13 and 13-14. The EPU must supply the data during write operations (R/$\overline{W}$ = low) or capture the data during read operations (R/$\overline{W}$ = high), just as if it were part of the CPU. In both cases, the CPU 3-states its AD lines while data is being transferred ($\overline{DS}$ = low). EPU reads from memory are three cycles long unless extended by wait states. EPU writes to memory are six cycles long unless extended by wait states.

13.5.5.3 EPU-CPU Transactions

If an extended instruction involves a transfer from the EPU to the Z280 CPU, the next non-refresh transaction following the fetch of the template is the EPU-to-CPU data transfer (Figure 13-15).

EPU-to-CPU transactions have the same form as I/O read transactions and thus are four clock cycles long, unless extended by wait states. Although $\overline{AS}$ is asserted, no address is generated and the contents of the address bus are undefined. The "1110" status code on the ST_3-ST_0 lines indicate an EPU-to-CPU transaction.

13.5.5.4 $\overline{PAUSE}$ Timing

The $\overline{PAUSE}$ signal is used to synchronize CPU-EPU activity in the case of overlapping extended instructions. The CPU samples the $\overline{PAUSE}$ signal within one bus clock period of the completion of the fetch of an extended instruction's template (Figure 13-16). If $\overline{PAUSE}$ is active when sampled, the CPU enters an idle state wherein all CPU activity is suspended. While in this idle state, the CPU samples the $\overline{PAUSE}$ input each processor clock cycle until $\overline{PAUSE}$ is deasserted. The CPU then resumes operation at the point at which it was suspended, either by executing the data transactions associated with the extended instruction (in the case of an extended instruction specifying an EPU-memory or CPU-EPU data transfer) or by starting the fetch of the next instruction (in the case of an extended instruction specifying an internal EPU operation).

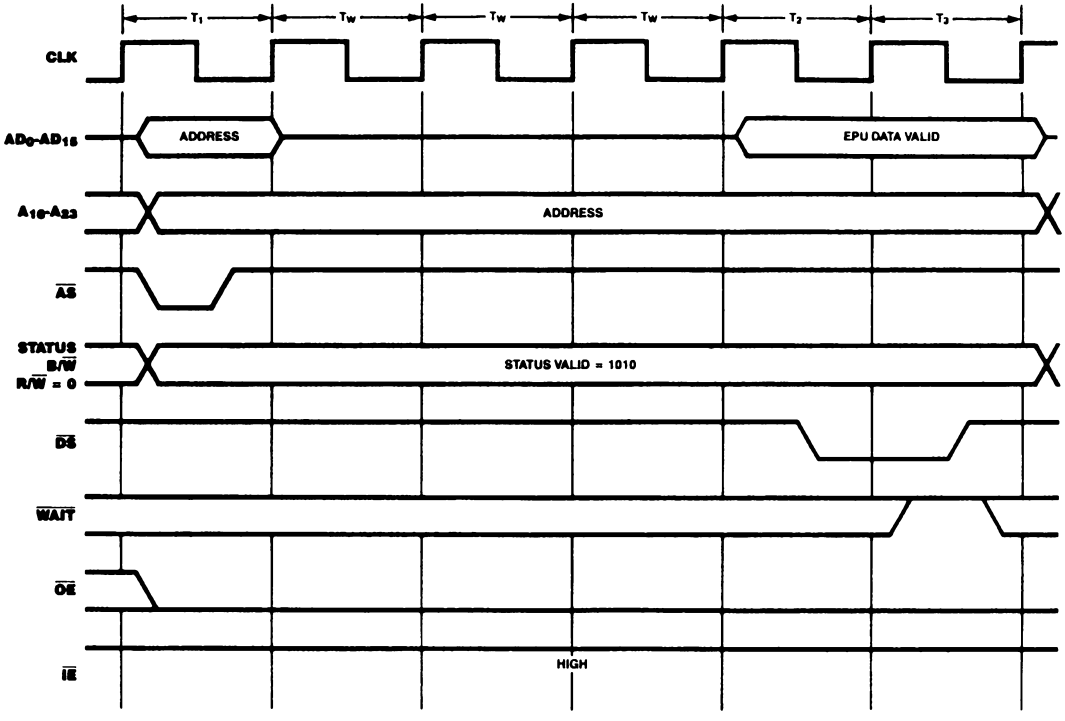


Figure 13-14. EPU Write to Memory

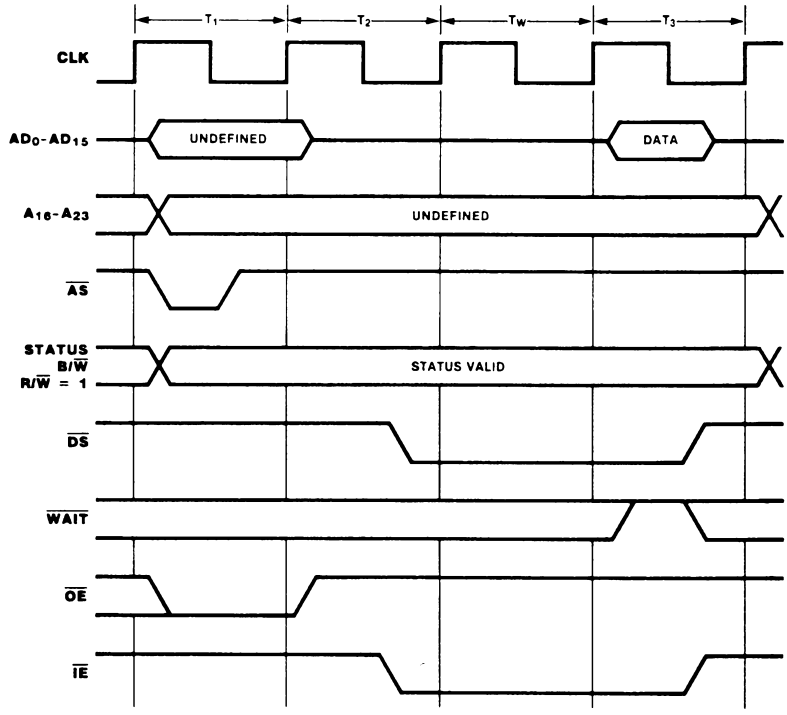


Figure 13-15. EPU to CPU Timing

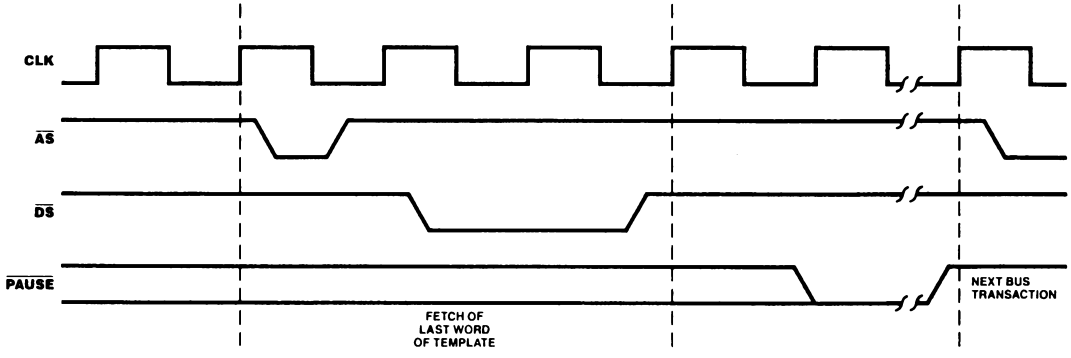


Figure 13-16. PAUSE Timing

13.5.6 DMA Flyby Transactions

On-chip DMA channels 0 and 1 can transfer data between memory and peripheral devices using flyby type transfers; external DMA controllers in Z280 MPU systems (such as the Z8016 DTC) may also have this capability. The timing of flyby transactions is similar to memory transaction timing, with the exception that the DMA Strobe ($\overline{DMASTB}$) signal is activated; the $\overline{DMASTB}$ signal is used to select the participating I/O device that must capture or supply the data during the memory access.

Flyby transactions controlled by the on-chip DMA channels always include one automatic wait state (Figures 13-17 and 13-18). As with all memory transactions, other hardware- and software-generated wait states can be added to the transaction. The external $\overline{WAIT}$ signal is sampled at two different times: during the automatic wait state and during T3.

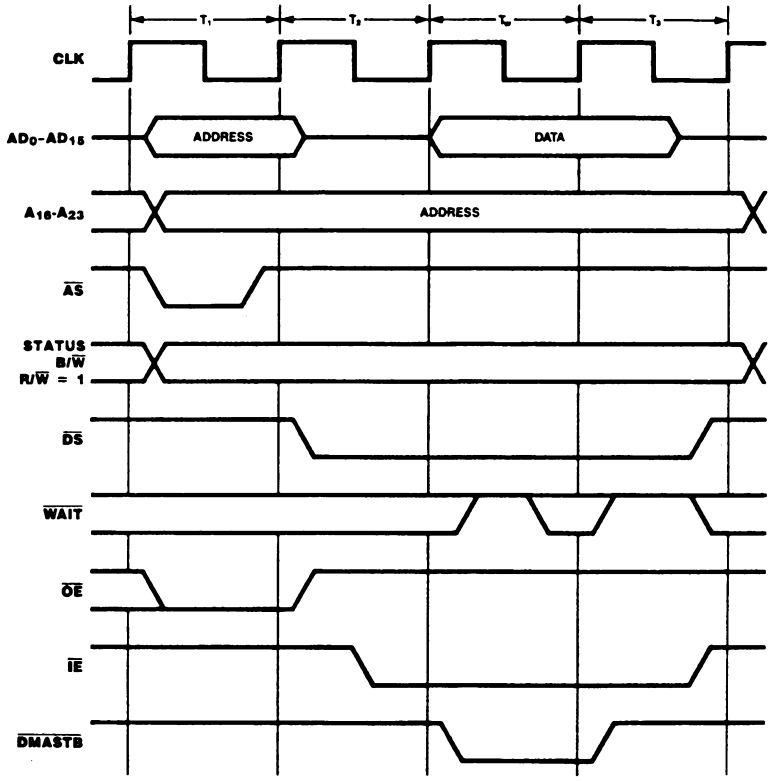


Figure 13-17. On-Chip DMA Channel Flyby Memory Read Transaction

For flyby transactions that read from memory and write to a peripheral (Figure 13-17), $\overline{DMASTB}$ is asserted during the automatic wait state and any subsequent wait states due to an active $\overline{WAIT}$ signal. Thus, if the $\overline{WAIT}$ input is asserted during the automatic wait state, the additional wait states extend the width of the $\overline{DMASTB}$ pulse. Wait states added via the assertion of $\overline{WAIT}$ during T3 (after $\overline{DMASTB}$ is deasserted) stretch the $\overline{DS}$ signal without affecting $\overline{DMASTB}$.

For flyby transactions that read from a peripheral and write to memory (Figure 13-18), $\overline{DMASTB}$ is asserted at the beginning of T2 and remains asserted until the second half of T3. The $\overline{DS}$ signal is asserted only during the automatic wait state. Wait states added via the assertion of $\overline{WAIT}$ stretch the $\overline{DMASTB}$ signal without affecting $\overline{DS}$.

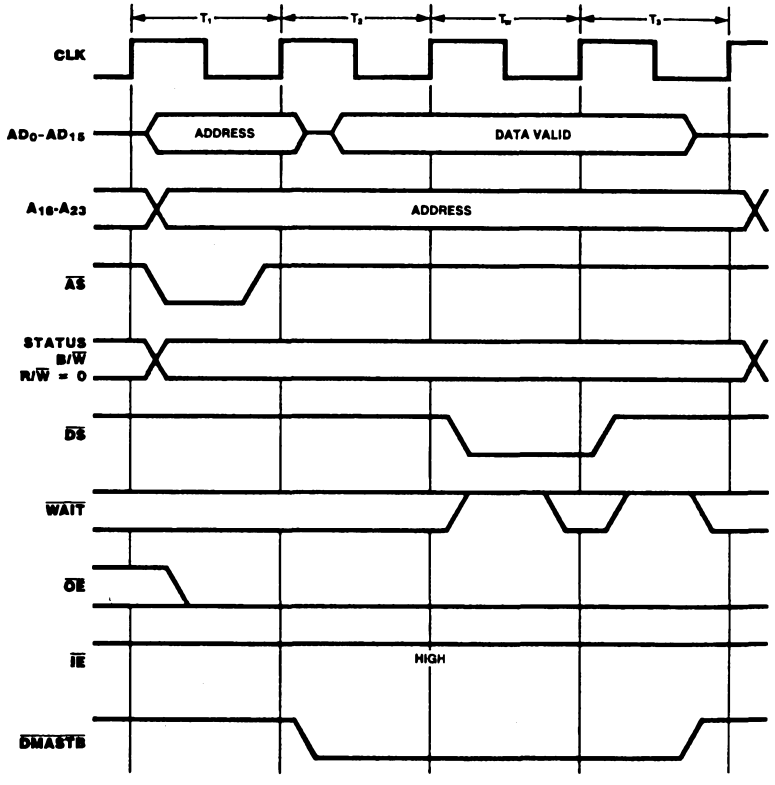


Figure 13-18. On-Chip DMA Channel Flyby Memory Write Transaction

13.6 REQUESTS

The Z280 MPU supports three types of request signals: interrupt requests, local bus requests, and global bus requests. A request is answered according to its type. Interrupt requests are generated by peripheral devices; the Z280 MPU responds with an Interrupt Acknowledge transaction. Local bus requests are initiated by an external potential bus master; the Z280 MPU responds by relinquishing the bus and generating an active Bus Acknowledge signal. Global bus requests are generated by the Z280 CPU or an on-chip DMA channel to access a global bus; the Z280 MPU receives a Global Bus Acknowledge signal in response to the request.

13.6.1 Interrupt Requests

The Z280 CPU supports two types of interrupts, maskable and nonmaskable ($\overline{NMI}$). The interrupt request line from a device capable of generating interrupts can be tied to the Z280 MPU's $\overline{NMI}$ or maskable interrupt request inputs; several devices can be connected to one interrupt request input, with interrupt priorities established via external logic or a priority daisy chain.

Nonmaskable interrupt requests are edge-triggered, but maskable interrupts are level-triggered. Any high-to-low transition on the $\overline{NMI}$ input is asynchronously edge-detected, and an internal $\overline{NMI}$ latch is set. At the beginning of the last clock cycle during execution of an instruction, the maskable interrupt inputs are sampled along with the state of the internal $\overline{NMI}$ latch. If an interrupt is detected, and that interrupt is enabled in the Master Status register, interrupt processing proceeds in accordance with the current interrupt mode, as described in Chapter 6.

13.6.2 Local Bus Requests

To generate transactions on the bus, a potential bus master (such as a DMA controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling $\overline{BUSREQ}$ low; the Z280 MPU responds by 3-stating its address, data, bus control, and bus status outputs and asserting an active $\overline{BUSACK}$, as described in section 10.2. The CPU regains control of the bus after $\overline{BUSREQ}$ rises. The on-chip DMA channels have higher priority than external devices requesting the bus via $\overline{BUSREQ}$.

13.6.3 Global Bus Requests

If the multiprocessor mode is specified in the Bus Timing and Initialization register, then the contents of the Local Address register determine the range of memory addresses dedicated to the shared global bus. Before accessing an address on the global bus, the Z280 MPU must issue a Global Bus Request ($\overline{GREQ}$) and receive an active Global Bus Acknowledge ($\overline{GACK}$) signal, as described in Section 10.3.

Figure 13-19 illustrates the timing of the global bus request/acknowledge sequence. When the Z280 MPU needs to access a location on the global bus, $\overline{GREQ}$ is asserted in order to request use of the global bus. $\overline{GACK}$ is then sampled on each successive rising edge of the clock; when $\overline{GACK}$ becomes active (and if $\overline{BUSREQ}$ is not asserted), the memory transaction proceeds as described in section 13.5.1. $\overline{GREQ}$ is deasserted in the bus cycle immediately following the end of the memory transaction (except when executing the Test and Set instruction, where both the memory read and write operations are executed before deasserting $\overline{GREQ}$).

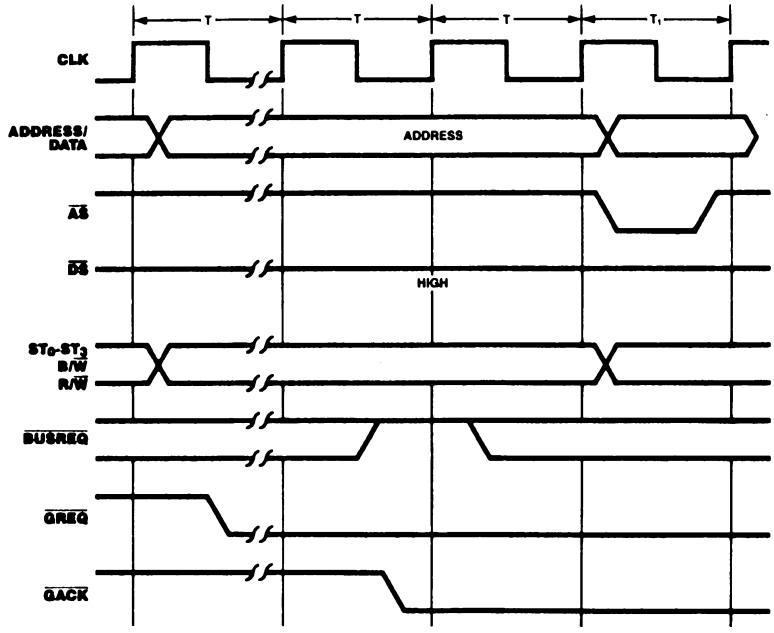


Figure 13-19. Multiprocessor Mode Timing

Appendix A.
Z80/Z280 Compatibility

The Z280 MPU architecture is an upward-compatible extension of the Z80 CPU architecture. This compatibility extends to the instruction set, register architecture, interrupt structure, and bus structure of the Z280 MPU and Z80 CPU.

The Z80 CPU's instruction set is a subset of the Z280 MPU's instruction set. Thus, the Z280 MPU is completely binary-compatible with Z80 code. However, since some Z80 instructions, such as HALT, are privileged instructions in the Z280 MPU, complete compatibility is achieved only when the Z280 MPU is executing in the system mode. All Z80 software will execute successfully on a Z280 MPU running in system mode, provided that the software contains no timing dependencies, does not modify itself, and does not use any of the Z80's reserved instruction encodings.

Since the Z280 MPU is binary-code compatible with the Z80 CPU, the Z280 MPU's general-purpose register set is the same as the Z80 CPU's, with the exception of the Stack Pointer. The Z280 MPU contains both a System Stack Pointer and a User Stack Pointer, whereas the Z80 CPU has only one Stack Pointer register. In the Z80 CPU, the R register is used to indicate the next refresh address; in the Z280 MPU, the R register is not involved with the refresh logic and may be used by the programmer as a general-purpose storage register.

The Z280 MPU's interrupt structure is also an upward-compatible extension of the Z80 CPU's. The Z280 MPU supports all three interrupt modes found on the Z80 CPU, as well as a fourth interrupt mode new to the Z280 MPU.

The Z80 Bus configurations of the Z280 MPU are also bus-compatible with the Z80 CPU, generating the same $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$, and $\overline{MREQ}$ bus control and status signals. However, $\overline{M1}$ is asserted during every instruction fetch and interrupt acknowledge cycle in the Z80 CPU; for the Z280 MPU, $\overline{M1}$ is asserted only during the special RETI bus transaction and interrupt acknowledge cycles. The Z8400 family of peripherals interface directly to both Z80 CPUs and Z80 bus configuration of the Z280 MPUs.

Following a reset, the Z280 MPU takes on a configuration that is fully compatible with Z80 code. The Memory Management Unit is disabled, meaning that the 16-bit logical addresses from the Z280 CPU are routed directly to the 16 least significant address pins on the external bus. The User/System bit in the Master Status register specifies system-mode operation, allowing execution of privileged instructions and enabling the System Stack Pointer. The I/O Page register is cleared to all 0s and Interrupt Mode 0 is selected. The Trap Control register is cleared to all zeros, disabling System Stack Overflow Warning traps and designating that I/O instructions are not privileged. All Z80 instructions can be successfully executed (and may execute from the on-chip memory that is enabled as an instruction-only cache upon reset). The Z280 MPU will remain in a Z80-compatible configuration as long as Z80 code is executed, since the Load Control instruction that acts on the Z280 MPU's control registers is not part of the Z80 instruction set.

The software routine shown below can be used to determine if code is executing on a Z80 CPU or Z280 MPU. This facilitates development of programs that can execute on either processor, but contain special routines invoked only when executing on a Z280 MPU and, therefore, allowing use of Z280 MPU features not available on the Z80 CPU. The routine differentiates the Z80 CPU from the Z280 MPU by executing the instruction with machine code $CB37_H$. This instruction code is reserved in the Z80 CPU, and results in logically shifting the A register one bit to the left while shifting a 1 into the least significant bit. For the Z280 MPU, $CB37_H$ is the code for the Test and Set instruction. If the A register holds a 40_H before executing this instruction code, the A register holds an 81_H and the Sign flag is set to 1 after executing the instruction on a Z80 CPU; the A register holds an FF_H and the Sign flag is cleared to 0 after executing the instruction on a Z280 MPU.

Code to Distinguish Execution on a Z80 CPU and Z280 MPU

```
; This instruction sequence exploits the difference when executing the CB37H
; machine code on the Z80 CPU and Z280 MPU, to allow a program to determine which
; processor it is executing on.  This instruction sets the S flag on the Z80 CPU
; and clears the S flag on the Z280 MPU.  The A and F registers are used by the
; routine.

        LD      A,40H       ; Initialize the operand.
        DEFB    0CBH,037H   ; This instruction will set the S flag on the
                            ; Z80 CPU and clear the S flag on the Z280 MPU.
        JP      M,Z80       ; Now test the flag and jump.
             or
        JP      P,Z280
```

Appendix B.
Z280 MPU Instruction Formats

Four formats are used to generate the machine-language bit encodings for the Z280 MPU instructions. Three formats are used for instructions that are executed solely by the Z280 CPU. (These same three formats are used for Z80 CPU instruction encoding.) A fourth format is dedicated to instructions that involve Extended Processing Units (EPUs).

The bit encodings of the Z280 MPU instructions are partitioned into bytes. Every instruction encoding contains one byte dedicated to specifying the type of operation to be performed; this byte is referred to as the instruction's operation code (opcode). Besides specifying a particular operation, opcodes typically include bit encodings specifying the operand addressing mode for the instruction and identifying any general-purpose registers used by the instruction. Along with the opcode, instruction encodings may include bytes that contain an address, displacement, and/or immediate value used by the instruction, and special bytes called "escape codes" that determine the meaning of the opcode itself.

By themselves, one byte opcodes would allow the encoding of only 256 unique instructions. Therefore, special "escape codes" that precede the opcode in the instruction encoding are used to expand the number of possible instructions. There are two types of escape codes: addressing mode escape codes and opcode escape codes. Escape codes are one byte in length.

Three of the instruction formats are differentiated by the opcode escape value used; the fourth format is for instructions that include an EPU template. Format 1 is for instructions without an opcode escape byte, Format 2 is for instructions whose opcode escape byte has the value ED_H, and Format 3 is for instructions whose opcode escape byte has the value CB_H. Instructions that support EPUs use Format 4 and always have the opcode escape byte with value ED_H as the first byte of the instruction encoding. In Formats 2 and 4, the opcode escape byte immediately preceeds the opcode byte itself.

In Format 3, a 1-byte displacement may be between the opcode escape byte and opcode itself. Opcode escape bytes are used to distinguish between two different instructions with the same opcode byte, thereby allowing more than 256 unique instructions. For example, the 01_H opcode, when alone, specifies a form of the Load Register Word instruction; when preceded by the CB_H escape byte, the opcode 01_H specifies a Rotate Left Circular instruction.

Addressing mode escape codes are used to determine the type of encoding for the addressing mode field within an instruction's opcode, and can be used in instructions with and without opcode escape values. An addressing mode escape byte can have the value DD_H or FD_H. The addressing mode escape byte, if present, is always the first byte of the instruction's machine code, and is immediately followed by either the opcode (Format 1) or the opcode escape byte (Formats 2 and 3). For example, the 79_H opcode, when alone, specifies a Load Accumulator instruction using Register addressing for the source operand; when preceded by the DD_H escape byte, the opcode 79_H specifies a Load Accumulator instruction using Base Index addressing for the source operand.

The four instruction formats are shown in Tables B-1 through B-4. Within each format, several different configurations are possible, depending on whether the instruction involves addressing mode escape bytes, addresses, displacements, or immediate data. In Tables B-1 through B-4, the symbol "A.esc" is used to indicate the presence of an addressing mode escape byte, "disp." is an abbreviation for displacement, "addr." is an abbreviation for address, and "temp." is an abbreviation for template. Templates in EPU instructions are four-byte fields that include the bit encodings that specify EPU operation.

Table B-1. Format 1 Instruction Encodings

Instruction Format				Example Instruction	
				Assembly	Machine Code (Hex)
	opcode			LD A,C	79
	opcode	2-byte address		LD A,(addr)	3A addr(low) addr(high)
	opcode	1-byte displacement		DJNZ addr	10 disp
	opcode	immediate		LD E,n	IE n
A.esc	opcode			LD A,(HL + IX)	DD 79
A.esc	opcode	2-byte address		LD IX,(addr)	DD 2A addr(low) addr(high)
A.esc	opcode	1-byte displacement		LD A,(IX + d)	DD 7E disp
A.esc	opcode	2-byte displacement		LD A,(IX + dd)	FD 79 d(low) d(high)
A.esc	opcode	immediate		LD IX,nn	DD 21 n(low) n(high)
A.esc	opcode	2-byte address	immediate	LD (addr),n	DD 3E addr(low) addr(high) n
A.esc	opcode	1-byte displacement	immediate	LD (IY + d),n	FD 36 d n
A.esc	opcode	2-byte displacement	immediate	LD <addr>,n	FD 06 disp(low) disp(high) n

Table B-2. Format 2 Instruction Encodings

Instruction Format					Example Instruction	
					Assembly	Machine Code (Hex)
	ED	opcode			MULT A,B	ED C0
	ED	opcode	immediate		SC nn	ED 71 n(low) n(high)
	ED	opcode	2-byte address		LD BC,(addr)	ED 4B addr(low) addr(high)
	ED	opcode	2-byte displacement		LD (HL + dd),A	ED 3B d(low) d(high)
A.esc	ED	opcode			MULT A,IY	FD ED E8
A.esc	ED	opcode	2-byte address		MULT A,(addr)	DD ED F8 addr(low) addr(high)
A.esc	ED	opcode	1-byte displacement		MULT A,(IY + d)	FD ED F8 d
A.esc	ED	opcode	2-byte displacement		LD IX,(IY + dd)	DD ED 34 d(low) d(high)
A.esc	ED	opcode	2-byte immediate		MULTUW HL,nn	FD ED F3 n(low) n(high)

Table B-3. Format 3 Instruction Encodings

Instruction Format				Example Instruction	
				Assembly	Machine Code (Hex)
	CB	opcode		RLC (HL)	CB 06
A.esc	CB	1-byte displacement	opcode	RCL (IX + d)	DD CB d 06

Table B-4. Format 4 Instruction Encodings

Instruction Format				Example Instruction	
				Assembly	Machine Code (Hex)
ED	opcode	4-byte template		EPU ← (HL)	ED A6 temp1 temp2 temp3 temp4
ED	opcode	2-byte displacement	4-byte template	EPU ← (HL + dd)	ED BC d(low) d(high) temp1 temp2 temp3 temp4
ED	opcode	2-byte address	4-byte template	EPU ← (addr)	ED A7 addr(low) addr(high) temp1 temp2 temp3 temp4

Appendix C.
Instructions in Alphabetic Order

SOURCE CODE		OBJECT CODE
ADC	A,(HL)	8E
ADC	A,(HL+IX)	DD89
ADC	A,(HL+IY)	DD8A
ADC	A,(HL+1122H)	FD8B2211
ADC	A,(IX+IY)	DD8B
ADC	A,(IX+55H)	DD8E55
ADC	A,(IX+1122H)	FD892211
ADC	A,(IY+55H)	FD8E55
ADC	A,(IY+1122H)	FD8A2211
ADC	A,(PC+1122H)	FD882211
ADC	A,(SP+1122H)	DD882211
ADC	A,(3344H)	DD8F4433
ADC	A,A	8F
ADC	A,B	88
ADC	A,C	89
ADC	A,D	8A
ADC	A,E	8B
ADC	A,H	8C
ADC	A,IXH	DD8C
ADC	A,IXL	DD8D
ADC	A,IYH	FD8C
ADC	A,IYL	FD8D
ADC	A,L	8D
ADC	A,66H	CE66
ADC	HL,BC	ED4A
ADC	HL,DE	ED5A
ADC	HL,HL	ED6A
ADC	HL,SP	ED7A
ADC	IX,BC	DDED4A
ADC	IX,DE	DDED5A
ADC	IX,IX	DDED6A
ADC	IX,SP	DDED7A
ADC	IY,BC	FDED4A
ADC	IY,DE	FDED5A
ADC	IY,IY	FDED6A
ADC	IY,SP	FDED7A
ADD	A,(HL)	86
ADD	A,(HL+IX)	DD81
ADD	A,(HL+IY)	DD82
ADD	A,(HL+1122H)	FD832211
ADD	A,(IX+IY)	DD83
ADD	A,(IX+55H)	DD8655
ADD	A,(IX+1122H)	FD812211
ADD	A,(IY+55H)	FD8655
ADD	A,(IY+1122H)	FD822211
ADD	A,(PC+1122H)	FD802211
ADD	A,(SP+1122H)	DD802211
ADD	A,(3344H)	DD874433
ADD	A,A	87
ADD	A,B	80
ADD	A,C	81

SOURCE CODE		OBJECT CODE
ADD	A,D	82
ADD	A,E	83
ADD	A,H	84
ADD	A,IXH	DD84
ADD	A,IXL	DD85
ADD	A,IYH	FD84
ADD	A,IYL	FD85
ADD	A,L	85
ADD	A,66H	C666
ADD	HL,A	ED6D
ADD	HL,BC	09
ADD	HL,DE	19
ADD	HL,HL	29
ADD	HL,SP	39
ADD	IX,A	DDED6D
ADD	IX,BC	DD09
ADD	IX,DE	DD19
ADD	IX,IX	DD29
ADD	IX,SP	DD39
ADD	IY,A	FDED6D
ADD	IY,BC	FD09
ADD	IY,DE	FD19
ADD	IY,IY	FD29
ADD	IY,SP	FD39
ADDW	HL,(HL)	DDEDC6
ADDW	HL,(IX+1122H)	FDEDC62211
ADDW	HL,(IY+1122H)	FDEDD62211
ADDW	HL,(PC+1122H)	DDEDF62211
ADDW	HL,(3344H)	DDEDD64433
ADDW	HL,BC	EDC6
ADDW	HL,DE	EDD6
ADDW	HL,HL	EDE6
ADDW	HL,IX	DDEDE6
ADDW	HL,IY	FDEDE6
ADDW	HL,SP	EDF6
ADDW	HL,3344H	FDEDF64433
AND	A,(HL)	A6
AND	A,(HL+IX)	DDA1
AND	A,(HL+IY)	DDA2
AND	A,(HL+1122H)	FDA32211
AND	A,(IX+IY)	DDA3
AND	A,(IX+55H)	DDA655
AND	A,(IX+1122H)	FDA12211
AND	A,(IY+55H)	FDA655
AND	A,(IY+1122H)	FDA22211
AND	A,(PC+1122H)	FDA02211
AND	A,(SP+1122H)	DDA02211
AND	A,(3344H)	DDA74433
AND	A,A	A7
AND	A,B	A0
AND	A,C	A1

SOURCE CODE		OBJECT CODE
AND	A,D	A2
AND	A,E	A3
AND	A,H	A4
AND	A,IXH	DDA4
AND	A,IXL	DDA5
AND	A,IYH	FDA4
AND	A,IYL	FDA5
AND	A,L	A5
AND	A,66H	E666
BIT	0,(HL)	CB46
BIT	0,(IX+55H)	DDCB5546
BIT	0,(IY+55H)	FDCB5546
BIT	0,A	CB47
BIT	0,B	CB40
BIT	0,C	CB41
BIT	0,D	CB42
BIT	0,E	CB43
BIT	0,H	CB44
BIT	0,L	CB45
BIT	1,(HL)	CB4E
BIT	1,(IX+55H)	DDCB554E
BIT	1,(IY+55H)	FDCB554E
BIT	1,A	CB4F
BIT	1,B	CB48
BIT	1,C	CB49
BIT	1,D	CB4A
BIT	1,E	CB4B
BIT	1,H	CB4C
BIT	1,L	CB4D
BIT	2,(HL)	CB56
BIT	2,(IX+55H)	DDCB5556
BIT	2,(IY+55H)	FDCB5556
BIT	2,A	CB57
BIT	2,B	CB50
BIT	2,C	CB51
BIT	2,D	CB52
BIT	2,E	CB53
BIT	2,H	CB54
BIT	2,L	CB55
BIT	3,(HL)	CB5E
BIT	3,(IX+55H)	DDCB555E
BIT	3,(IY+55H)	FDCB555E
BIT	3,A	CB5F
BIT	3,B	CB58
BIT	3,C	CB59
BIT	3,D	CB5A
BIT	3,E	CB5B
BIT	3,H	CB5C
BIT	3,L	CB5D
BIT	4,(HL)	CB66
BIT	4,(IX+55H)	DDCB5566
BIT	4,(IY+55H)	FDCB5566
BIT	4,A	CB67
BIT	4,B	CB60
BIT	4,C	CB61
BIT	4,D	CB62
BIT	4,E	CB63
BIT	4,H	CB64
BIT	4,L	CB65
BIT	5,(HL)	CB6E
BIT	5,(IX+55H)	DDCB556E
BIT	5,(IY+55H)	FDCB556E
BIT	5,A	CB6F
BIT	5,B	CB68
BIT	5,C	CB69
BIT	5,D	CB6A
BIT	5,E	CB6B
BIT	5,H	CB6C
BIT	5,L	CB6D
BIT	6,(HL)	CB76
BIT	6,(IX+55H)	DDCB5576
BIT	6,(IY+55H)	FDCB5576
BIT	6,A	CB77
BIT	6,B	CB70
BIT	6,C	CB71
BIT	6,D	CB72
BIT	6,E	CB73
BIT	6,H	CB74
BIT	6,L	CB75
BIT	7,(HL)	CB7E
BIT	7,(IX+55H)	DDCB557E
BIT	7,(IY+55H)	FDCB557E
BIT	7,A	CB7F
BIT	7,B	CB78
BIT	7,C	CB79
BIT	7,D	CB7A
BIT	7,E	CB7B
BIT	7,H	CB7C
BIT	7,L	CB7D
CALL	(HL)	DDCD
CALL	(PC+1122H)	FDCD2211
CALL	C,(HL)	DDDC
CALL	C,(PC+1122H)	FDDC2211
CALL	C,3344H	DC4433
CALL	M,(HL)	DDFC
CALL	M,(PC+1122H)	FDFC2211
CALL	M,3344H	FC4433
CALL	NC,(HL)	DDD4
CALL	NC,(PC+1122H)	FDD42211
CALL	NC,3344H	D44433
CALL	NZ,(HL)	DDC4
CALL	NZ,(PC+1122H)	FDC42211
CALL	NZ,3344H	C44433
CALL	P,(HL)	DDF4
CALL	P,(PC+1122H)	FDF42211
CALL	P,3344H	F44433
CALL	PE,(HL)	DDEC
CALL	PE,(PC+1122H)	FDEC2211
CALL	PE,3344H	EC4433
CALL	PO,(HL)	DDE4
CALL	PO,(PC+1122H)	FDE42211
CALL	PO,3344H	E44433
CALL	Z,(HL)	DDCC
CALL	Z,(PC+1122H)	FDCC2211
CALL	Z,3344H	CC4433
CALL	3344H	CD4433
CCF		3F
CP	A,(HL)	BE
CP	A,(HL+IX)	DDB9
CP	A,(HL+IY)	DDBA

SOURCE CODE		OBJECT CODE
CP	A,(HL+1122H)	FDBB2211
CP	A,(IX+IY)	DDBB
CP	A,(IX+55H)	DDBE55
CP	A,(IX+1122H)	FDB92211
CP	A,(IY+55H)	FDBE55
CP	A,(IY+1122H)	FDBA2211
CP	A,(PC+1122H)	FDB82211
CP	A,(SP+1122H)	DDB82211
CP	A,(3344H)	DDBF4433
CP	A,A	BF
CP	A,B	B8
CP	A,C	B9
CP	A,D	BA
CP	A,E	BB
CP	A,H	BC
CP	A,IXH	DDBC
CP	A,IXL	DDBD
CP	A,IYH	FDBC
CP	A,IYL	FDBD
CP	A,L	BD
CP	A,66H	FE66
CPD		EDA9
CPDR		EDB9
CPI		EDA1
CPIR		EDB1
CPL		2F
CPW	HL,(HL)	DDEDC7
CPW	HL,(IX+1122H)	FDEDC72211
CPW	HL,(IY+1122H)	FDEDD72211
CPW	HL,(PC+1122H)	DDEDF72211
CPW	HL,(3344H)	DDEDD74433
CPW	HL,BC	EDC7
CPW	HL,DE	EDD7
CPW	HL,HL	EDE7
CPW	HL,IX	DDEDE7
CPW	HL,IY	FDEDE7
CPW	HL,SP	EDF7
CPW	HL,3344H	FDEDF74433
DAA		27
DEC	(HL)	35
DEC	(HL+IX)	DD0D
DEC	(HL+IY)	DD15
DEC	(HL+1122H)	FD1D2211
DEC	(IX+IY)	DD1D
DEC	(IX+55H)	DD3555
DEC	(IX+1122H)	FD0D2211
DEC	(IY+55H)	FD3555
DEC	(IY+1122H)	FD152211
DEC	(PC+1122H)	FD052211
DEC	(SP+1122H)	DD052211
DEC	(3344H)	DD3D4433
DEC	A	3D
DEC	B	05
DEC	BC	0B
DEC	C	0D
DEC	D	15
DEC	DE	1B
DEC	E	1D
DEC	H	25
DEC	HL	2B

SOURCE CODE		OBJECT CODE
DEC	IX	DD2B
DEC	IXH	DD25
DEC	IXL	DD2D
DEC	IY	FD2B
DEC	IYH	FD25
DEC	IYL	FD2D
DEC	L	2D
DEC	SP	3B
DECW	(HL)	DD0B
DECW	(IX+1122H)	FD0B2211
DECW	(IY+1122H)	FD1B2211
DECW	(PC+1122H)	DD3B2211
DECW	(3344H)	DD1B4433
DECW	BC	0B
DECW	DE	1B
DECW	HL	2B
DECW	IX	DD2B
DECW	IY	FD2B
DECW	SP	3B
DI		F3
DI	66H	ED7766
DIV	HL,(HL)	EDF4
DIV	HL,(HL+IX)	DDEDCC
DIV	HL,(HL+IY)	DDEDD4
DIV	HL,(HL+1122H)	FDEDDC2211
DIV	HL,(IX+IY)	DDEDDC
DIV	HL,(IX+55H)	DDEDF455
DIV	HL,(IX+1122H)	FDEDCC2211
DIV	HL,(IY+55H)	FDEDF455
DIV	HL,(IY+1122H)	FDEDD42211
DIV	HL,(PC+1122H)	FDEDC42211
DIV	HL,(SP+1122H)	DDEDC42211
DIV	HL,(3344H)	DDEDFC4433
DIV	HL,A	EDFC
DIV	HL,B	EDC4
DIV	HL,C	EDCC
DIV	HL,D	EDD4
DIV	HL,E	EDDC
DIV	HL,H	EDE4
DIV	HL,IXH	DDEDE4
DIV	HL,IXL	DDEDEC
DIV	HL,IYH	FDEDE4
DIV	HL,IYL	FDEDEC
DIV	HL,L	EDEC
DIV	HL,66H	FDEDFC66
DIVU	HL,(HL)	EDF5
DIVU	HL,(HL+IX)	DDEDCD
DIVU	HL,(HL+IY)	DDEDD5
DIVU	HL,(HL+1122H)	FDEDDD2211
DIVU	HL,(IX+IY)	DDEDDD
DIVU	HL,(IX+55H)	DDEDF555
DIVU	HL,(IX+1122H)	FDEDCD2211
DIVU	HL,(IY+55H)	FDEDF555
DIVU	HL,(IY+1122H)	FDEDD52211
DIVU	HL,(PC+1122H)	FDEDC52211
DIVU	HL,(SP+1122H)	DDEDC52211
DIVU	HL,(3344H)	DDEDFD4433
DIVU	HL,A	EDFD
DIVU	HL,B	EDC5
DIVU	HL,C	EDCD

SOURCE CODE	OBJECT CODE
DIVU HL,D	EDD5
DIVU HL,E	EDDD
DIVU HL,H	EDE5
DIVU HL,IXH	DDEDE5
DIVU HL,IXL	DDEDED
DIVU HL,IYH	FDEDE5
DIVU HL,IYL	FDEDED
DIVU HL,L	EDED
DIVU HL,66H	FDEDFD66
DIVUW DEHL,(HL)	DDEDCB
DIVUW DEHL,(IX+1122H)	FDEDCB2211
DIVUW DEHL,(IY+1122H)	FDEDDB2211
DIVUW DEHL,(PC+1122H)	DDEDFB2211
DIVUW DEHL,(3344H)	DDEDDB4433
DIVUW DEHL,BC	EDCB
DIVUW DEHL,DE	EDDB
DIVUW DEHL,HL	EDEB
DIVUW DEHL,IX	DDEDEB
DIVUW DEHL,IY	FDEDEB
DIVUW DEHL,SP	EDFB
DIVUW DEHL,3344H	FDEDFB4433
DIVW DEHL,(HL)	DDEDCA
DIVW DEHL,(IX+1122H)	FDEDCA2211
DIVW DEHL,(IY+1122H)	FDEDDA2211
DIVW DEHL,(PC+1122H)	DDEDFA2211
DIVW DEHL,(3344H)	DDEDDA4433
DIVW DEHL,BC	EDCA
DIVW DEHL,DE	EDDA
DIVW DEHL,HL	EDEA
DIVW DEHL,IX	DDEDEA
DIVW DEHL,IY	FDEDEA
DIVW DEHL,SP	EDFA
DIVW DEHL,3344H	FDEDFA4433
DJNZ 77H	1075
EI	FB
EI 66H	ED7F66
EPUF	ED97
EPUI	ED9F
EPUM (HL)	EDA6
EPUM (HL+IX)	ED8C
EPUM (HL+IY)	ED94
EPUM (HL+1122H)	EDBC2211
EPUM (IX+IY)	ED9C
EPUM (IX+1122H)	EDAC2211
EPUM (IY+1122H)	EDB42211
EPUM (PC+1122H)	EDA42211
EPUM (SP+1122H)	ED842211
EPUM (3344H)	EDA74433
EX (SP),HL	E3
EX (SP),IX	DDE3
EX (SP),IY	FDE3
EX A,(HL)	ED37
EX A,(HL+IX)	DDED0F
EX A,(HL+IY)	DDED17
EX A,(HL+1122H)	FDED1F2211
EX A,(IX+IY)	DDED1F
EX A,(IX+55H)	DDED3755
EX A,(IX+1122H)	FDED0F2211
EX A,(IY+55H)	FDED3755
EX A,(IY+1122H)	FDED172211

SOURCE CODE	OBJECT CODE
EX A,(PC+1122H)	FDED072211
EX A,(SP+1122H)	DDED072211
EX A,(3344H)	DDED3F4433
EX A,A	ED3F
EX A,B	ED07
EX A,C	ED0F
EX A,D	ED17
EX A,E	ED1F
EX A,H	ED27
EX A,IXH	DDED27
EX A,IXL	DDED2F
EX A,IYH	FDED27
EX A,IYL	FDED2F
EX A,L	ED2F
EX AF,AF'	08
EX DE,HL	EB
EX H,L	EDEF
EX IX,HL	DDEB
EX IY,HL	FDEB
EXTS A	ED64
EXTS HL	ED6C
EXX	D9
HALT	76
IM 0	ED46
IM 1	ED56
IM 2	ED5E
IM 3	ED4E
IN (HL+IX),(C)	DDED48
IN (HL+IY),(C)	DDED50
IN (HL+1122H),(C)	FDED582211
IN (IX+IY),(C)	DDED58
IN (IX+1122H),(C)	FDED482211
IN (IY+1122H),(C)	FDED502211
IN (PC+1122H),(C)	FDED402211
IN (SP+1122H),(C)	DDED402211
IN (3344H),(C)	DDED784433
IN A,(C)	ED78
IN A,(66H)	DB66
IN B,(C)	ED40
IN C,(C)	ED48
IN D,(C)	ED50
IN E,(C)	ED58
IN H,(C)	ED60
IN HL,(C)	EDB7
IN IXH,(C)	DDED60
IN IXL,(C)	DDED68
IN IYH,(C)	FDED60
IN IYL,(C)	FDED68
IN L,(C)	ED68
INC (HL)	34
INC (HL+IX)	DD0C
INC (HL+IY)	DD14
INC (HL+1122H)	FD1C2211
INC (IX+IY)	DD1C
INC (IX+55H)	DD3455
INC (IX+1122H)	FD0C2211
INC (IY+55H)	FD3455
INC (IY+1122H)	FD142211
INC (PC+1122H)	FD042211
INC (SP+1122H)	DD042211

SOURCE CODE		OBJECT CODE
INC	(3344H)	DD3C4433
INC	A	3C
INC	B	04
INC	BC	03
INC	C	0C
INC	D	14
INC	DE	13
INC	E	1C
INC	H	24
INC	HL	23
INC	IX	DD23
INC	IXH	DD24
INC	IXL	DD2C
INC	IY	FD23
INC	IYH	FD24
INC	IYL	FD2C
INC	L	2C
INC	SP	33
INCW	(HL)	DD03
INCW	(IX+1122H)	FD032211
INCW	(IY+1122H)	FD132211
INCW	(PC+1122H)	DD332211
INCW	(3344H)	DD134433
INCW	BC	03
INCW	DE	13
INCW	HL	23
INCW	IX	DD23
INCW	IY	FD23
INCW	SP	33
IND		EDAA
INDR		EDBA
INDRW		ED9A
INDW		ED8A
INI		EDA2
INIR		EDB2
INIRW		ED92
INIW		ED82
INW	HL,(C)	EDB7
JAF	77H	DD2874
JAR	77H	DD2074
JP	(HL)	E9
JP	(IX)	DDE9
JP	(IY)	FDE9
JP	(PC+1122H)	FDC32211
JP	C,(HL)	DDDA
JP	C,(PC+1122H)	FDDA2211
JP	C,3344H	DA4433
JP	M,(HL)	DDFA
JP	M,(PC+1122H)	FDFA2211
JP	M,3344H	FA4433
JP	NC,(HL)	DDD2
JP	NC,(PC+1122H)	FDD22211
JP	NC,3344H	D24433
JP	NZ,(HL)	DDC2
JP	NZ,(PC+1122H)	FDC22211
JP	NZ,3344H	C24433
JP	P,(HL)	DDF2
JP	P,(PC+1122H)	FDF22211
JP	P,3344H	F24433
JP	PE,(HL)	DDEA
JP	PE,(PC+1122H)	FDEA2211
JP	PE,3344H	EA4433
JP	PO,(HL)	DDE2
JP	PO,(PC+1122H)	FDE22211
JP	PO,3344H	E24433
JP	Z,(HL)	DDCA
JP	Z,(PC+1122H)	FDCA2211
JP	Z,3344H	CA4433
JP	3344H	C34433
JR	C,77H	3875
JR	NC,77H	3075
JR	NZ,77H	2075
JR	Z,77H	2875
JR	77H	1875
LD	(BC),A	02
LD	(DE),A	12
LD	(HL),A	77
LD	(HL),B	70
LD	(HL),BC	ED0E
LD	(HL),C	71
LD	(HL),D	72
LD	(HL),DE	ED1E
LD	(HL),E	73
LD	(HL),H	74
LD	(HL),HL	ED2E
LD	(HL),L	75
LD	(HL),SP	ED3E
LD	(HL),66H	3666
LD	(HL+IX),A	ED0B
LD	(HL+IX),HL	ED0D
LD	(HL+IX),IX	DDED0D
LD	(HL+IX),IY	FDED0D
LD	(HL+IX),66H	DD0E66
LD	(HL+IY),A	ED13
LD	(HL+IY),HL	ED15
LD	(HL+IY),IX	DDED15
LD	(HL+IY),IY	FDED15
LD	(HL+IY),66H	DD1666
LD	(HL+1122H),A	ED3B2211
LD	(HL+1122H),HL	ED3D2211
LD	(HL+1122H),IX	DDED3D2211
LD	(HL+1122H),IY	FDED3D2211
LD	(HL+1122H),66H	FD1E221166
LD	(IX+IY),A	ED1B
LD	(IX+IY),HL	ED1D
LD	(IX+IY),IX	DDED1D
LD	(IX+IY),IY	FDED1D
LD	(IX+IY),66H	DD1E66
LD	(IX+55H),A	DD7755
LD	(IX+55H),B	DD7055
LD	(IX+55H),BC	DDED0E55
LD	(IX+55H),C	DD7155
LD	(IX+55H),D	DD7255
LD	(IX+55H),DE	DDED1E55
LD	(IX+55H),E	DD7355
LD	(IX+55H),H	DD7455
LD	(IX+55H),HL	DDED2E55
LD	(IX+55H),L	DD7555
LD	(IX+55H),SP	DDED3E55
LD	(IX+55H),66H	DD365566

SOURCE CODE		OBJECT CODE	SOURCE CODE		OBJECT CODE
LD	(IX+1122H),A	ED2B2211	LD	A,I	ED57
LD	(IX+1122H),HL	ED2D2211	LD	A,IXH	DD7C
LD	(IX+1122H),IX	DDED2D2211	LD	A,IXL	DD7D
LD	(IX+1122H),IY	FDED2D2211	LD	A,IYH	FD7C
LD	(IX+1122H),66H	FD0E221166	LD	A,IYL	FD7D
LD	(IY+55H),A	FD7755	LD	A,L	7D
LD	(IY+55H),B	FD7055	LD	A,R	ED5F
LD	(IY+55H),BC	FDED0E55	LD	A,66H	3E66
LD	(IY+55H),C	FD7155	LD	B,(HL)	46
LD	(IY+55H),D	FD7255	LD	B,(IX+55H)	DD4655
LD	(IY+55H),DE	FDED1E55	LD	B,(IY+55H)	FD4655
LD	(IY+55H),E	FD7355	LD	B,A	47
LD	(IY+55H),H	FD7455	LD	B,B	40
LD	(IY+55H),HL	FDED2E55	LD	B,C	41
LD	(IY+55H),L	FD7555	LD	B,D	42
LD	(IY+55H),SP	FDED3E55	LD	B,E	43
LD	(IY+55H),66H	FD365566	LD	B,H	44
LD	(IY+1122H),A	ED332211	LD	B,IXH	DD44
LD	(IY+1122H),HL	ED352211	LD	B,IXL	DD45
LD	(IY+1122H),IX	DDED352211	LD	B,IYH	FD44
LD	(IY+1122H),IY	FDED352211	LD	B,IYL	FD45
LD	(IY+1122H),66H	FD16221166	LD	B,L	45
LD	(PC+1122H),A	ED232211	LD	B,66H	0666
LD	(PC+1122H),HL	ED252211	LD	BC,(HL)	ED06
LD	(PC+1122H),IX	DDED252211	LD	BC,(IX+55H)	DDED0655
LD	(PC+1122H),IY	FDED252211	LD	BC,(IY+55H)	FDED0655
LD	(PC+1122H),66H	FD06221166	LD	BC,(3344H)	ED4B4433
LD	(SP+1122H),A	ED032211	LD	BC,3344H	014433
LD	(SP+1122H),HL	ED052211	LD	C,(HL)	4E
LD	(SP+1122H),IX	DDED052211	LD	C,(IX+55H)	DD4E55
LD	(SP+1122H),IY	FDED052211	LD	C,(IY+55H)	FD4E55
LD	(SP+1122H),66H	DD06221166	LD	C,A	4F
LD	(3344H),A	324433	LD	C,B	48
LD	(3344H),BC	ED434433	LD	C,C	49
LD	(3344H),DE	ED534433	LD	C,D	4A
LD	(3344H),HL	224433	LD	C,E	4B
LD	(3344H),IX	DD224433	LD	C,H	4C
LD	(3344H),IY	FD224433	LD	C,IXH	DD4C
LD	(3344H),SP	ED734433	LD	C,IXL	DD4D
LD	(3344H),66H	DD3E443366	LD	C,IYH	FD4C
LD	A,(BC)	0A	LD	C,IYL	FD4D
LD	A,(DE)	1A	LD	C,L	4D
LD	A,(HL)	7E	LD	C,66H	0E66
LD	A,(HL+IX)	DD79	LD	D,(HL)	56
LD	A,(HL+IY)	DD7A	LD	D,(IX+55H)	DD5655
LD	A,(HL+1122H)	FD7B2211	LD	D,(IY+55H)	FD5666
LD	A,(IX+IY)	DD7B	LD	D,A	57
LD	A,(IX+55H)	DD7E55	LD	D,B	50
LD	A,(IX+1122H)	FD792211	LD	D,C	51
LD	A,(IY+55H)	FD7E55	LD	D,D	52
LD	A,(IY+1122H)	FD7A2211	LD	D,E	53
LD	A,(PC+1122H)	FD782211	LD	D,H	54
LD	A,(SP+1122H)	DD782211	LD	D,IXH	DD54
LD	A,(3344H)	3A4433	LD	D,IXL	DD55
LD	A,A	7F	LD	D,IYH	FD54
LD	A,B	78	LD	D,IYL	FD55
LD	A,C	79	LD	D,L	55
LD	A,D	7A	LD	D,66H	1666
LD	A,E	7B	LD	DE,(HL)	ED16
LD	A,H	7C	LD	DE,(IX+55H)	DDED1655

SOURCE CODE		OBJECT CODE
LD	DE,(IY+55H)	FDED1655
LD	DE,(3344H)	ED5B4433
LD	DE,3344H	114433
LD	E,(HL)	5E
LD	E,(IX+55H)	DD5E55
LD	E,(IY+55H)	FD5E55
LD	E,A	5F
LD	E,B	58
LD	E,C	59
LD	E,D	5A
LD	E,E	5B
LD	E,H	5C
LD	E,IXH	DD5C
LD	E,IXL	DD5D
LD	E,IYH	FD5C
LD	E,IYL	FD5D
LD	E,L	5D
LD	E,66H	1E66
LD	H,(HL)	66
LD	H,(IX+55H)	DD6655
LD	H,(IY+55H)	FD6655
LD	H,A	67
LD	H,B	60
LD	H,C	61
LD	H,D	62
LD	H,E	63
LD	H,H	64
LD	H,L	65
LD	H,66H	2666
LD	HL,(HL)	ED26
LD	HL,(HL+IX)	ED0C
LD	HL,(HL+IY)	ED14
LD	HL,(IX+IY)	ED1C
LD	HL,(IX+55H)	DDED2655
LD	HL,(IX+1122H)	ED2C2211
LD	HL,(IY+55H)	FDED2655
LD	HL,(IY+1122H)	ED342211
LD	HL,(PC+1122H)	ED242211
LD	HL,(SP+1122H)	ED042211
LD	HL,(3344H)	2A4433
LD	HL,3344H	214433
LD	I,A	ED47
LD	IX,(HL+IX)	DDED0C
LD	IX,(HL+IY)	DDED14
LD	IX,(HL+1122H)	DDED3C2211
LD	IX,(IX+IY)	DDED1C
LD	IX,(IX+1122H)	DDED2C2211
LD	IX,(IY+1122H)	DDED342211
LD	IX,(PC+1122H)	DDED242211
LD	IX,(SP+1122H)	DDED042211
LD	IX,(3344H)	DD2A4433
LD	IX,3344H	DD214433
LD	IXH,A	DD67
LD	IXH,B	DD60
LD	IXH,C	DD61
LD	IXH,D	DD62
LD	IXH,E	DD63
LD	IXH,IXH	DD64
LD	IXH,IXL	DD65
LD	IXH,66H	DD2666

SOURCE CODE		OBJECT CODE
LD	IXL,A	DD6F
LD	IXL,B	DD68
LD	IXL,C	DD69
LD	IXL,D	DD6A
LD	IXL,E	DD6B
LD	IXL,IXH	DD6C
LD	IXL,IXL	DD6D
LD	IXL,66H	DD2E66
LD	IY,(HL+IX)	FDED0C
LD	IY,(HL+IY)	FDED14
LD	IY,(HL+1122H)	FDED3C2211
LD	IY,(IX+IY)	FDED1C
LD	IY,(IX+1122H)	FDED2C2211
LD	IY,(IY+1122H)	FDED342211
LD	IY,(PC+1122H)	FDED242211
LD	IY,(SP+1122H)	FDED042211
LD	IY,3344H	FD214433
LD	IYH,A	FD67
LD	IYH,B	FD60
LD	IYH,C	FD61
LD	IYH,D	FD62
LD	IYH,E	FD63
LD	IYH,IYH	FD64
LD	IYH,IYL	FD65
LD	IYH,66H	FD2666
LD	IYL,A	FD6F
LD	IYL,B	FD68
LD	IYL,C	FD69
LD	IYL,D	FD6A
LD	IYL,E	FD6B
LD	IYL,IYH	FD6C
LD	IYL,IYL	FD6D
LD	IYL,66H	FD2E66
LD	L,(HL)	6E
LD	L,(IX+55H)	DD6E55
LD	L,(IY+55H)	FD6E55
LD	L,A	6F
LD	L,B	68
LD	L,C	69
LD	L,D	6A
LD	L,E	6B
LD	L,H	6C
LD	L,L	6D
LD	L,66H	2E66
LD	R,A	ED4F
LD	SP,(HL)	ED36
LD	SP,(IX+55H)	DDED3655
LD	SP,(IY+55H)	FDED3655
LD	SP,(3344H)	ED7B4433
LD	SP,HL	F9
LD	SP,IX	DDF9
LD	SP,IY	FDF9
LD	SP,3344H	314433
LDA	HL,(HL+IX)	ED0A
LDA	HL,(HL+IY)	ED12
LDA	HL,(HL+1122H)	ED3A2211
LDA	HL,(IX+IY)	ED1A
LDA	HL,(IX+1122H)	ED2A2211
LDA	HL,(IY+1122H)	ED322211
LDA	HL,(PC+1122H)	ED222211

SOURCE CODE		OBJECT CODE
LDA	HL,(SP+1122H)	ED022211
LDA	HL,(3344H)	214433
LDA	IX,(HL+IX)	DDED0A
LDA	IX,(HL+IY)	DDED12
LDA	IX,(HL+1122H)	DDED3A2211
LDA	IX,(IX+IY)	DDED1A
LDA	IX,(IX+1122H)	DDED2A2211
LDA	IX,(IY+1122H)	DDED322211
LDA	IX,(PC+1122H)	DDED222211
LDA	IX,(SP+1122H)	DDED022211
LDA	IX,(3344H)	DD214433
LDA	IY,(HL+IX)	FDED0A
LDA	IY,(HL+IY)	FDED12
LDA	IY,(HL+1122H)	FDED3A2211
LDA	IY,(IX+IY)	FDED1A
LDA	IY,(IX+1122H)	FDED2A2211
LDA	IY,(IY+1122H)	FDED322211
LDA	IY,(PC+1122H)	FDED222211
LDA	IY,(SP+1122H)	FDED022211
LDA	IY,(3344H)	FD214433
LDCTL	(C),HL	ED6E
LDCTL	(C),IX	DDED6E
LDCTL	(C),IY	FDED6E
LDCTL	HL,(C)	ED66
LDCTL	HL,USP	ED87
LDCTL	IX,(C)	DDED66
LDCTL	IX,USP	DDED87
LDCTL	IY,(C)	FDED66
LDCTL	IY,USP	FDED87
LDCTL	USP,HL	ED8F
LDCTL	USP,IX	DDED8F
LDCTL	USP,IY	FDED8F
LDD		EDA8
LDDR		EDB8
LDI		EDA0
LDIR		EDB0
LDUD	(HL),A	ED8E
LDUD	(IX+55H),A	DDED8E55
LDUD	(IY+55H),A	FDED8E55
LDUD	A,(HL)	ED86
LDUD	A,(IX+55H)	DDED8655
LDUD	A,(IY+55H)	FDED8655
LDUP	(HL),A	ED9E
LDUP	(IX+55H),A	DDED9E55
LDUP	(IY+55H),A	FDED9E55
LDUP	A,(HL)	ED96
LDUP	A,(IX+55H)	DDED9655
LDUP	A,(IY+55H)	FDED9655
LDW	(HL),BC	ED0E
LDW	(HL),DE	ED1E
LDW	(HL),HL	ED2E
LDW	(HL),SP	ED3E
LDW	(HL),3344H	DD014433
LDW	(HL+IX),HL	ED0D
LDW	(HL+IX),IX	DDED0D
LDW	(HL+IX),IY	FDED0D
LDW	(HL+IY),HL	ED15
LDW	(HL+IY),IX	DDED15
LDW	(HL+IY),IY	FDED15
LDW	(HL+1122H),HL	ED3D2211

SOURCE CODE		OBJECT CODE
LDW	(HL+1122H),IX	DDED3D2211
LDW	(HL+1122H),IY	FDED3D2211
LDW	(IX+IY),HL	ED1D
LDW	(IX+IY),IX	DDED1D
LDW	(IX+IY),IY	FDED1D
LDW	(IX+55H),BC	DDED0E55
LDW	(IX+55H),DE	DDED1E55
LDW	(IX+55H),HL	DDED2E55
LDW	(IX+55H),SP	DDED3E55
LDW	(IX+1122H),HL	ED2D2211
LDW	(IX+1122H),IX	DDED2D2211
LDW	(IX+1122H),IY	FDED2D2211
LDW	(IY+55H),BC	FDED0E55
LDW	(IY+55H),DE	FDED1E55
LDW	(IY+55H),HL	FDED2E55
LDW	(IY+55H),SP	FDED3E55
LDW	(IY+1122H),HL	ED352211
LDW	(IY+1122H),IX	DDED352211
LDW	(IY+1122H),IY	FDED352211
LDW	(PC+1122H),HL	ED252211
LDW	(PC+1122H),IX	DDED252211
LDW	(PC+1122H),IY	FDED252211
LDW	(PC+1122H),3344H	DD3122114433
LDW	(SP+1122H),HL	ED052211
LDW	(SP+1122H),IX	DDED052211
LDW	(SP+1122H),IY	FDED052211
LDW	(3344H),BC	ED434433
LDW	(3344H),DE	ED534433
LDW	(3344H),HL	224433
LDW	(3344H),IX	DD224433
LDW	(3344H),IY	FD224433
LDW	(3344H),SP	ED734433
LDW	(3344H),8899H	DD1144339988
LDW	BC,(HL)	ED06
LDW	BC,(IX+55H)	DDED0655
LDW	BC,(IY+55H)	FDED0655
LDW	BC,(3344H)	ED4B4433
LDW	BC,3344H	014433
LDW	DE,(HL)	ED16
LDW	DE,(IX+55H)	DDED1655
LDW	DE,(IY+55H)	FDED1655
LDW	DE,(3344H)	ED5B4433
LDW	DE,3344H	114433
LDW	HL,(HL)	ED26
LDW	HL,(HL+IX)	ED0C
LDW	HL,(HL+IY)	ED14
LDW	HL,(HL+1122H)	ED3C2211
LDW	HL,(IX+IY)	ED1C
LDW	HL,(IX+55H)	DDED2655
LDW	HL,(IX+1122H)	ED2C2211
LDW	HL,(IY+55H)	FDED2655
LDW	HL,(IY+1122H)	ED342211
LDW	HL,(PC+1122H)	ED242211
LDW	HL,(SP+1122H)	ED042211
LDW	HL,(3344H)	2A4433
LDW	HL,3344H	214433
LDW	IX,(HL+IX)	DDED0C
LDW	IX,(HL+IY)	DDED14
LDW	IX,(HL+1122H)	DDED3C2211
LDW	IX,(IX+IY)	DDED1C

SOURCE CODE	OBJECT CODE
LDW IX,(IX+1122H)	DDED2C2211
LDW IX,(IY+1122H)	DDED342211
LDW IX,(PC+1122H)	DDED242211
LDW IX,(SP+1122H)	DDED042211
LDW IX,(3344H)	DD2A4433
LDW IX,3344H	DD214433
LDW IY,(HL+IX)	FDED0C
LDW IY,(HL+IY)	FDED14
LDW IY,(HL+1122H)	FDED3C2211
LDW IY,(IX+IY)	FDED1C
LDW IY,(IX+1122H)	FDED2C2211
LDW IY,(IY+1122H)	FDED342211
LDW IY,(PC+1122H)	FDED242211
LDW IY,(SP+1122H)	FDED042211
LDW IY,(3344H)	FD2A4433
LDW IY,3344H	FD214433
LDW SP,(HL)	ED36
LDW SP,(IX+55H)	DDED3655
LDW SP,(IY+55H)	FDED3655
LDW SP,(3344H)	ED7B4433
LDW SP,HL	F9
LDW SP,IX	DDF9
LDW SP,IY	FDF9
LDW SP,3344H	314433
MEPU (HL)	EDAE
MEPU (HL+IX)	ED8D
MEPU (HL+IY)	ED95
MEPU (HL+1122H)	EDBD2211
MEPU (IX+IY)	ED9D
MEPU (IX+1122H)	EDAD2211
MEPU (IY+1122H)	EDB52211
MEPU (PC+1122H)	EDA52211
MEPU (SP+1122H)	ED852211
MEPU (3344H)	EDAF4433
MULT A,(HL)	EDF0
MULT A,(HL+IX)	DDEDC8
MULT A,(HL+IY)	DDEDD0
MULT A,(HL+1122H)	FDEDD82211
MULT A,(IX+IY)	DDEDD8
MULT A,(IX+55H)	DDEDF055
MULT A,(IX+1122H)	FDEDC82211
MULT A,(IY+55H)	FDEDF055
MULT A,(IY+1122H)	FDEDD02211
MULT A,(PC+1122H)	FDEDC02211
MULT A,(SP+1122H)	DDEDC02211
MULT A,(3344H)	DDEDF84433
MULT A,A	EDF8
MULT A,B	EDC0
MULT A,C	EDC8
MULT A,D	EDD0
MULT A,E	EDD8
MULT A,H	EDE0
MULT A,IXH	DDEDE0
MULT A,IXL	DDEDE8
MULT A,IYH	FDEDE0
MULT A,IYL	FDEDE8
MULT A,L	EDE8
MULT A,66H	FDEDF866
MULTU A,(HL)	EDF1
MULTU A,(HL+IX)	DDEDC9

SOURCE CODE	OBJECT CODE
MULTU A,(HL+IY)	DDEDD1
MULTU A,(HL+1122H)	FDEDD92211
MULTU A,(IX+IY)	DDEDD9
MULTU A,(IX+55H)	DDEDF155
MULTU A,(IX+1122H)	FDEDC92211
MULTU A,(IY+55H)	FDEDF155
MULTU A,(IY+1122H)	FDEDD12211
MULTU A,(PC+1122H)	FDEDC12211
MULTU A,(SP+1122H)	DDEDC12211
MULTU A,(3344H)	DDEDF94433
MULTU A,A	EDF9
MULTU A,B	EDC1
MULTU A,C	EDC9
MULTU A,D	EDD1
MULTU A,E	EDD9
MULTU A,H	EDE1
MULTU A,IXH	DDEDE1
MULTU A,IXL	DDEDE9
MULTU A,IYH	FDEDE1
MULTU A,IYL	FDEDE9
MULTU A,L	EDE9
MULTU A,66H	FDEDF966
MULTUW HL,(HL)	DDEDC3
MULTUW HL,(IX+1122H)	FDEDC32211
MULTUW HL,(IY+1122H)	FDEDD32211
MULTUW HL,(PC+1122H)	DDEDF32211
MULTUW HL,(3344H)	DDEDD34433
MULTUW HL,BC	EDC3
MULTUW HL,DE	EDD3
MULTUW HL,HL	EDE3
MULTUW HL,IX	DDEDE3
MULTUW HL,IY	FDEDE3
MULTUW HL,SP	EDF3
MULTUW HL,3344H	FDEDF34433
MULTW HL,(HL)	DDEDC2
MULTW HL,(IX+1122H)	FDEDC22211
MULTW HL,(IY+1122H)	FDEDD22211
MULTW HL,(PC+1122H)	DDEDF22211
MULTW HL,(3344H)	DDEDD24433
MULTW HL,BC	EDC2
MULTW HL,DE	EDD2
MULTW HL,HL	EDE2
MULTW HL,IX	DDEDE2
MULTW HL,IY	FDEDE2
MULTW HL,SP	EDF2
MULTW HL,3344H	FDEDF24433
NEG A	ED44
NEG HL	ED4C
NOP	00
OR A,(HL)	B6
OR A,(HL+IX)	DDB1
OR A,(HL+IY)	DDB2
OR A,(HL+1122H)	FDB32211
OR A,(IX+IY)	DDB3
OR A,(IX+55H)	DDB655
OR A,(IX+1122H)	FDB12211
OR A,(IY+55H)	FDB655
OR A,(IY+1122H)	FDB22211
OR A,(PC+1122H)	FDB02211
OR A,(SP+1122H)	DDB02211

SOURCE CODE	OBJECT CODE
OR A,(3344H)	DDB74433
OR A,A	B7
OR A,B	B0
OR A,C	B1
OR A,D	B2
OR A,E	B3
OR A,H	B4
OR A,IXH	DDB4
OR A,IXL	DDB5
OR A,IYH	FDB4
OR A,IYL	FDB5
OR A,L	B5
OR A,66H	F666
OTDR	EDBB
OTDRW	ED9B
OTIR	EDB3
OTIRW	ED93
OUT (C),(HL+IX)	DDED49
OUT (C),(HL+IY)	DDED51
OUT (C),(HL+1122H)	FDED592211
OUT (C),(IX+IY)	DDED59
OUT (C),(IX+1122H)	FDED492211
OUT (C),(IY+1122H)	FDED512211
OUT (C),(PC+1122H)	FDED412211
OUT (C),(SP+1122H)	DDED412211
OUT (C),(3344H)	DDED794433
OUT (C),A	ED79
OUT (C),B	ED41
OUT (C),C	ED49
OUT (C),D	ED51
OUT (C),E	ED59
OUT (C),H	ED61
OUT (C),HL	EDBF
OUT (C),IXH	DDED61
OUT (C),IXL	DDED69
OUT (C),IYH	FDED61
OUT (C),IYL	FDED69
OUT (C),L	ED69
OUT (66H),A	D366
OUTD	EDAB
OUTDW	ED8B
OUTI	EDA3
OUTIW	ED83
OUTW (C),HL	EDBF
PCACHE	ED65
POP (HL)	DDC1
POP (PC+1122H)	DDF12211
POP (3344H)	DDD14433
POP AF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DDE1
POP IY	FDE1
PUSH (HL)	DDC5
PUSH (PC+1122H)	DDF52211
PUSH (3344H)	DDD54433
PUSH AF	F5
PUSH BC	C5
PUSH DE	D5

SOURCE CODE	OBJECT CODE
PUSH HL	E5
PUSH IX	DDE5
PUSH IY	FDE5
PUSH 3344H	FDF54433
RES 0,(HL)	CB86
RES 0,(IX+55H)	DDCB5586
RES 0,(IY+55H)	FDCB5586
RES 0,A	CB87
RES 0,B	CB80
RES 0,C	CB81
RES 0,D	CB82
RES 0,E	CB83
RES 0,H	CB84
RES 0,L	CB85
RES 1,(HL)	CB8E
RES 1,(IX+55H)	DDCB558E
RES 1,(IY+55H)	FDCB558E
RES 1,A	CB8F
RES 1,B	CB88
RES 1,C	CB89
RES 1,D	CB8A
RES 1,E	CB8B
RES 1,H	CB8C
RES 1,L	CB8D
RES 2,(HL)	CB96
RES 2,(IX+55H)	DDCB5596
RES 2,(IY+55H)	FDCB5596
RES 2,A	CB97
RES 2,B	CB90
RES 2,C	CB91
RES 2,D	CB92
RES 2,E	CB93
RES 2,H	CB94
RES 2,L	CB95
RES 3,(HL)	CB9E
RES 3,(IX+55H)	DDCB559E
RES 3,(IY+55H)	FDCB559E
RES 3,A	CB9F
RES 3,B	CB98
RES 3,C	CB99
RES 3,D	CB9A
RES 3,E	CB9B
RES 3,H	CB9C
RES 3,L	CB9D
RES 4,(HL)	CBA6
RES 4,(IX+55H)	DDCB55A6
RES 4,(IY+55H)	FDCB55A6
RES 4,A	CBA7
RES 4,B	CBA0
RES 4,C	CBA1
RES 4,D	CBA2
RES 4,E	CBA3
RES 4,H	CBA4
RES 4,L	CBA5
RES 5,(HL)	CBAE
RES 5,(IX+55H)	DDCB55AE
RES 5,(IY+55H)	FDCB55AE
RES 5,A	CBAF
RES 5,B	CBA8
RES 5,C	CBA9

SOURCE CODE		OBJECT CODE
RES	5,D	CBAA
RES	5,E	CBAB
RES	5,H	CBAC
RES	5,L	CBAD
RES	6,(HL)	CBB6
RES	6,(IX+55H)	DDCB55B6
RES	6,(IY+55H)	FDCB55B6
RES	6,A	CBB7
RES	6,B	CBB0
RES	6,C	CBB1
RES	6,D	CBB2
RES	6,E	CBB3
RES	6,H	CBB4
RES	6,L	CBB5
RES	7,(HL)	CBBE
RES	7,(IX+55H)	DDCB55BE
RES	7,(IY+55H)	FDCB55BE
RES	7,A	CBBF
RES	7,B	CBB8
RES	7,C	CBB9
RES	7,D	CBBA
RES	7,E	CBBB
RES	7,H	CBBC
RES	7,L	CBBD
RET		C9
RET	C	D8
RET	M	F8
RET	NC	D0
RET	NZ	C0
RET	P	F0
RET	PE	E8
RET	PO	E0
RET	Z	C8
RETI		ED4D
RETIL		ED55
RETN		ED45
RL	(HL)	CB16
RL	(IX+55H)	DDCB5516
RL	(IY+55H)	FDCB5516
RL	A	CB17
RL	B	CB10
RL	C	CB11
RL	D	CB12
RL	E	CB13
RL	H	CB14
RL	L	CB15
RLA		17
RLC	(HL)	CB06
RLC	(IX+55H)	DDCB5506
RLC	(IY+55H)	FDCB5506
RLC	A	CB07
RLC	B	CB00
RLC	C	CB01
RLC	D	CB02
RLC	E	CB03
RLC	H	CB04
RLC	L	CB05
RLCA		07
RLD		ED6F
RR	(HL)	CB1E

SOURCE CODE		OBJECT CODE
RR	(IX+55H)	DDCB551E
RR	(IY+55H)	FDCB551E
RR	A	CB1F
RR	B	CB18
RR	C	CB19
RR	D	CB1A
RR	E	CB1B
RR	H	CB1C
RR	L	CB1D
RRA		1F
RRC	(HL)	CB0E
RRC	(IX+55H)	DDCB550E
RRC	(IY+55H)	FDCB550E
RRC	A	CB0F
RRC	B	CB08
RRC	C	CB09
RRC	D	CB0A
RRC	E	CB0B
RRC	H	CB0C
RRC	L	CB0D
RRCA		0F
RRD		ED67
RST	00H	C7
RST	08H	CF
RST	10H	D7
RST	18H	DF
RST	20H	E7
RST	28H	EF
RST	30H	F7
RST	38H	FF
SBC	A,(HL)	9E
SBC	A,(HL+IX)	DD99
SBC	A,(HL+IY)	DD9A
SBC	A,(HL+1122H)	FD9B2211
SBC	A,(IX+IY)	DD9B
SBC	A,(IX+55H)	DD9E55
SBC	A,(IX+1122H)	FD992211
SBC	A,(IY+55H)	FD9E55
SBC	A,(IY+1122H)	FD9A2211
SBC	A,(PC+1122H)	FD982211
SBC	A,(SP+1122H)	DD982211
SBC	A,(3344H)	DD9F4433
SBC	A,A	9F
SBC	A,B	98
SBC	A,C	99
SBC	A,D	9A
SBC	A,E	9B
SBC	A,H	9C
SBC	A,IXH	DD9C
SBC	A,IXL	DD9D
SBC	A,IYH	FD9C
SBC	A,IYL	FD9D
SBC	A,L	9D
SBC	A,66H	DE66
SBC	HL,BC	ED42
SBC	HL,DE	ED52
SBC	HL,HL	ED62
SBC	HL,SP	ED72
SBC	IX,BC	DDED42
SBC	IX,DE	DDED52

SOURCE CODE		OBJECT CODE	SOURCE CODE		OBJECT CODE
SBC	IX,IX	DDED62	SET	5,(IY+55H)	FDCB55EE
SBC	IX,SP	DDED72	SET	5,A	CBEF
SBC	IY,BC	FDED42	SET	5,B	CBE8
SBC	IY,DE	FDED52	SET	5,C	CBE9
SBC	IY,IY	FDED62	SET	5,D	CBEA
SBC	IY,SP	FDED72	SET	5,E	CBEB
SC	3344H	ED714433	SET	5,H	CBEC
SCF		37	SET	5,L	CBED
SET	0,(HL)	CBC6	SET	6,(HL)	CBF6
SET	0,(IX+55H)	DDCB55C6	SET	6,(IX+55H)	DDCB55F6
SET	0,(IY+55H)	FDCB55C6	SET	6,(IY+55H)	FDCB55F6
SET	0,A	CBC7	SET	6,A	CBF7
SET	0,B	CBC0	SET	6,B	CBF0
SET	0,C	CBC1	SET	6,C	CBF1
SET	0,D	CBC2	SET	6,D	CBF2
SET	0,E	CBC3	SET	6,E	CBF3
SET	0,H	CBC4	SET	6,H	CBF4
SET	0,L	CBC5	SET	6,L	CBF5
SET	1,(HL)	CBCE	SET	7,(HL)	CBFE
SET	1,(IX+55H)	DDCB55CE	SET	7,(IX+55H)	DDCB55FE
SET	1,(IY+55H)	FDCB55CE	SET	7,(IY+55H)	FDCB55FE
SET	1,A	CBCF	SET	7,A	CBFF
SET	1,B	CBC8	SET	7,B	CBF8
SET	1,C	CBC9	SET	7,C	CBF9
SET	1,D	CBCA	SET	7,D	CBFA
SET	1,E	CBCB	SET	7,E	CBFB
SET	1,H	CBCC	SET	7,H	CBFC
SET	1,L	CBCD	SET	7,L	CBFD
SET	2,(HL)	CBD6	SLA	(HL)	CB26
SET	2,(IX+55H)	DDCB55D6	SLA	(IX+55H)	DDCB5526
SET	2,(IY+55H)	FDCB55D6	SLA	(IY+55H)	FDCB5526
SET	2,A	CBD7	SLA	A	CB27
SET	2,B	CBD0	SLA	B	CB20
SET	2,C	CBD1	SLA	C	CB21
SET	2,D	CBD2	SLA	D	CB22
SET	2,E	CBD3	SLA	E	CB23
SET	2,H	CBD4	SLA	H	CB24
SET	2,L	CBD5	SLA	L	CB25
SET	3,(HL)	CBDE	SRA	(HL)	CB2E
SET	3,(IX+55H)	DDCB55DE	SRA	(IX+55H)	DDCB552E
SET	3,(IY+55H)	FDCB55DE	SRA	(IY+55H)	FDCB552E
SET	3,A	CBDF	SRA	A	CB2F
SET	3,B	CBD8	SRA	B	CB28
SET	3,C	CBD9	SRA	C	CB29
SET	3,D	CBDA	SRA	D	CB2A
SET	3,E	CBDB	SRA	E	CB2B
SET	3,H	CBDC	SRA	H	CB2C
SET	3,L	CBDD	SRA	L	CB2D
SET	4,(HL)	CBE6	SRL	(HL)	CB3E
SET	4,(IX+55H)	DDCB55E6	SRL	(IX+55H)	DDCB553E
SET	4,(IY+55H)	FDCB55E6	SRL	(IY+55H)	FDCB553E
SET	4,A	CBE7	SRL	A	CB3F
SET	4,B	CBE0	SRL	B	CB38
SET	4,C	CBE1	SRL	C	CB39
SET	4,D	CBE2	SRL	D	CB3A
SET	4,E	CBE3	SRL	E	CB3B
SET	4,H	CBE4	SRL	H	CB3C
SET	4,L	CBE5	SRL	L	CB3D
SET	5,(HL)	CBEE	SUB	A,(HL)	96
SET	5,(IX+55H)	DDCB55EE	SUB	A,(HL+IX)	DD91

SOURCE CODE		OBJECT CODE
SUB	A,(HL+IY)	DD92
SUB	A,(HL+1122H)	FD932211
SUB	A,(IX+IY)	DD93
SUB	A,(IX+55H)	DD9655
SUB	A,(IX+1122H)	FD912211
SUB	A,(IY+55H)	FD9655
SUB	A,(IY+1122H)	FD922211
SUB	A,(PC+1122H)	FD902211
SUB	A,(SP+1122H)	DD902211
SUB	A,(3344H)	DD974433
SUB	A,A	97
SUB	A,B	90
SUB	A,C	91
SUB	A,D	92
SUB	A,E	93
SUB	A,H	94
SUB	A,IXH	DD94
SUB	A,IXL	DD95
SUB	A,IYH	FD94
SUB	A,IYL	FD95
SUB	A,L	95
SUB	A,66H	D666
SUBW	HL,(HL)	DDEDCE
SUBW	HL,(IX+1122H)	FDEDCE2211
SUBW	HL,(IY+1122H)	FDEDDE2211
SUBW	HL,(PC+1122H)	DDEDFE2211
SUBW	HL,(3344H)	DDEDDE4433
SUBW	HL,BC	EDCE
SUBW	HL,DE	EDDE
SUBW	HL,HL	EDEE
SUBW	HL,IX	DDEDEE
SUBW	HL,IY	FDEDEE
SUBW	HL,SP	EDFE
SUBW	HL,3344H	FDEDFE4433
TSET	(HL)	CB36
TSET	(IX+55H)	DDCB5536
TSET	(IY+55H)	FDCB5536
TSET	A	CB37
TSET	B	CB30
TSET	C	CB31
TSET	D	CB32
TSET	E	CB33
TSET	H	CB34
TSET	L	CB35
TSTI	(C)	ED70
XOR	A,(HL)	AE
XOR	A,(HL+IX)	DDA9
XOR	A,(HL+IY)	DDAA
XOR	A,(HL+1122H)	FDAB2211
XOR	A,(IX+IY)	DDAB
XOR	A,(IX+55H)	DDAE55
XOR	A,(IX+1122H)	FDA92211
XOR	A,(IY+55H)	FDAE55
XOR	A,(IY+1122H)	FDAA2211
XOR	A,(PC+1122H)	FDA82211
XOR	A,(SP+1122H)	DDA82211
XOR	A,(3344H)	DDAF4433
XOR	A,A	AF
XOR	A,B	A8
XOR	A,C	A9
XOR	A,D	AA
XOR	A,E	AB
XOR	A,H	AC
XOR	A,IXH	DDAC
XOR	A,IXL	DDAD
XOR	A,IYH	FDAC
XOR	A,IYL	FDAD
XOR	A,L	AD
XOR	A,66H	EE66

Appendix D.
Instructions in Numeric Order

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
00	NOP		29	ADD	HL,HL
014433	LD	BC,3344H	2A4433	LD	HL,(3344H)
014433	LDW	BC,3344H	2A4433	LDW	HL,(3344H)
02	LD	(BC),A	2B	DEC	HL
03	INCW	BC	2B	DECW	HL
03	INC	BC	2C	INC	L
04	INC	B	2D	DEC	L
05	DEC	B	2E66	LD	L,66H
0666	LD	B,66H	2F	CPL	
07	RLCA		3075	JR	NC,77H
08	EX	AF,AF'	314433	LD	SP,3344H
09	ADD	HL,BC	314433	LDW	SP,3344H
0A	LD	A,(BC)	324433	LD	(3344H),A
0B	DEC	BC	33	INC	SP
0B	DECW	BC	33	INCW	SP
0C	INC	C	34	INC	(HL)
0D	DEC	C	35	DEC	(HL)
0E66	LD	C,66H	3666	LD	(HL),66H
0F	RRCA		37	SCF	
1075	DJNZ	77H	3875	JR	C,77H
114433	LD	DE,3344H	39	ADD	HL,SP
114433	LDW	DE,3344H	3A4433	LD	A,(3344H)
12	LD	(DE),A	3B	DEC	SP
13	INC	DE	3B	DECW	SP
13	INCW	DE	3C	INC	A
14	INC	D	3D	DEC	A
15	DEC	D	3E66	LD	A,66H
1666	LD	D,66H	3F	CCF	
17	RLA		40	LD	B,B
1875	JR	77H	41	LD	B,C
19	ADD	HL,DE	42	LD	B,D
1A	LD	A,(DE)	43	LD	B,E
1B	DEC	DE	44	LD	B,H
1B	DECW	DE	45	LD	B,L
1C	INC	E	46	LD	B,(HL)
1D	DEC	E	47	LD	B,A
1E66	LD	E,66H	48	LD	C,B
1F	RRA		49	LD	C,C
2075	JR	NZ,77H	4A	LD	C,D
214433	LD	HL,3344H	4B	LD	C,E
214433	LDA	HL,(3344H)	4C	LD	C,H
214433	LDW	HL,3344H	4D	LD	C,L
224433	LD	(3344H),HL	4E	LD	C,(HL)
224433	LDW	(3344H),HL	4F	LD	C,A
23	INCW	HL	50	LD	D,B
23	INC	HL	51	LD	D,C
24	INC	H	52	LD	D,D
25	DEC	H	53	LD	D,E
2666	LD	H,66H	54	LD	D,H
27	DAA		55	LD	D,L
2875	JR	Z,77H	56	LD	D,(HL)

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
57	LD	D,A	93	SUB	A,E
58	LD	E,B	94	SUB	A,H
59	LD	E,C	95	SUB	A,L
5A	LD	E,D	96	SUB	A,(HL)
5B	LD	E,E	97	SUB	A,A
5C	LD	E,H	98	SBC	A,B
5D	LD	E,L	99	SBC	A,C
5E	LD	E,(HL)	9A	SBC	A,D
5F	LD	E,A	9B	SBC	A,E
60	LD	H,B	9C	SBC	A,H
61	LD	H,C	9D	SBC	A,L
62	LD	H,D	9E	SBC	A,(HL)
63	LD	H,E	9F	SBC	A,A
64	LD	H,H	A0	AND	A,B
65	LD	H,L	A1	AND	A,C
66	LD	H,(HL)	A2	AND	A,D
67	LD	H,A	A3	AND	A,E
68	LD	L,B	A4	AND	A,H
69	LD	L,C	A5	AND	A,L
6A	LD	L,D	A6	AND	A,(HL)
6B	LD	L,E	A7	AND	A,A
6C	LD	L,H	A8	XOR	A,B
6D	LD	L,L	A9	XOR	A,C
6E	LD	L,(HL)	AA	XOR	A,D
6F	LD	L,A	AB	XOR	A,E
70	LD	(HL),B	AC	XOR	A,H
71	LD	(HL),C	AD	XOR	A,L
72	LD	(HL),D	AE	XOR	A,(HL)
73	LD	(HL),E	AF	XOR	A,A
74	LD	(HL),H	B0	OR	A,B
75	LD	(HL),L	B1	OR	A,C
76	HALT		B2	OR	A,D
77	LD	(HL),A	B3	OR	A,E
78	LD	A,B	B4	OR	A,H
79	LD	A,C	B5	OR	A,L
7A	LD	A,D	B6	OR	A,(HL)
7B	LD	A,E	B7	OR	A,A
7C	LD	A,H	B8	CP	A,B
7D	LD	A,L	B9	CP	A,C
7E	LD	A,(HL)	BA	CP	A,D
7F	LD	A,A	BB	CP	A,E
80	ADD	A,B	BC	CP	A,H
81	ADD	A,C	BD	CP	A,L
82	ADD	A,D	BE	CP	A,(HL)
83	ADD	A,E	BF	CP	A,A
84	ADD	A,H	C0	RET	NZ
85	ADD	A,L	C1	POP	BC
86	ADD	A,(HL)	C24433	JP	NZ,3344H
87	ADD	A,A	C34433	JP	3344H
88	ADC	A,B	C44433	CALL	NZ,3344H
89	ADC	A,C	C5	PUSH	BC
8A	ADC	A,D	C666	ADD	A,66H
8B	ADC	A,E	C7	RST	00H
8C	ADC	A,H	C8	RET	Z
8D	ADC	A,L	C9	RET	
8E	ADC	A,(HL)	CA4433	JP	Z,3344H
8F	ADC	A,A	CB00	RLC	B
90	SUB	A,B	CB01	RLC	C
91	SUB	A,C	CB02	RLC	D
92	SUB	A,D	CB03	RLC	E

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
CB04	RLC	H	CB40	BIT	0,B
CB05	RLC	L	CB41	BIT	0,C
CB06	RLC	(HL)	CB42	BIT	0,D
CB07	RLC	A	CB43	BIT	0,E
CB08	RRC	B	CB44	BIT	0,H
CB09	RRC	C	CB45	BIT	0,L
CB0A	RRC	D	CB46	BIT	0,(HL)
CB0B	RRC	E	CB47	BIT	0,A
CB0C	RRC	H	CB48	BIT	1,B
CB0D	RRC	L	CB49	BIT	1,C
CB0E	RRC	(HL)	CB4A	BIT	1,D
CB0F	RRC	A	CB4B	BIT	1,E
CB10	RL	B	CB4C	BIT	1,H
CB11	RL	C	CB4D	BIT	1,L
CB12	RL	D	CB4E	BIT	1,(HL)
CB13	RL	E	CB4F	BIT	1,A
CB14	RL	H	CB50	BIT	2,B
CB15	RL	L	CB51	BIT	2,C
CB16	RL	(HL)	CB52	BIT	2,D
CB17	RL	A	CB53	BIT	2,E
CB18	RR	B	CB54	BIT	2,H
CB19	RR	C	CB55	BIT	2,L
CB1A	RR	D	CB56	BIT	2,(HL)
CB1B	RR	E	CB57	BIT	2,A
CB1C	RR	H	CB58	BIT	3,B
CB1D	RR	L	CB59	BIT	3,C
CB1E	RR	(HL)	CB5A	BIT	3,D
CB1F	RR	A	CB5B	BIT	3,E
CB20	SLA	B	CB5C	BIT	3,H
CB21	SLA	C	CB5D	BIT	3,L
CB22	SLA	D	CB5E	BIT	3,(HL)
CB23	SLA	E	CB5F	BIT	3,A
CB24	SLA	H	CB60	BIT	4,B
CB25	SLA	L	CB61	BIT	4,C
CB26	SLA	(HL)	CB62	BIT	4,D
CB27	SLA	A	CB63	BIT	4,E
CB28	SRA	B	CB64	BIT	4,H
CB29	SRA	C	CB65	BIT	4,L
CB2A	SRA	D	CB66	BIT	4,(HL)
CB2B	SRA	E	CB67	BIT	4,A
CB2C	SRA	H	CB68	BIT	5,B
CB2D	SRA	L	CB69	BIT	5,C
CB2E	SRA	(HL)	CB6A	BIT	5,D
CB2F	SRA	A	CB6B	BIT	5,E
CB30	TSET	B	CB6C	BIT	5,H
CB31	TSET	C	CB6D	BIT	5,L
CB32	TSET	D	CB6E	BIT	5,(HL)
CB33	TSET	E	CB6F	BIT	5,A
CB34	TSET	H	CB70	BIT	6,B
CB35	TSET	L	CB71	BIT	6,C
CB36	TSET	(HL)	CB72	BIT	6,D
CB37	TSET	A	CB73	BIT	6,E
CB38	SRL	B	CB74	BIT	6,H
CB39	SRL	C	CB75	BIT	6,L
CB3A	SRL	D	CB76	BIT	6,(HL)
CB3B	SRL	E	CB77	BIT	6,A
CB3C	SRL	H	CB78	BIT	7,B
CB3D	SRL	L	CB79	BIT	7,C
CB3E	SRL	(HL)	CB7A	BIT	7,D
CB3F	SRL	A	CB7B	BIT	7,E

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
CB7C	BIT	7,H	CBB8	RES	7,B
CB7D	BIT	7,L	CBB9	RES	7,C
CB7E	BIT	7,(HL)	CBBA	RES	7,D
CB7F	BIT	7,A	CBBB	RES	7,E
CB80	RES	0,B	CBBC	RES	7,H
CB81	RES	0,C	CBBD	RES	7,L
CB82	RES	0,D	CBBE	RES	7,(HL)
CB83	RES	0,E	CBBF	RES	7,A
CB84	RES	0,H	CBC0	SET	0,B
CB85	RES	0,L	CBC1	SET	0,C
CB86	RES	0,(HL)	CBC2	SET	0,D
CB87	RES	0,A	CBC3	SET	0,E
CB88	RES	1,B	CBC4	SET	0,H
CB89	RES	1,C	CBC5	SET	0,L
CB8A	RES	1,D	CBC6	SET	0,(HL)
CB8B	RES	1,E	CBC7	SET	0,A
CB8C	RES	1,H	CBC8	SET	1,B
CB8D	RES	1,L	CBC9	SET	1,C
CB8E	RES	1,(HL)	CBCA	SET	1,D
CB8F	RES	1,A	CBCB	SET	1,E
CB90	RES	2,B	CBCC	SET	1,H
CB91	RES	2,C	CBCD	SET	1,L
CB92	RES	2,D	CBCE	SET	1,(HL)
CB93	RES	2,E	CBCF	SET	1,A
CB94	RES	2,H	CBD0	SET	2,B
CB95	RES	2,L	CBD1	SET	2,C
CB96	RES	2,(HL)	CBD2	SET	2,D
CB97	RES	2,A	CBD3	SET	2,E
CB98	RES	3,B	CBD4	SET	2,H
CB99	RES	3,C	CBD5	SET	2,L
CB9A	RES	3,D	CBD6	SET	2,(HL)
CB9B	RES	3,E	CBD7	SET	2,A
CB9C	RES	3,H	CBD8	SET	3,B
CB9D	RES	3,L	CBD9	SET	3,C
CB9E	RES	3,(HL)	CBDA	SET	3,D
CB9F	RES	3,A	CBDB	SET	3,E
CBA0	RES	4,B	CBDC	SET	3,H
CBA1	RES	4,C	CBDD	SET	3,L
CBA2	RES	4,D	CBDE	SET	3,(HL)
CBA3	RES	4,E	CBDF	SET	3,A
CBA4	RES	4,H	CBE0	SET	4,B
CBA5	RES	4,L	CBE1	SET	4,C
CBA6	RES	4,(HL)	CBE2	SET	4,D
CBA7	RES	4,A	CBE3	SET	4,E
CBA8	RES	5,B	CBE4	SET	4,H
CBA9	RES	5,C	CBE5	SET	4,L
CBAA	RES	5,D	CBE6	SET	4,(HL)
CBAB	RES	5,E	CBE7	SET	4,A
CBAC	RES	5,H	CBE8	SET	5,B
CBAD	RES	5,L	CBE9	SET	5,C
CBAE	RES	5,(HL)	CBEA	SET	5,D
CBAF	RES	5,A	CBEB	SET	5,E
CBB0	RES	6,B	CBEC	SET	5,H
CBB1	RES	6,C	CBED	SET	5,L
CBB2	RES	6,D	CBEE	SET	5,(HL)
CBB3	RES	6,E	CBEF	SET	5,A
CBB4	RES	6,H	CBF0	SET	6,B
CBB5	RES	6,L	CBF1	SET	6,C
CBB6	RES	6,(HL)	CBF2	SET	6,D
CBB7	RES	6,A	CBF3	SET	6,E

OBJECT CODE	SOURCE CODE	
CBF4	SET	6,H
CBF5	SET	6,L
CBF6	SET	6,(HL)
CBF7	SET	6,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CBFE	SET	7,(HL)
CBFF	SET	7,A
CC4433	CALL	Z,3344H
CD4433	CALL	3344H
CE66	ADC	A,66H
CF	RST	08H
D0	RET	NC
D1	POP	DE
D24433	JP	NC,3344H
D366	OUT	(66H),A
D44433	CALL	NC,3344H
D5	PUSH	DE
D666	SUB	A,66H
D7	RST	10H
D8	RET	C
D9	EXX	
DA4433	JP	C,3344H
DB66	IN	A,(66H)
DC4433	CALL	C,3344H
DD014433	LDW	(HL),3344H
DD03	INCW	(HL)
DD042211	INC	(SP+1122H)
DD052211	DEC	(SP+1122H)
DD06221166	LD	(SP+1122H),66H
DD09	ADD	IX,BC
DD0B	DECW	(HL)
DD0C	INC	(HL+IX)
DD0D	DEC	(HL+IX)
DD0E66	LD	(HL+IX),66H
DD1144339988	LDW	(3344H),8899H
DD134433	INCW	(3344H)
DD14	INC	(HL+IY)
DD15	DEC	(HL+IY)
DD1666	LD	(HL+IY),66H
DD19	ADD	IX,DE
DD1B4433	DECW	(3344H)
DD1C	INC	(IX+IY)
DD1D	DEC	(IX+IY)
DD1E66	LD	(IX+IY),66H
DD2074	JAR	77H
DD214433	LD	IX,3344H
DD214433	LDA	IX,(3344H)
DD214433	LDW	IX,3344H
DD224433	LD	(3344H),IX
DD224433	LDW	(3344H),IX
DD23	INC	IX
DD23	INCW	IX
DD24	INC	IXH
DD25	DEC	IXH
DD2666	LD	IXH,66H
DD2874	JAF	77H
DD29	ADD	IX,IX
DD2A4433	LD	IX,(3344H)
DD2A4433	LDW	IX,(3344H)
DD2B	DEC	IX
DD2B	DECW	IX
DD2C	INC	IXL
DD2D	DEC	IXL
DD2E66	LD	IXL,66H
DD31221144	LDW	(PC+1122H),3344H
DD332211	INCW	(PC+1122H)
DD3455	INC	(IX+55H)
DD3555	DEC	(IX+55H)
DD365566	LD	(IX+55H),66H
DD39	ADD	IX,SP
DD3B2211	DECW	(PC+1122H)
DD3C4433	INC	(3344H)
DD3D4433	DEC	(3344H)
DD3E443366	LD	(3344H),66H
DD44	LD	B,IXH
DD45	LD	B,IXL
DD4655	LD	B,(IX+55H)
DD4C	LD	C,IXH
DD4D	LD	C,IXL
DD4E55	LD	C,(IX+55H)
DD54	LD	D,IXH
DD55	LD	D,IXL
DD5655	LD	D,(IX+55H)
DD5C	LD	E,IXH
DD5D	LD	E,IXL
DD5E55	LD	E,(IX+55H)
DD60	LD	IXH,B
DD61	LD	IXH,C
DD62	LD	IXH,D
DD63	LD	IXH,E
DD64	LD	IXH,IXH
DD65	LD	IXH,IXL
DD6655	LD	H,(IX+55H)
DD67	LD	IXH,A
DD68	LD	IXL,B
DD69	LD	IXL,C
DD6A	LD	IXL,D
DD6B	LD	IXL,E
DD6C	LD	IXL,IXH
DD6D	LD	IXL,IXL
DD6E55	LD	L,(IX+55H)
DD6F	LD	IXL,A
DD7055	LD	(IX+55H),B
DD7155	LD	(IX+55H),C
DD7255	LD	(IX+55H),D
DD7355	LD	(IX+55H),E
DD7455	LD	(IX+55H),H
DD7555	LD	(IX+55H),L
DD7755	LD	(IX+55H),A
DD782211	LD	A,(SP+1122H)
DD79	LD	A,(HL+IX)
DD7A	LD	A,(HL+IY)
DD7B	LD	A,(IX+IY)
DD7C	LD	A,IXH
DD7D	LD	A,IXL

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
DD7E55	LD	A,(IX+55H)	DDBB	CP	A,(IX+IY)
DD802211	ADD	A,(SP+1122H)	DDBC	CP	A,IXH
DD81	ADD	A,(HL+IX)	DDBD	CP	A,IXL
DD82	ADD	A,(HL+IY)	DDBE55	CP	A,(IX+55H)
DD83	ADD	A,(IX+IY)	DDBF4433	CP	A,(3344H)
DD84	ADD	A,IXH	DDC1	POP	(HL)
DD85	ADD	A,IXL	DDC2	JP	NZ,(HL)
DD8655	ADD	A,(IX+55H)	DDC4	CALL	NZ,(HL)
DD874433	ADD	A,(3344H)	DDC5	PUSH	(HL)
DD882211	ADC	A,(SP+1122H)	DDCA	JP	Z,(HL)
DD89	ADC	A,(HL+IX)	DDCB5506	RLC	(IX+55H)
DD8A	ADC	A,(HL+IY)	DDCB550E	RRC	(IX+55H)
DD8B	ADC	A,(IX+IY)	DDCB5516	RL	(IX+55H)
DD8C	ADC	A,IXH	DDCB551E	RR	(IX+55H)
DD8D	ADC	A,IXL	DDCB5526	SLA	(IX+55H)
DD8E55	ADC	A,(IX+55H)	DDCB552E	SRA	(IX+55H)
DD8F4433	ADC	A,(3344H)	DDCB5536	TSET	(IX+55H)
DD902211	SUB	A,(SP+1122H)	DDCB553E	SRL	(IX+55H)
DD91	SUB	A,(HL+IX)	DDCB5546	BIT	0,(IX+55H)
DD92	SUB	A,(HL+IY)	DDCB554E	BIT	1,(IX+55H)
DD93	SUB	A,(IX+IY)	DDCB5556	BIT	2,(IX+55H)
DD94	SUB	A,IXH	DDCB555E	BIT	3,(IX+55H)
DD95	SUB	A,IXL	DDCB5566	BIT	4,(IX+55H)
DD9655	SUB	A,(IX+55H)	DDCB556E	IT	5,(IX+55H)
DD974433	SUB	A,(3344H)	DDCB5576	BIT	6,(IX+55H)
DD982211	SBC	A,(SP+1122H)	DDCB557E	BIT	7,(IX+55H)
DD99	SBC	A,(HL+IX)	DDCB5586	RES	0,(IX+55H)
DD9A	SBC	A,(HL+IY)	DDCB558E	RES	1,(IX+55H)
DD9B	SBC	A,(IX+IY)	DDCB5596	RES	2,(IX+55H)
DD9C	SBC	A,IXH	DDCB559E	RES	3,(IX+55H)
DD9D	SBC	A,IXL	DDCB55A6	RES	4,(IX+55H)
DD9E55	SBC	A,(IX+55H)	DDCB55AE	RES	5,(IX+55H)
DD9F4433	SBC	A,(3344H)	DDCB55B6	RES	6,(IX+55H)
DDA02211	AND	A,(SP+1122H)	DDCB55BE	RES	7,(IX+55H)
DDA1	AND	A,(HL+IX)	DDCB55C6	SET	0,(IX+55H)
DDA2	AND	A,(HL+IY)	DDCB55CE	SET	1,(IX+55H)
DDA3	AND	A,(IX+IY)	DDCB55D6	SET	2,(IX+55H)
DDA4	AND	A,IXH	DDCB55DE	SET	3,(IX+55H)
DDA5	AND	A,IXL	DDCB55E6	SET	4,(IX+55H)
DDA655	AND	A,(IX+55H)	DDCB55EE	SET	5,(IX+55H)
DDA74433	AND	A,(3344H)	DDCB55F6	SET	6,(IX+55H)
DDA82211	XOR	A,(SP+1122H)	DDCB55FE	SET	7,(IX+55H)
DDA9	XOR	A,(HL+IX)	DDCC	CALL	Z,(HL)
DDAA	XOR	A,(HL+IY)	DDCD	CALL	(HL)
DDAB	XOR	A,(IX+IY)	DDD14433	POP	(3344H)
DDAC	XOR	A,IXH	DDD2	JP	NC,(HL)
DDAD	XOR	A,IXL	DDD4	CALL	NC,(HL)
DDAE55	XOR	A,(IX+55H)	DDD54433	PUSH	(3344H)
DDAF4433	XOR	A,(3344H)	DDDA	JP	C,(HL)
DDB02211	OR	A,(SP+1122H)	DDDC	CALL	C,(HL)
DDB1	OR	A,(HL+IX)	DDE1	POP	IX
DDB2	OR	A,(HL+IY)	DDE2	JP	PO,(HL)
DDB3	OR	A,(IX+IY)	DDE3	EX	(SP),IX
DDB4	OR	A,IXH	DDE4	CALL	PO,(HL)
DDB5	OR	A,IXL	DDE5	PUSH	IX
DDB655	OR	A,(IX+55H)	DDE9	JP	(IX)
DDB74433	OR	A,(3344H)	DDEA	JP	PE,(HL)
DDB82211	CP	A,(SP+1122H)	DDEB	EX	IX,HL
DDB9	CP	A,(HL+IX)	DDEC	CALL	PE,(HL)
DDBA	CP	A,(HL+IY)	DDED022211	LDA	IX,(SP+1122H)

OBJECT CODE	SOURCE CODE	
DDED042211	LD	IX,(SP+1122H)
DDED042211	LDW	IX,(SP+1122H)
DDED052211	LD	(SP+1122H),IX
DDED052211	LDW	(SP+1122H),IX
DDED0655	LD	BC,(IX+55H)
DDED0655	LDW	BC,(IX+55H)
DDED072211	EX	A,(SP+1122H)
DDED0A	LDA	IX,(HL+IX)
DDED0C	LD	IX,(HL+IX)
DDED0C	LDW	IX,(HL+IX)
DDED0D	LD	(HL+IX),IX
DDED0D	LDW	(HL+IX),IX
DDED0E55	LD	(IX+55H),BC
DDED0E55	LDW	(IX+55H),BC
DDED0F	EX	A,(HL+IX)
DDED12	LDA	IX,(HL+IY)
DDED14	LD	IX,(HL+IY)
DDED14	LDW	IX,(HL+IY)
DDED15	LD	(HL+IY),IX
DDED15	LDW	(HL+IY),IX
DDED1655	LD	DE,(IX+55H)
DDED1655	LDW	DE,(IX+55H)
DDED17	EX	A,(HL+IY)
DDED1A	LDA	IX,(IX+IY)
DDED1C	LD	IX,(IX+IY)
DDED1C	LDW	IX,(IX+IY)
DDED1D	LD	(IX+IY),IX
DDED1D	LDW	(IX+IY),IX
DDED1E55	LD	(IX+55H),DE
DDED1E55	LDW	(IX+55H),DE
DDED1F	EX	A,(IX+IY)
DDED222211	LDA	IX,(PC+1122H)
DDED242211	LD	IX,(PC+1122H)
DDED242211	LDW	IX,(PC+1122H)
DDED252211	LD	(PC+1122H),IX
DDED252211	LDW	(PC+1122H),IX
DDED2655	LD	HL,(IX+55H)
DDED2655	LDW	HL,(iX+55H)
DDED27	EX	A,IXH
DDED2A2211	LDA	IX,(IX+1122H)
DDED2C2211	LD	IX,(IX+1122H)
DDED2C2211	LDW	IX,(IX+1122H)
DDED2D2211	LD	(IX+1122H),IX
DDED2D2211	LDW	(IX+1122H),IX
DDED2E55	LD	(IX+55H),HL
DDED2E55	LDW	(IX+55H),HL
DDED2F	EX	A,IXL
DDED322211	LDA	IX,(IY+1122H)
DDED342211	LD	IX,(IY+1122H)
DDED342211	LDW	IX,(IY+1122H)
DDED352211	LD	(IY+1122H),IX
DDED352211	LDW	(IY+1122H),IX
DDED3655	LD	SP,(IX+55H)
DDED3655	LDW	SP,(IX+55H)
DDED3755	EX	A,(IX+55H)
DDED3A2211	LDA	IX,(HL+1122H)
DDED3C2211	LD	IX,(HL+1122H)
DDED3C2211	LDW	IX,(HL+1122H)
DDED3D2211	LD	(HL+1122H),IX
DDED3D2211	LDW	(HL+1122H),IX

OBJECT CODE	SOURCE CODE	
DDED3E55	LD	(IX+55H),SP
DDED3E55	LDW	(IX+55H),SP
DDED3F4433	EX	A,(3344H)
DDED402211	IN	(SP+1122H),(C)
DDED412211	OUT	(C),(SP+1122H)
DDED42	SBC	IX,BC
DDED48	IN	(HL+IX),(C)
DDED49	OUT	(C),(HL+IX)
DDED4A	ADC	IX,BC
DDED50	IN	(HL+IY),(C)
DDED51	OUT	(C),(HL+IY)
DDED52	SBC	IX,DE
DDED58	IN	(IX+IY),(C)
DDED59	OUT	(C),(IX+IY)
DDED5A	ADC	IX,DE
DDED60	IN	IXH,(C)
DDED61	OUT	(C),IXH
DDED62	SBC	IX,IX
DDED66	LDCTL	IX,(C)
DDED68	IN	IXL,(C)
DDED69	OUT	(C),IXL
DDED6A	ADC	IX,IX
DDED6D	ADD	IX,A
DDED6E	LDCTL	(C),IX
DDED72	SBC	IX,SP
DDED784433	IN	(3344H),(C)
DDED794433	OUT	(C),(3344H)
DDED7A	ADC	IX,SP
DDED8655	LDUD	A,(IX+55H)
DDED87	LDCTL	IX,USP
DDED8E55	LDUD	(IX+55H),A
DDED8F	LDCTL	USP,IX
DDED9655	LDUP	A,(IX+55H)
DDED9E55	LDUP	(IX+55H),A
DDEDC02211	MULT	A,(SP+1122H)
DDEDC12211	MULTU	A,(SP+1122H)
DDEDC2	MULTW	HL,(HL)
DDEDC3	MULTUW	HL,(HL)
DDEDC42211	DIV	HL,(SP+1122H)
DDEDC52211	DIVU	HL,(SP+1122H)
DDEDC6	ADDW	HL,(HL)
DDEDC7	CPW	HL,(HL)
DDEDC8	MULT	A,(HL+IX)
DDEDC9	MULTU	A,(HL+IX)
DDEDCA	DIVW	DEHL,(HL)
DDEDCB	DIVUW	DEHL,(HL)
DDEDCC	DIV	HL,(HL+IX)
DDEDCD	DIVU	HL,(HL+IX)
DDEDCE	SUBW	HL,(HL)
DDEDD0	MULT	A,(HL+IY)
DDEDD1	MULTU	A,(HL+IY)
DDEDD24433	MULTW	HL,(3344H)
DDEDD34433	MULTUW	HL,(3344H)
DDEDD4	DIV	HL,(HL+IY)
DDEDD5	DIVU	HL,(HL+IY)
DDEDD64433	ADDW	HL,(3344H)
DDEDD74433	CPW	HL,(3344H)
DDEDD8	MULT	A,(IX+IY)
DDEDD9	MULTU	A,(IX+IY)
DDEDDA4433	DIVW	DEHL,(3344H)

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
DDEDDB4433	DIVUW	DEHL,(3344H)	ED042211	LDW	HL,(SP+1122H)
DDEDDC	DIV	HL,(IX+IY)	ED052211	LD	(SP+1122H),HL
DDEDDD	DIVU	HL,(IX+IY)	ED052211	LDW	(SP+1122H),HL
DDEDDE4433	SUBW	HL,(3344H)	ED06	LD	BC,(HL)
DDEDE0	MULT	A,IXH	ED06	LDW	BC,(HL)
DDEDE1	MULTU	A,IXH	ED07	EX	A,B
DDEDE2	MULTW	HL,IX	ED0A	LDA	HL,(HL+IX)
DDEDE3	MULTUW	HL,IX	ED0B	LD	(HL+IX),A
DDEDE4	DIV	HL,IXH	ED0C	LD	HL,(HL+IX)
DDEDE5	DIVU	HL,IXH	ED0C	LDW	HL,(HL+IX)
DDEDE6	ADDW	HL,IX	ED0D	LD	(HL+IX),HL
DDEDE7	CPW	HL,IX	ED0D	LDW	(HL+IX),HL
DDEDE8	MULT	A,IXL	ED0E	LD	(HL),BC
DDEDE9	MULTU	A,IXL	ED0E	LDW	(HL),BC
DDEDEA	DIVW	DEHL,IX	ED0F	EX	A,C
DDEDEB	DIVUW	DEHL,IX	ED12	LDA	HL,(HL+IY)
DDEDEC	DIV	HL,IXL	ED13	LD	(HL+IY),A
DDEDED	DIVU	HL,IXL	ED14	LD	HL,(HL+IY)
DDEDEE	SUBW	HL,IX	ED14	LDW	HL,(HL+IY)
DDEDF055	MULT	A,(IX+55H)	ED15	LD	(HL+IY),HL
DDEDF155	MULTU	A,(IX+55H)	ED15	LDW	(HL+IY),HL
DDEDF22211	MULTW	HL,(PC+1122H)	ED16	LD	DE,(HL)
DDEDF32211	MULTUW	HL,(PC+1122H)	ED16	LDW	DE,(HL)
DDEDF455	DIV	HL,(IX+55H)	ED17	EX	A,D
DDEDF555	DIVU	HL,(IX+55H)	ED1A	LDA	HL,(IX+IY)
DDEDF62211	ADDW	HL,(PC+1122H)	ED1B	LD	(IX+IY),A
DDEDF72211	CPW	HL,(PC+1122H)	ED1C	LD	HL,(IX+IY)
DDEDF84433	MULT	A,(3344H)	ED1C	LDW	HL,(IX+IY)
DDEDF94433	MULTU	A,(3344H)	ED1D	LD	(IX+IY),HL
DDEDFA2211	DIVW	DEHL,(PC+1122H)	ED1D	LDW	(IX+IY),HL
DDEDFB2211	DIVUW	DEHL,(PC+1122H)	ED1E	LD	(HL),DE
DDEDFC4433	DIV	HL,(3344H)	ED1E	LDW	(HL),DE
DDEDFD4433	DIVU	HL,(3344H)	ED1F	EX	A,E
DDEDFE2211	SUBW	HL,(PC+1122H)	ED222211	LDA	HL,(PC+1122H)
DDF12211	POP	(PC+1122H)	ED232211	LD	(PC+1122H),A
DDF2	JP	P,(HL)	ED242211	LD	HL,(PC+1122H)
DDF4	CALL	P,(HL)	ED242211	LDW	HL,(PC+1122H)
DDF52211	PUSH	(PC+1122H)	ED252211	LD	(PC+1122H),HL
DDF9	LDW	SP,IX	ED252211	LDW	(PC+1122H),HL
DDF9	LD	SP,IX	ED26	LD	HL,(HL)
DDFA	JP	M,(HL)	ED26	LDW	HL,(HL)
DDFC	CALL	M,(HL)	ED27	EX	A,H
DE66	SBC	A,66H	ED2A2211	LDA	HL,(IX+1122H)
DF	RST	18H	ED2B2211	LD	(IX+1122H),A
E0	RET	PO	ED2C2211	LD	HL,(IX+1122H)
E1	POP	HL	ED2C2211	LDW	HL,(IX+1122H)
E24433	JP	PO,3344H	ED2D2211	LD	(IX+1122H),HL
E3	EX	(SP),HL	ED2D2211	LDW	(IX+1122H),HL
E44433	CALL	PO,3344H	ED2E	LD	(HL),HL
E5	PUSH	HL	ED2E	LDW	(HL),HL
E666	AND	A,66H	ED2F	EX	A,L
E7	RST	20H	ED322211	LDA	HL,(IY+1122H)
E8	RET	PE	ED332211	LD	(IY+1122H),A
E9	JP	(HL)	ED342211	LD	HL,(IY+1122H)
EA4433	JP	PE,3344H	ED342211	LDW	HL,(IY+1122H)
EB	EX	DE,HL	ED352211	LD	(IY+1122H),HL
EC4433	CALL	PE,3344H	ED352211	LDW	(IY+1122H),HL
ED022211	LDA	HL,(SP+1122H)	ED36	LD	SP,(HL)
ED032211	LD	(SP+1122H),A	ED36	LDW	SP,(HL)
ED042211	LD	HL,(SP+1122H)	ED37	EX	A,(HL)

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
ED3A2211	LDA	HL,(HL+1122H)	ED7766	DI	66H
ED3B2211	LD	(HL+1122H),A	ED78	IN	A,(C)
ED3C2211	LDW	HL,(HL+1122H)	ED79	OUT	(C),A
ED3D2211	LD	(HL+1122H),HL	ED7A	ADC	HL,SP
ED3D2211	LDW	(HL+1122H),HL	ED7B4433	LD	SP,(3344H)
ED3E	LD	(HL),SP	ED7B4433	LDW	SP,(3344H)
ED3E	LDW	(HL),SP	ED7F66	EI	66H
ED3F	EX	A,A	ED82	INIW	
ED40	IN	B,(C)	ED83	OUTIW	
ED41	OUT	(C),B	ED842211	EPUM	(SP+1122H)
ED42	SBC	HL,BC	ED852211	MEPU	(SP+1122H)
ED434433	LD	(3344H),BC	ED86	LDUD	A,(HL)
ED434433	LDW	(3344H),BC	ED87	LDCTL	HL,USP
ED44	NEG	A	ED8A	INDW	
ED45	RETN		ED8B	OUTDW	
ED46	IM	0	ED8C	EPUM	(HL+IX)
ED47	LD	I,A	ED8D	MEPU	(HL+IX)
ED48	IN	C,(C)	ED8E	LDUD	(HL),A
ED49	OUT	(C),C	ED8F	LDCTL	USP,HL
ED4A	ADC	HL,BC	ED92	INIRW	
ED4B4433	LD	BC,(3344H)	ED93	OTIRW	
ED4B4433	LDW	BC,(3344H)	ED94	EPUM	(HL+IY)
ED4C	NEG	HL	ED95	MEPU	(HL+IY)
ED4D	RETI		ED96	LDUP	A,(HL)
ED4E	IM	3	ED97	EPUF	
ED4F	LD	R,A	ED9A	INDRW	
ED50	IN	D,(C)	ED9B	OTDRW	
ED51	OUT	(C),D	ED9C	EPUM	(IX+IY)
ED52	SBC	HL,DE	ED9D	MEPU	(IX+IY)
ED534433	LD	(3344H),DE	ED9E	LDUP	(HL),A
ED534433	LDW	(3344H),DE	ED9F	EPUI	
ED55	RETIL		EDA0	LDI	
ED56	IM	1	EDA1	CPI	
ED57	LD	A,I	EDA2	INI	
ED58	IN	E,(C)	EDA3	OUTI	
ED59	OUT	(C),E	EDA42211	EPUM	(PC+1122H)
ED5A	ADC	HL,DE	EDA52211	MEPU	(PC+1122H)
ED5B4433	LD	DE,(3344H)	EDA6	EPUM	(HL)
ED5B4433	LDW	DE,(3344H)	EDA74433	EPUM	(3344H)
ED5E	IM	2	EDA8	LDD	
ED5F	LD	A,R	EDA9	CPD	
ED60	IN	H,(C)	EDAA	IND	
ED61	OUT	(C),H	EDAB	OUTD	
ED62	SBC	HL,HL	EDAC2211	EPUM	(IX+1122H)
ED64	EXTS	A	EDAD2211	MEPU	(IX+1122H)
ED65	PCACHE		EDAE	MEPU	(HL)
ED66	LDCTL	HL,(C)	EDAF4433	MEPU	(3344H)
ED67	RRD		EDB0	LDIR	
ED68	IN	L,(C)	EDB1	CPIR	
ED69	OUT	(C),L	EDB2	INIR	
ED6A	ADC	HL,HL	EDB3	OTIR	
ED6C	EXTS	HL	EDB42211	EPUM	(IY+1122H)
ED6D	ADD	HL,A	EDB52211	MEPU	(IY+1122H)
ED6E	LDCTL	(C),HL	EDB7	IN	HL,(C)
ED6F	RLD		EDB7	INW	HL,(C)
ED70	TSTI	(C)	EDB8	LDDR	
ED714433	SC	3344H	EDB9	CPDR	
ED72	SBC	HL,SP	EDBA	INDR	
ED734433	LD	(3344H),SP	EDBB	OTDR	
ED734433	LDW	(3344H),SP	EDBC2211	EPUM	(HL+1122H)

OBJECT CODE	SOURCE CODE	OBJECT CODE	SOURCE CODE
EDBD2211	MEPU (HL+1122H)	EDFB	DIVUW DEHL,SP
EDBF	OUT (C),HL	EDFC	DIV HL,A
EDBF	OUTW (C),HL	EDFD	DIVU HL,A
EDC0	MULT A,B	EDFE	SUBW HL,SP
EDC1	MULTU A,B	EE66	XOR A,66H
EDC2	MULTW HL,BC	EF	RST 28H
EDC3	MULTUW HL,BC	F0	RET P
EDC4	DIV HL,B	F1	POP AF
EDC5	DIVU HL,B	F24433	JP P,3344H
EDC6	ADDW HL,BC	F3	DI
EDC7	CPW HL,BC	F44433	CALL P,3344H
EDC8	MULT A,C	F5	PUSH AF
EDC9	MULTU A,C	F666	OR A,66H
EDCA	DIVW DEHL,BC	F7	RST 30H
EDCB	DIVUW DEHL,BC	F8	RET M
EDCC	DIV HL,C	F9	LDW SP,HL
EDCD	DIVU HL,C	F9	LD SP,HL
EDCE	SUBW HL,BC	FA4433	JP M,3344H
EDD0	MULT A,D	FB	EI
EDD1	MULTU A,D	FC4433	CALL M,3344H
EDD2	MULTW HL,DE	FD032211	INCW (IX+1122H)
EDD3	MULTUW HL,DE	FD042211	INC (PC+1122H)
EDD4	DIV HL,D	FD052211	DEC (PC+1122H)
EDD5	DIVU HL,D	FD06221166	LD (PC+1122H),66H
EDD6	ADDW HL,DE	FD09	ADD IY,BC
EDD7	CPW HL,DE	FD0B2211	DECW (IX+1122H)
EDD8	MULT A,E	FD0C2211	INC (IX+1122H)
EDD9	MULTU A,E	FD0D2211	DEC (IX+1122H)
EDDA	DIVW DEHL,DE	FD0E221166	LD (IX+1122H),66H
EDDB	DIVUW DEHL,DE	FD132211	INCW (IY+1122H)
EDDC	DIV HL,E	FD142211	INC (IY+1122H)
EDDD	DIVU HL,E	FD152211	DEC (IY+1122H)
EDDE	SUBW HL,DE	FD16221166	LD (IY+1122H),66H
EDE0	MULT A,H	FD19	ADD IY,DE
EDE1	MULTU A,H	FD1B2211	DECW (IY+1122H)
EDE2	MULTW HL,HL	FD1C2211	INC (HL+1122H)
EDE3	MULTUW HL,HL	FD1D2211	DEC (HL+1122H)
EDE4	DIV HL,H	FD1E221166	LD (HL+1122H),66H
EDE5	DIVU HL,H	FD214433	LD IY,3344H
EDE6	ADDW HL,HL	FD214433	LDA IY,(3344H)
EDE7	CPW HL,HL	FD214433	LDW IY,3344H
EDE8	MULT A,L	FD224433	LD (3344H),IY
EDE9	MULTU A,L	FD224433	LDW (3344H),IY
EDEA	DIVW DEHL,HL	FD23	INC IY
EDEB	DIVUW DEHL,HL	FD23	INCW IY
EDEC	DIV HL,L	FD24	INC IYH
EDED	DIVU HL,L	FD25	DEC IYH
EDEE	SUBW HL,HL	FD2666	LD IYH,66H
EDEF	EX H,L	FD29	ADD IY,IY
EDF0	MULT A,(HL)	FD2A4433	LDW IY,(3344H)
EDF1	MULTU A,(HL)	FD2B	DEC IY
EDF2	MULTW HL,SP	FD2B	DECW IY
EDF3	MULTUW HL,SP	FD2C	INC IYL
EDF4	DIV HL,(HL)	FD2D	DEC IYL
EDF5	DIVU HL,(HL)	FD2E66	LD IYL,66H
EDF6	ADDW HL,SP	FD3455	INC (IY+55H)
EDF7	CPW HL,SP	FD3555	DEC (IY+55H)
EDF8	MULT A,A	FD365566	LD (IY+55H),66H
EDF9	MULTU A,A	FD39	ADD IY,SP
EDFA	DIVW DEHL,SP	FD44	LD B,IYH

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
FD45	LD	B,IYL	FD95	SUB	A,IYL
FD4655	LD	B,(IY+55H)	FD9655	SUB	A,(IY+55H)
FD4C	LD	C,IYH	FD982211	SBC	A,(PC+1122H)
FD4D	LD	C,IYL	FD992211	SBC	A,(IX+1122H)
FD4E55	LD	C,(IY+55H)	FD9A2211	SBC	A,(IY+1122H)
FD54	LD	D,IYH	FD9B2211	SBC	A,(HL+1122H)
FD55	LD	D,IYL	FD9C	SBC	A,IYH
FD5666	LD	D,(IY+55H)	FD9D	SBC	A,IYL
FD5C	LD	E,IYH	FD9E55	SBC	A,(IY+55H)
FD5D	LD	E,IYL	FDA02211	AND	A,(PC+1122H)
FD5E55	LD	E,(IY+55H)	FDA12211	AND	A,(IX+1122H)
FD60	LD	IYH,B	FDA22211	AND	A,(IY+1122H)
FD61	LD	IYH,C	FDA32211	AND	A,(HL+1122H)
FD62	LD	IYH,D	FDA4	AND	A,IYH
FD63	LD	IYH,E	FDA5	AND	A,IYL
FD64	LD	IYH,IYH	FDA655	AND	A,(IY+55H)
FD65	LD	IYH,IYL	FDA82211	XOR	A,(PC+1122H)
FD6655	LD	H,(IY+55H)	FDA92211	XOR	A,(IX+1122H)
FD67	LD	IYH,A	FDAA2211	XOR	A,(IY+1122H)
FD68	LD	IYL,B	FDAB2211	XOR	A,(HL+1122H)
FD69	LD	IYL,C	FDAC	XOR	A,IYH
FD6A	LD	IYL,D	FDAD	XOR	A,IYL
FD6B	LD	IYL,E	FDAE55	XOR	A,(IY+55H)
FD6C	LD	IYL,IYH	FDB02211	OR	A,(PC+1122H)
FD6D	LD	IYL,IYL	FDB12211	OR	A,(IX+1122H)
FD6E55	LD	L,(IY+55H)	FDB22211	OR	A,(IY+1122H)
FD6F	LD	IYL,A	FDB32211	OR	A,(HL+1122H)
FD7055	LD	(IY+55H),B	FDB4	OR	A,IYH
FD7155	LD	(IY+55H),C	FDB5	OR	A,IYL
FD7255	LD	(IY+55H),D	FDB655	OR	A,(IY+55H)
FD7355	LD	(IY+55H),E	FDB82211	CP	A,(PC+1122H)
FD7455	LD	(IY+55H),H	FDB92211	CP	A,(IX+1122H)
FD7555	LD	(IY+55H),L	FDBA2211	CP	A,(IY+1122H)
FD7755	LD	(IY+55H),A	FDBB2211	CP	A,(HL+1122H)
FD782211	LD	A,(PC+1122H)	FDBC	CP	A,IYH
FD792211	LD	A,(IX+1122H)	FDBD	CP	A,IYL
FD7A2211	LD	A,(IY+1122H)	FDBE55	CP	A,(IY+55H)
FD7B2211	LD	A,(HL+1122H)	FDC22211	JP	NZ,(PC+1122H)
FD7C	LD	A,IYH	FDC32211	JP	(PC+1122H)
FD7D	LD	A,IYL	FDC42211	CALL	NZ,(PC+1122H)
FD7E55	LD	A,(IY+55H)	FDCA2211	JP	Z,(PC+1122H)
FD802211	ADD	A,(PC+1122H)	FDCB5506	RLC	(IY+55H)
FD812211	ADD	A,(IX+1122H)	FDCB550E	RRC	(IY+55H)
FD822211	ADD	A,(IY+1122H)	FDCB5516	RL	(IY+55H)
FD832211	ADD	A,(HL+1122H)	FDCB551E	RR	(IY+55H)
FD84	ADD	A,IYH	FDCB5526	SLA	(IY+55H)
FD85	ADD	A,IYL	FDCB552E	SRA	(IY+55H)
FD8655	ADD	A,(IY+55H)	FDCB5536	TSET	(IY+55H)
FD882211	ADC	A,(PC+1122H)	FDCB553E	SRL	(IY+55H)
FD892211	ADC	A,(IX+1122H)	FDCB5546	BIT	0,(IY+55H)
FD8A2211	ADC	A,(IY+1122H)	FDCB554E	BIT	1,(IY+55H)
FD8B2211	ADC	A,(HL+1122H)	FDCB5556	BIT	2,(IY+55H)
FD8C	ADC	A,IYH	FDCB555E	BIT	3,(IY+55H)
FD8D	ADC	A,IYL	FDCB5566	BIT	4,(IY+55H)
FD8E55	ADC	A,(IY+55H)	FDCB556E	BIT	5,(IY+55H)
FD902211	SUB	A,(PC+1122H)	FDCB5576	BIT	6,(IY+55H)
FD912211	SUB	A,(IX+1122H)	FDCB557E	BIT	7,(IY+55H)
FD922211	SUB	A,(IY+1122H)	FDCB5586	RES	0,(IY+55H)
FD932211	SUB	A,(HL+1122H)	FDCB558E	RES	1,(IY+55H)
FD94	SUB	A,IYH	FDCB5596	RES	2,(IY+55H)

OBJECT CODE	SOURCE CODE		OBJECT CODE	SOURCE CODE	
FDCB559E	RES	3,(IY+55H)	FDED222211	LDA	IY,(PC+1122H)
FDCB55A6	RES	4,(IY+55H)	FDED242211	LD	IY,(PC+1122H)
FDCB55AE	RES	5,(IY+55H)	FDED242211	LDW	IY,(PC+1122H)
FDCB55B6	RES	6,(IY+55H)	FDED252211	LD	(PC+1122H),IY
FDCB55BE	RES	7,(IY+55H)	FDED252211	LDW	(PC+1122H),IY
FDCB55C6	SET	0,(IY+55H)	FDED2655	LD	HL,(IY+55H)
FDCB55CE	SET	1,(IY+55H)	FDED2655	LDW	HL,(IY+55H)
FDCB55D6	SET	2,(IY+55H)	FDED27	EX	A,IYH
FDCB55DE	SET	3,(IY+55H)	FDED2A2211	LDA	IY,(IX+1122H)
FDCB55E6	SET	4,(IY+55H)	FDED2C2211	LD	IY,(IX+1122H)
FDCB55EE	SET	5,(IY+55H)	FDED2C2211	LDW	IY,(IX+1122H)
FDCB55F6	SET	6,(IY+55H)	FDED2D2211	LD	(IX+1122H),IY
FDCB55FE	SET	7,(IY+55H)	FDED2D2211	LDW	(IX+1122H),IY
FDCC2211	CALL	Z,(PC+1122H)	FDED2E55	LD	(IY+55H),HL
FDCD2211	CALL	(PC+1122H)	FDED2E55	LDW	(IY+55H),HL
FDD22211	JP	NC,(PC+1122H)	FDED2F	EX	A,IYL
FDD42211	CALL	NC,(PC+1122H)	FDED322211	LDA	IY,(IY+1122H)
FDDA2211	JP	C,(PC+1122H)	FDED342211	LD	IY,(IY+1122H)
FDDC2211	CALL	C,(PC+1122H)	FDED342211	LDW	IY,(IY+1122H)
FDE1	POP	IY	FDED352211	LD	(IY+1122H),IY
FDE22211	JP	PO,(PC+1122H)	FDED352211	LDW	(IY+1122H),IY
FDE3	EX	(SP),IY	FDED3655	LD	SP,(IY+55H)
FDE42211	CALL	PO,(PC+1122H)	FDED3655	LDW	SP,(IY+55H)
FDE5	PUSH	IY	FDED3755	EX	A,(IY+55H)
FDE9	JP	(IY)	FDED3A2211	LDA	IY,(HL+1122H)
FDEA2211	JP	PE,(PC+1122H)	FDED3C2211	LD	IY,(HL+1122H)
FDEB	EX	IY,HL	FDED3C2211	LDW	IY,(HL+1122H)
FDEC2211	CALL	PE,(PC+1122H)	FDED3D2211	LD	(HL+1122H),IY
FDED022211	LDA	IY,(SP+1122H)	FDED3D2211	LDW	(HL+1122H),IY
FDED042211	LD	IY,(SP+1122H)	FDED3E55	LD	(IY+55H),SP
FDED042211	LDW	IY,(SP+1122H)	FDED3E55	LDW	(IY+55H),SP
FDED052211	LD	(SP+1122H),IY	FDED402211	IN	(PC+1122H),(C)
FDED052211	LDW	(SP+1122H),IY	FDED412211	OUT	(C),(PC+1122H)
FDED0655	LD	BC,(IY+55H)	FDED42	SBC	IY,BC
FDED0655	LDW	BC,(IY+55H)	FDED482211	IN	(IX+1122H),(C)
FDED072211	EX	A,(PC+1122H)	FDED492211	OUT	(C),(IX+1122H)
FDED0A	LDA	IY,(HL+IX)	FDED4A	ADC	IY,BC
FDED0C	LD	IY,(HL+IX)	FDED502211	IN	(IY+1122H),(C)
FDED0C	LDW	IY,(HL+IX)	FDED512211	OUT	(C),(IY+1122H)
FDED0D	LD	(HL+IX),IY	FDED52	SBC	IY,DE
FDED0D	LDW	(HL+IX),IY	FDED582211	IN	(HL+1122H),(C)
FDED0E55	LD	(IY+55H),BC	FDED592211	OUT	(C),(HL+1122H)
FDED0E55	LDW	(IY+55H),BC	FDED5A	ADC	IY,DE
FDED0F2211	EX	A,(IX+1122H)	FDED60	IN	IYH,(C)
FDED12	LDA	IY,(HL+IY)	FDED61	OUT	(C),IYH
FDED14	LD	IY,(HL+IY)	FDED62	SBC	IY,IY
FDED14	LDW	IY,(HL+IY)	FDED66	LDCTL	IY,(C)
FDED15	LD	(HL+IY),IY	FDED68	IN	IYL,(C)
FDED15	LDW	(HL+IY),IY	FDED69	OUT	(C),IYL
FDED1655	LD	DE,(IY+55H)	FDED6A	ADC	IY,IY
FDED1655	LDW	DE,(IY+55H)	FDED6D	ADD	IY,A
FDED172211	EX	A,(IY+1122H)	FDED6E	LDCTL	(C),IY
FDED1A	LDA	IY,(IX+IY)	FDED72	SBC	IY,SP
FDED1C	LD	IY,(IX+IY)	FDED7A	ADC	IY,SP
FDED1C	LDW	IY,(IX+IY)	FDED8655	LDUD	A,(IY+55H)
FDED1D	LD	(IX+IY),IY	FDED87	LDCTL	IY,USP
FDED1D	LDW	(IX+IY),IY	FDED8E55	LDUD	(IY+55H),A
FDED1E55	LD	(IY+55H),DE	FDED8F	LDCTL	USP,IY
FDED1E55	LDW	(IY+55H),DE	FDED9655	LDUP	A,(IY+55H)
FDED1F2211	EX	A,(HL+1122H)	FDED9E55	LDUP	(IY+55H),A

OBJECT CODE	SOURCE CODE
FDEDC02211	MULT A,(PC+1122H)
FDEDC12211	MULTU A,(PC+1122H)
FDEDC22211	MULTW HL,(IX+1122H)
FDEDC32211	MULTUW HL,(IX+1122H)
FDEDC42211	DIV HL,(PC+1122H)
FDEDC52211	DIVU HL,(PC+1122H)
FDEDC62211	ADDW HL,(IX+1122H)
FDEDC72211	CPW HL,(IX+1122H)
FDEDC82211	MULT A,(IX+1122H)
FDEDC92211	MULTU A,(IX+1122H)
FDEDCA2211	DIVW DEHL,(IX+1122H)
FDEDCB2211	DIVUW DEHL,(IX+1122H)
FDEDCC2211	DIV HL,(IX+1122H)
FDEDCD2211	DIVU HL,(IX+1122H)
FDEDCE2211	SUBW HL,(IX+1122H)
FDEDD02211	MULT A,(IY+1122H)
FDEDD12211	MULTU A,(IY+1122H)
FDEDD22211	MULTW HL,(IY+1122H)
FDEDD32211	MULTUW HL,(IY+1122H)
FDEDD42211	DIV HL,(IY+1122H)
FDEDD52211	DIVU HL,(IY+1122H)
FDEDD62211	ADDW HL,(IY+1122H)
FDEDD72211	CPW HL,(IY+1122H)
FDEDD82211	MULT A,(HL+1122H)
FDEDD92211	MULTU A,(HL+1122H)
FDEDDA2211	DIVW DEHL,(IY+1122H)
FDEDDB2211	DIVUW DEHL,(IY+1122H)
FDEDDC2211	DIV HL,(HL+1122H)
FDEDDD2211	DIVU HL,(HL+1122H)
FDEDDE2211	SUBW HL,(IY+1122H)
FDEDE0	MULT A,IYH
FDEDE1	MULTU A,IYH
FDEDE2	MULTW HL,IY
FDEDE3	MULTUW HL,IY
FDEDE4	DIV HL,IYH
FDEDE5	DIVU HL,IYH
FDEDE6	ADDW HL,IY
FDEDE7	CPW HL,IY
FDEDE8	MULT A,IYL
FDEDE9	MULTU A,IYL
FDEDEA	DIVW DEHL,IY
FDEDEB	DIVUW DEHL,IY
FDEDEC	DIV HL,IYL
FDEDED	DIVU HL,IYL
FDEDEE	SUBW HL,IY
FDEDF055	MULT A,(IY+55H)
FDEDF155	MULTU A,(IY+55H)
FDEDF24433	MULTW HL,3344H
FDEDF34433	MULTUW HL,3344H
FDEDF455	DIV HL,(IY+55H)
FDEDF555	DIVU HL,(IY+55H)
FDEDF64433	ADDW HL,3344H
FDEDF74433	CPW HL,3344H
FDEDF866	MULT A,66H
FDEDF966	MULTU A,66H
FDEDFA4433	DIVW DEHL,3344H
FDEDFB4433	DIVUW DEHL,3344H
FDEDFC66	DIV HL,66H
FDEDFD66	DIVU HL,66H
FDEDFE4433	SUBW HL,3344H
FDF22211	JP P,(PC+1122H)
FDF42211	CALL P,(PC+1122H)
FDF54433	PUSH 3344H
FDF9	LD SP,IY
FDF9	LDW SP,IY
FDFA2211	JP M,(PC+1122H)
FDFC2211	CALL M,(PC+1122H)
FE66	CP A,66H
FF	RST 38H

Appendix E.
Instruction Timing

The Z280 CPU processes instructions using a three-stage pipeline consisting of an instruction prefetch unit, an instruction decoder, and an instruction execution unit. Each section of the pipeline operates autonomously, communicating with the other stages of the pipeline via handshakes and local buses. The pipelined architecture of the Z280 MPU greatly increases program throughput; as one instruction is being executed, the next instruction can be decoded, and the instruction after that can be fetched.

The autonomous operation of the three stages in the Z280 CPU instruction pipeline makes it difficult to calculate exact instruction execution times. Furthermore, execution times are affected by cache activity; the current cache contents determine the number of external memory transactions made during the fetch and execution of a given instruction. In this appendix, three types of tables are provided for calculation of instruction timings: instruction execution timing, instruction fetch and decode timing, and bus transaction timing. All tables list execution and transaction timings in terms of CPU clock cycles.

Tables E-1, E-2, and E-3 show the execution times for all instructions and interrupt and trap processing. Table E-1 lists the execution times for all CPU-executed instructions, with the instructions listed by functional group. Table E-2 lists the execution times for the extended instructions. Table E-3 shows execution times for interrupt and trap events. These tables assume that the instruction has been fetched, decoded, and is ready for execution, and that the bus is idle when the execution unit makes a request for a transaction. Thus, the execution times shown in these tables represent the maximum execution rate of the machine. The actual execution rate will be somewhat lower than this maximum for two reasons: (1) the execution unit must compete with the prefetch unit for use of the external bus, and (2) some instructions may take longer to prefetch and decode than the previous instruction will take to execute.

Furthermore, the activity of the execution unit can affect the prefetch unit when certain instructions are executed. In Tables E-1 and E-2, an "F" on the right-hand side of the table indicates that the pipeline is flushed when that instruction is executed; the pipeline is also flushed during all interrupt and trap processing.

In these cases, the next instruction must be completely fetched and decoded before the execution unit can proceed. The execution times in these tables do not include the time necessary to fetch and decode the next instruction if the pipeline is flushed.

In Tables E-1 through E-3, execution times are given as the number of absolute CPU clock cycles plus the number and type of bus transactions. Bus transaction timings are shown separately in Tables E-5 through E-10.

Table E-4 contains the instruction fetch and decode timing, and Tables E-5 through E-10 show bus transaction timings. The CPU clock is divided by a factor of 1, 2, or 4 to form the bus clock; thus, bus transaction timing depends on the relationship between the CPU clock and bus clock. All three types of bus timing are shown in the tables. Furthermore, because of the different phase relationships between the request for a transaction and the bus clock, a variable number of cycles is included in parentheses in Tables E-4 through E-10; the average would be half of the sum of the minimum and maximum numbers listed in the parentheses. The notation "w" in these tables refers to the number of wait states added to the transaction (either by asserting the $\overline{\text{WAIT}}$ input or by programming the appropriate CPU control registers) in addition to any automatically inserted wait states. Again, the numbers in these tables assume that the bus is idle when the transaction request is made.

Table E-1. Instruction Execution Times

Instruction	Addressing Modes	Execution Time
8-BIT LOAD GROUP		
EX A,src	src = R,RX,IR,DA,X,SX RA,SR,BX	R,RX: 4 IR,DA,X,SX,RA,SR,BX: 5 + rd(src) + wr(src)
EX H,L		4
LD dst,src	dst = A src = R,RX,IM,IR,DA,X SX,RA,SR,BX (BC),(DE)	R,RX: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src) (BC),(DE): 3 + rd(IR)
	or	
	dst = R,RX,IR,DA,X SX,RA,SR,BX (BC),(DE) src = A	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + wr(dst) (BC),(DE): 3 + wr(IR)
LD dst,src	dst = R src = R,RX,IM,IR,SX	R,RX,IM: 2 IR,SX: 3 + rd(src)
	or	
	dst = R,RX,IR,SX src = R	R,RX: 2 IR,SX: 3 + wr(dst)
LD dst,n	dst = R,RX,IR,DA,X, SX,RA,SR,BX	R,RX: 2 IR,DA,X,SX,RA,SR,BX: 3 + wr(dst)
LDUD dst,src	dst = A src = IR,SX in user	3 + rd(src)
	or	
	dst = IR,SX in user src = A	3 + wr(dst)
LDUP dst,src	dst = A src = IR,SX in user	3 + rd(src)
	or	
	dst = IR,SX in user src = A	3 + wr(dst)

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
16-BIT LOAD GROUP		
EX src,HL	src = DE,IX,IY	5
EX (SP),dst	dst = HL,IX,IY	5 + rd(IR) + wr(IR)
EX AF,AF'		2
EXX		2
LD[W] dst,src	dst = HL,IX,IY src = IM,DA,X,RA,SR,BX or dst = DA,X,RA,SR,BX src = HL,IX,IY	IM: 2 DA,X,RA,SR,BX: 3 + rd(src) 3 + wr(dst)
LD[W] dst,src	dst = BC,DE,HL,SP src = IM,IR,DA,SX or dst = IR,DA,SX src = BC,DE,HL,SP	IM: 2 IR,DA,SX: 3 + rd(src) 3 + wr(dst)
LD[W] dst,nn	dst = RR,IR,DA,RA	RR: 2 IR,DA,RA: 3 + wr(dst)
LD[W] dst,nn	dst = RR	2
LD[W] dst,src	dst = SP src = HL,IX,IY,IM,IR,DA,SX or dst = IR,DA,SX src = SP	HL,IX,IY,IM: 2 IR,DA,SX: 3 + rd(src) 3 + wr(dst)
LDA dst,src	dst = HL,IX,IY src = DA,X,RA,SR,BX	DA: 2 X,RA,SR: 5 BX: 6
POP dst	dst = RR,IR,DA,RA	RR: 9 + rd(IR) IR,DA,RA: 9 + rd(IR) + wr(dst)
PUSH src	src = RR,IM,IR,DA,RA	RR,IM: 8 + wr(IR) IR,DA,RA: 9 + rd(src) + wr(IR)

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
BLOCK TRANSFER AND SEARCH GROUP		
CPD		8 + rd(IR)
CPDR		8 + rd(IR), each iteration
CPI		8 + rd(IR)
CPIR		8 + rd(IR), each iteration
LDD		9 + rd(IR) + wr(IR)
LDDR		9 + rd(IR) + wr(IR), each iteration
LDI		9 + rd(IR) + wr(IR)
LDIR		9 + rd(IR) + wr(IR), each iteration
8-BIT ARITHMETIC AND LOGIC GROUP		
ADC [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
ADD [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
AND [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
CP [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
CPL [A]		2
DAA [A]		3
DEC dst	dst = R,RX,IR,DA,X, SX,RA,SR,BX	R,RX: 3 IR,DA,X,SX,RA,SR,BX: 4 + rd(dst) + wr(dst)
DIV [HL,]src	src = R,RX,IM,DA,X, SX,RA,SR,BX	R,RX,IM: 46 4 if divide by zero 20 if overflow DA,X,SX,RA,SR,BX: 47 + rd(src) 5 + rd(src) if divide by zero 21 + rd(src) if overflow
DIVU [HL,]src	src = R,RX,IM,DA,X, SX,RA,SR,BX	R,RX,IM: 34 4 if divide by zero 13 if overflow DA,X,SX,RA,SR,BX: 35 + rd(src) 5 + rd(src) if divide by zero 14 + rd(src) if overflow
EXTS [A]		4
INC dst	dst = R,RX,IR,DA,X, SX,RA,SR,BX	R,RX: 3 IR,DA,X,SX,RA,SR,BX: 4 + rd(dst) + wr(dst)
MULT [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 17* IR,DA,X,SX,RA,SR,BX: 18 + rd(src)* *add 1 if src < 0

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
8-BIT ARITHMETIC AND LOGIC GROUP (Continued)		
MULTU [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 17 IR,DA,X,SX,RA,SR,BX: 18 + rd(src)
NEG [A]		3
OR [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
SBC [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
SUB [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
XOR [A,]src	src = R,RX,IM,IR,DA, X,SX,RA,SR,BX	R,RX,IM: 2 IR,DA,X,SX,RA,SR,BX: 3 + rd(src)
16-BIT ARITHMETIC AND LOGIC GROUP		
ADC dst,src	dst = HL src = BC,DE,HL,SP or dst = IX src = BC,DE,IX,SP or dst = IY src = BC,DE,IY,SP	3 3 3
ADD dst,src	dst = HL src = BC,DE,HL,SP or dst = IX src + BC,DE,IX,SP or dst = IY src = BC,DE,IY,SP	3 3 3
ADD dst,A	dst = HL,IX,IY	3
ADDW [HL,]src	src = RR,IM,DA,X,RA	RR,IM: 3 DA,X,RA: 3 + rd(src)
CPW [HL,]src	src = RR,IM,DA,X,RA	RR,IM: 3 DA,X,RA: 3 + rd(src)
DECW dst	dst = RR,IR,DA,X,RA	RR: 3 IR,DA,X,RA: 4 + rd(dst) + wr(dst)
DEC[W] dst	dst = RR	3
DIVUW [DEHL,]src	src = RR,IM,DA,X,RA	RR,IM: 51 4 if divide by zero 13 if overflow DA,X,RA: 52 + rd(src) 5 + rd(src) if divide by zero 14 + rd(src) if overflow

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
16-BIT ARITHMETIC AND LOGIC GROUP (Continued)		
DIVW [DEHL,]src	src = RR,IM,DA,X,RA	RR,IM: 63 4 if divide by zero 20 if overflow DA,X,RA: 64 + rd(src) 5 + rd(src) if divide by zero 21 + rd(src) if overflow
EXTS HL		4
INCW dst	dst = RR,IR,DA,X,RA	RR: 3 IR,DA,X,RA: 4 + rd(dst) + wr(dst)
INC[W] dst	dst = RR	3
MULTUW [HL,]src	src = RR,IM,DA,X,RA	RR,IM: 24* DA,X,RA: 25 + rd(src)* *add 1 if src < 0
MULTW [HL,]src	src = RR,IM,DA,X,RA	RR,IM: 24 DA,X,RA: 25 + rd(src)
NEG HL		3
SBC dst,src	dst = HL src = BC,DE,HL,SP or dst = IX src = BC,DE,IX,SP or dst = IY src = BC,DE,IY,SP	3 3 3
SUBW [HL,]src	src = RR,IM,DA,X,RA	RR,IM: 3 DA,X,RA: 3 + rd(src)

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
BIT MANIPULATION, ROTATE AND SHIFT GROUP		
BIT b,dst	dst = R,IR,SX	R: 2 IR,SX: 3 + rd(dst)
RES b,dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
RL dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
RLA		2
RLC dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
RLCA		2
RLD		5 + rd(IR) + wr(IR)
RR dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
RRA		2
RRC dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
RRCA		2
RRD		5 + rd(IR) + wr(IR)
SET b,dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
SLA dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
SRA dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
SRL dst	dst = R,IR,SX	R: 2 IR,SX: 4 + rd(dst) + wr(dst)
TSET dst	dst = R,IR,SX	R: 3 IR,SX: 1 + rd(dst) + wr(dst)

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time	
PROGRAM CONTROL GROUP			
CALL cc,dst	dst = IR,DA,RA	cc not true: 3	
		IR,DA: 11 + wr(IR)	F*
		RA: 12 + wr(IR)	F
CALL dst	dst = IR,DA,RA	IR,DA: 11 + wr(IR)	F
		RA: 12 + wr(IR)	F
CCF		2	
DJNZ dst	dst = RA	B is zero: 6	
		B is non-zero: 7	F
JAF dst	dst = RA	AF' not in use: 3	
		AF' in use: 4	F
JAR dst	dst = RA	Alternate file not in use: 3	
		Alternate file in use: 4	F
JP cc,dst	dst = IR,DA,RA	cc not true: 3	
		cc true: 4	F
JP dst	dst = IR,DA,RA	4	F
JR cc,dst	dst = RA	cc not true: 3	
		cc true: 4	F
JR dst	dst = RA	4	F
RET		9 + rd(IR)	F
RET cc		cc not true: 3	
		cc true: 9 + rd(IR)	F
RST dst	dst = DA	9 + wr(IR)	F
SC nn		1 + System Call Trap	
SCF		2	

*"F" indicates that the pipeline is flushed when that instruction is executed.

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time
INPUT/OUTPUT INSTRUCTION GROUP		
IN dst,(C)	dst = R,RX,DA,X,RA,SR,BX	R,RX: 3 + in( ) DA,X,RA,SR,BX: 4 + in( ) + wr(dst)
IN A,(n)		5 + in( )
IN[W] HL,(C)		3 + in( )
IND		8 + in( ) + wr(IR)
INDW		8 + in( ) + wr(IR)
INDR		8 + in( ) + wr(IR), each iteration
INDRW		8 + in( ) + wr(IR), each iteration
INI		8 + in( ) + wr(IR)
INIW		8 + in( ) + wr(IR)
INIR		8 + in( ) + wr(IR), each iteration
INIRW		8 + in( ) + wr(IR), each iteration
OUT (C),src	src = R,RX,DA,X,RA,SR,BX	R,RX: 3 + out( ) DA,X,RA,SR,BX: 3 + rd(src) + out( )
OUT (n),A		5 + out( )
OUT[W] (C),HL		3 + out( )
OUTD		8 + rd(IR) + out( )
OUTDW		8 + rd(IR) + out( )
OTDR		8 + rd(IR) + out( ), each iteration
OTDRW		8 + rd(IR) + out( ), each iteration
OUTI		8 + rd(IR) + out( )
OUTIW		8 + rd(IR) + out( )
OTIR		8 + rd(IR) + out( ), each iteration
OTIRW		8 + rd(IR) + out( ), each iteration
TSTI (C)		3 + in( )

See Table E-1 Note on page E-10.

Table E-1. Instruction Execution Times (Continued)

Instruction	Addressing Modes	Execution Time	
CPU CONTROL GROUP			
DI mask	mask = Hex value	3 + out(I)	
EI mask	mask = Hex value	3 + out(I)	
HALT		11 + rd(halt) minimum	
IM p	p = 0,1,2,3	3	
LD dst,src	dst = A src = I,R	2	
LD dst,src	dst = I,R src = A	2	
LDCTL dst,src	dst = (C),USP src = HL,IX,IY or dst = HL,IX,IY src = (C),USP	(C): 4 + out(I) USP: 2 (C): 3 + in(I) USP: 2	F*
NOP		2	
PCACHE		2	F
RETI		Z-BUS: 8 + rd(IR) Z80: 8 + rd(reti) + rd(IR)	F F
RETIL		14 + 2*rd(IR) + out(I)	F
RETN		7 + rd(IR)	F

*"F" indicates that the pipeline is flushed when that instruction is executed.

NOTES:

1. This table assumes that the instruction has been fetched, decoded, and is ready for execution. The execution time for instructions that cause the pipeline to be flushed do not include the time necessary to fetch and decode the following instruction.
2. This table assumes that the $\overline{\text{PAUSE}}$ input is inactive. If $\overline{\text{PAUSE}}$ is active, the execution unit will wait before beginning the next instruction.
3. The bus is assumed to be idle when the execution unit makes a request for a transaction.
4. This table assumes that no exceptions occur during instruction execution except where indicated.

Table E-2. Extended Instruction Execution Times

Instruction	Addressing Modes	Execution Time	
EXTENDED INSTRUCTION GROUP TEMPLATE FETCH (EPU ENABLE BIT SET TO 1)			
Aligned template		7 + epu(if1) + epu(ifn) + out(I)	
Unaligned template		7 + epu(if1) + 2*epu(ifn) + out(I)	
EXTENDED INSTRUCTION GROUP			
EPUI (Internal Operation)		4 + p	F*
EPUF (CPU←EPU)		6 + p + epu(cpu)	F
MEPU dst (Memory←EPU)	dst = IR,DA,X,RA,SR,BX	5 + p + k*[3 + epu(wr)]	F
EPUM src (EPU←Memory)	src = IR,DA,X,RA,SR,BX	5 + p + k*[3 + epu(rd)]	F

*"F" indicates that the pipeline is flushed when that instruction is executed.

NOTES:

1. Additional cycles are necessary for address computation in the case of EPU-to-Memory and Memory-to-EPU instructions, as shown below.

IR,DA	no additional cycles
X,SX,RA,SR	1 additional cycle
BX	2 additional cycles

2. The notation "p" in the table is the number of pause cycles added to the bus cycle.
3. The notation "k" in the table is a function of n, the number of bytes to be transferred that is specified in the template, and the address of the source or destination as shown below.

n is odd	k = (n + 1)/2
n is even and aligned	k = n/2
n is even and unaligned	k = (n = 2)/2

4. See "Notes" from Table E-1.

Table E-3. Interrupt, Trap, and Special Condition Execution Times

Type	Execution Time
INTERRUPTS	
NMI in Modes 0, 1, 2	13 + iack(nmi012) + in(I) + out(I) + wr(IR)
Mode 0	9 + out(I) + [iack(m0) for each byte of opcode]
Mode 1	13 + iack(m1) + in(I) + wr(IR) + out(I)
Mode 2	16 + iack(m2) + in(I) + wr(IR) + rd(IR) + out(I)
Mode 3 Nonvectored	28 + iack(m3) + in(I) + 3*wr(IR) +2*rd(IR) + out(I)
Mode 3 Vectored	31 + iack(m3) + in(I) + 3*wr(IR) +2*rd(IR) + out(I)
On-Chip (Mode 3)	28 + iack(m3) + in(I) + 3*wr(IR) +2*rd(IR) + out(I)
TRAPS	
Single-Step	26 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
Breakpoint-on-Halt	26 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
Division Exception	25 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
Stack Overflow Warning	26 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
Access Violation	25 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
System Call	30 + in(I) +3*wr(IR) + 2*rd(IR) + out(I)
Privileged Instruction	26 + in(I) +2*wr(IR) + 2*rd(IR) + out(I)
EPU ← Memory	38 + in(I) +4*wr(IR) + 2*rd(IR) + out(I)
Memory ← EPU	38 + in(I) +4*wr(IR) + 2*rd(IR) + out(I)
A ← EPU	31 + in(I) +3*wr(IR) + 2*rd(IR) + out(I)
EPU Internal Operation	31 + in(I) +3*wr(IR) + 2*rd(IR) + out(I)
MISCELLANEOUS	
FATAL	15 + out(I) + rd(halt) minimum
RESET	3 + rd(reset) + out(I) minimum
EPU Data Page Fault	1 + epu(if1) and then Access Violation trap

NOTES:

1. Additional cycles are necessary for address computation in the case of EPU-to-Memory and Memory-to-EPU traps, as shown below.

IR,DA	no additional cycles
X,SX,RA,SR	1 additional cycle
BX	2 additional cycles

2. The pipeline is flushed at the end of any interrupt or trap sequence.

Table E-4. Instruction Fetch and Decode Timing

Condition	1 x Bus Timing	2 x Bus Timing	4 x Bus Timing
First byte, cache	4	4	4
First byte, external	9 + w	12 + 2w + (0-1)	17 + 4w + (0-3)
First byte, burst	12 + w	18 + 2w + (0-1)	29 + 4w + (0-3)
Subsequent byte, cache	1	1	1
Subsequent byte, external	5 + w	8 + 2w + (0-1)	13 + 4w + (0-3)
Subsequent byte, burst	8 + w	14 + 2w + (0-1)	25 + 4w + (0-3)

NOTES:

1. The term "first" means the first byte fetched following a flushed pipeline. All other bytes are "subsequent". With a full pipeline, only the execution times are necessary.
2. With a 16-bit external bus, the prefetch unit tries to fetch words from external memory though bytes are transferred to the pipeline. Bytes other than the one requested are placed in cache.
3. A burst transfer transfers a four-word block starting with the word with the three least significant bits being zero. The appropriate byte is transferred to the decoder as it is written to the cache. The execution unit of the pipeline can begin execution prior to the burst transaction completion if the necessary bytes are fetched during the early part of the burst transaction.
4. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
5. The notation "w" in the transaction tables is the number of WAIT states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.
6. Examples of instruction fetch/decode time (assuming flushed pipeline and 1 x bus timing):
   a) Two-byte instruction in cache — [4 + 1] processor cycles
   b) Two-byte instruction both bytes not in cache — [(9 + w) + (5 + w)]
   c) Two-byte instruction, first byte in cache, second not in cache — [4 + (5 + w)]
   d) Four-byte instruction in cache — [4 + 1 + 1 + 1] processor cycles
   e) Four-byte instruction not in cache, no burst, not cacheable — [9 + w + 3 • (5 + w)] processor cycles
   f) Four-byte instruction not in cache, burst, cacheable — [12 + w + 1 + 1 + 1]
   g) Six-byte instruction, burst, first two bytes in cache — [4 + 1 + (8 + w) + 1 + 1 + 1]

Table E-5. Data Read Timing — rd(src), rd(dst), and rd(IR)

Condition	1× Bus Timing	2× Bus Timing	4× Bus Timing
Byte Hit	5	5	5
Byte Miss	8 + w	11 + 2w + (0–1)	16 + 4w + (0–3)
Aligned Word Hit	5	5	5
Aligned Word Miss	8 + w	11 + 2w + (0–1)	16 + 4w + (0–3)
Unaligned Word Hit Hit	9	9	9
Unaligned Word Miss Hit	12 + w	15 + 2w + (0–1)	20 + 4w + (0–3)
Unaligned Word Hit Miss	12 + w	15 + 2w + (0–1)	20 + 4w + (0–3)
Unaligned Word Miss Miss	15 + w	21 + 2w + (0–2)	31 + 4w + (0–6)
TSET (cache)	8 + w	11 + 2w + (0–1)	16 + 4w + (0–3)
TSET (fixed memory)	6	6	6
Page Fault	4 + Access Violation trap	4 + Access Violation trap	4 + Access Violation trap

NOTES:

1. Additional cycles are necessary for address computation, as shown below.

IR,DA	no additional cycles
X,SX,RA,SR	1 additional cycle
BX	2 additional cycles

2. A word is aligned if the address is even and the transfer is over a 16-bit bus. It is otherwise unaligned.
3. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
4. The notation "w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Table E-6. Data Write Timing — wr(src), wr(dst), and wr(IR)

Condition	1× Bus Timing	2× Bus Timing	4× Bus Timing
Byte	5	5	5
Aligned Word	5	5	5
Unaligned Word	9 + w	12 + 2w + (0–1)	17 + 4w + (0–3)
Page Fault	4 + Access Violation trap	4 + Access Violation trap	4 + Access Violation trap

NOTES:

1. Additional cycles are necessary for address computation, as shown below.

IR,DA	no additional cycles
X,SX,RA,SR	1 additional cycle
BX	2 additional cycles

2. A word is aligned if the address is even and the transfer is over a 16-bit bus. It is otherwise unaligned.
3. The pipeline is flushed whenever a byte being written is valid in the cache.
4. In the unaligned word case where the first byte is valid in cache, the execution time is 10 cycles with zero or one wait states and 9 + w cycles for two or more wait states.
5. The number in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
6. The notation "w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Table E-7. I/O Read and Write Timing

Type	1 x Bus Timing	2 x Bus Timing	4 x Bus Timing
in(I)	5	5	5
in( )	9 + w	13 + 2w + (0-1)	20 + 4w + (0-3)
wr(I)	5	5	5
wr( )	5	5	5

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation "w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.
3. in(I) and wr(I) are performed internally within the Z280 MPU.

Table E-8. EPU Read and Write Timing

Type	1 x Bus Timing	2 x Bus Timing	4 x Bus Timing
epu(if1)	8 + w	11 + 2w + (0-1)	16 + 4w + (0-3)
epu(ifn)	8 + w	11 + 2w + (0-1)	16 + 4w + (0-3)
epu(cpu)	9 + w	13 + 2w + (0-1)	20 + 4w + (0-3)
epu(wr)	10 + w	15 + 2w + (0-1)	24 + 4w + (0-3)
epu(rd)	8 + w	11 + 2w + (0-1)	16 + 4w + (0-3)

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation "w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.

Table E-9. Interrupt Acknowledge Timing

Type	1 x Bus Timing	2 x Bus Timing	4 x Bus Timing
iack(nmi012)	4	4	4
iack(m0)	8 + w	13 + 2w + (0-1)	22 + 4w + (0-3)
iack(m1)	10 + w	15 + 2w + (0-1)	24 + 4w + (0-3)
iack(m2)	10 + w	15 + 2w + (0-1)	24 + 4w + (0-3)
iack(m3)	10 + w	15 + 2w + (0-1)	24 + 4w + (0-3)

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation "w" in the transaction tables is the number of wait states added to the bus cycle that are either externally generated or programmably added. Wait states that are an integral part of the transaction (e.g., one wait state for I/O transactions) should not be included.
3. iack(nmi012) is for NMI in modes 0, 1, and 2.
   iack(m0) is for mode 0 interrupts.

Table E-10. Miscellaneous Transaction Timing

Type	1× Bus Timing	2× Bus Timing	4× Bus Timing
HALT Transaction	5	5	5
RESET Transaction	6	6	6
RETI Transaction	21 + w	31 + 2w + (0–2)	49 + 4w + (0–6)

NOTES:

1. The numbers in parentheses depend on the phase relationship between the transaction request and the bus clock. The average will be half of the sum of the minimum and maximum numbers in parentheses.
2. The notation "w" in the transaction tables is the number of WAIT states added to the bus cycle in addition to any automatically inserted WAIT states. This includes any WAITs added under program control.

Appendix F.
Compatible Peripheral Families

The Z280 MPU supports two different types of bus interface: the Z80-Bus and the Z-BUS. Families of peripheral devices are available for both types of component interconnect buses.

The Z80 Bus configurations of the Z280 MPU have two compatible peripheral families: the Z8400 and Z8000/Z8500 peripheral families (Tables F-1 and F-2). The Z8400 family of devices were originally designed to support the Z80-Bus. The Z8000 series of peripherals are designed for systems employing multiplexed address/data buses, and are also easily interfaced to Z80-Bus Z8000 MPU systems. The Z8500 peripheral family is identical to the Z8000 family, except the devices are configured to interface to non-multiplexed buses: the Z8500 series devices can be used in Z280 MPU systems where the address/data bus is de-multiplexed external to the processor.

The Z-BUS versions of the Z280 MPU are supported by the Z8000/Z8500 peripheral family (Table F-2). These devices interface directly to the Z-BUS.

Refer to the Zilog Components Data Book for further information regarding these peripheral families.

Table F-1. Z8400 Peripheral Family

Part Number	Description
Z8410	DMA Direct Memory Access Controller
Z8420	PIO Parallel Input/Output Controller
Z8430	CTC Counter/Timer Circuit
Z8440/1/2	SIO Serial Input/Output Controller
Z8470	DART Dual Asynchronous Receiver/Transmitter

Table F-2. Z8000/Z8500 Peripheral Family

Part Number	Description
Z8016/Z8516	DTC Direct Memory Access Transfer Controller
Z8030/Z8530	SCC Serial Communications Controller
Z8036/Z8536	CIO Counter/Timer and Parallel I/O Unit
Z8038	Z-FIO FIFO Input/Output Interface Unit
Z8060	Z-FIFO Buffer Unit and Z-FIO Expander
Z8065	BEP Burst Error Processor
Z8068	Z-DCP Data Ciphering Processor
Z8090/Z8590	UPC Universal Peripheral Controller (ROM-based)
Z8094/Z8594	UPC Universal Peripheral Controller (RAM-based)

Glossary

access protection: A function of memory management that controls read, write, and execute access to memory locations, protecting proprietary or operating system memory areas from tampering by unauthorized users.

access protection violation: An incorrect or forbidden attempt to access a memory location; for example, an attempt to write to a read-only page. An access violation causes the CPU to abort the current instruction and generate an Access Violation trap.

accumulator: A register within a central processing unit (CPU) that can hold the result of an arithmetic or logical operation.

address space: A set of addresses that are accessed in a similar manner. The Z280 MPU contains four types of address spaces: CPU register, CPU control register, memory, and I/O. The memory space can be divided into four separate memory address spaces: system-mode program, system-mode data, user-mode program, and user-mode data.

addressing mode: The way in which the location of an operand is specified. There are nine addressing modes in the Z280 MPU: Register, Immediate, Register Indirect, Direct Address, Indexed, Short Index, Base Index, Relative Address, and Stack Pointer Relative.

address tag: The portion of certain associative memories that is compared against a referenced address to determine whether the matching value is found. The address tag for a cache block is the physical memory address.

address translation: The process of mapping logical addresses into physical addresses.

aligned address: An address that is a multiple of an operand's size in bytes. Aligned word addresses are a multiple of two.

associative memory: A memory in which data is accessed by specifying a value rather than a location. The cache is an associative memory.

autodecrement: The operation of decrementing an address in a register by the operand's size in bytes. The decrement amount is one for byte operands, two for word operands.

autoincrement: The operation of incrementing an address in a register by the operand's size in bytes. The increment amount is one for byte operands, two for word operands.

base address: The address used, along with an index and/or displacement value, to calculate the effective address of an operand. The base address is located in a register, the Program Counter, or the instruction.

Base Index (BX) addressing mode: In this mode, the contents of the base register and index register are added to obtain the effective address.

burst transaction: The transfer of several consecutive items of data in one memory transaction.

bus master: The device in control of the bus.

bus request: A request for control of the bus initiated by a device other than the bus master.

byte: A data item containing eight contiguous bits. A byte is the basic data unit for addressing memory and peripherals.

cache: An on-chip buffer that automatically stores copies of recently used memory locations (both instructions and data), allowing fast access for memory fetches.

Central Processing Unit (CPU): The primary functioning unit of a computer, consisting of an ALU, control logic for decoding and executing instructions and controlling program flow, and registers.

coprocessor: A processor that works synchronously with the CPU to execute a single instruction stream using the Extended Processing Architecture (EPA).

destination: The register, memory location, or device to which data are to be transferred.

Direct Address (DA) addressing mode: In this mode, the effective address is contained in the instruction.

displacement: A constant value located in the instruction that is used for calculating the effective address of an operand.

effective address: The logical memory address of an operand, calculated by adding the base address, an optional index value, and an optional displacement.

EPU internal operation: An EPU-handled operation that controls EPU operations but does not transfer data.

exception: A condition or event that alters the usual flow of instruction processing. The Z280 MPU supports three types of exception: reset, interrupts, and traps.

Extended Processing Architecture (EPA): A CPU facility that allows the operations defined in the architecture to be extended by hardware or software. If enabled, the CPU transfers EPA instructions to an Extended Processing Unit (EPU) for execution; if disabled, the CPU traps EPA instructions for software emulation.

Extended Processing Unit (EPU): An external device, that handles Extended Processing Architecture instructions (such as floating-point arithmetic).

flowthrough transaction: A DMA-initiated data transfer consisting of separate read and write transactions. Data is temporarily stored in the DMA channel between the read and write transactions.

flyby transaction: A transaction controlled by the bus master, but in which another device transfers data to the responding device.

frame: A block of physical memory used by the memory management mechanism to map logical memory pages.

global bus: A bus shared by tightly coupled, multiple CPUs; the bus master is chosen by an external arbiter device.

hit: A hit occurs when a associative memory is searched for a value and a match is found.

identifier word: A 16-bit code saved on the system stack during exception processing that provides information about the cause of the exception.

Immediate (IM) addressing mode: In this mode, the operand is contained in the instruction.

index: A value located in a register used for calculating the effective address of an operand. The index value usually specifies the calculated offset of an operand from the orgin of an array or other data structure.

Indexed (X) addressing mode: In this mode, the contents of an index register are added to a base address contained in the instruction to obtain the effective address.

Indirect Register (IR) addressing mode: In this mode, the effective address is contained in a register.

interrupt: An asynchronous exception that occurs when an NMI or INT line is activated, usually when a peripheral device needs attention.

least recently used (LRU): The CPU records the order of use for cache blocks. When a tag miss occurs, the CPU replaces the least recently used block.

local bus: The bus controlled by the CPU and shared with slave processors.

logical address: The address manipulated by the program. The memory management mechanism translates logical addresses to physical addresses.

loosely coupled CPUs: CPUs that execute independent instruction streams and communicate through a multi-ported peripheral, such as a Z8038 FIO I/O interface unit.

Master Status register: A 16-bit CPU control register that contains status information describing the current operating states of the CPU.

memory management: The process of translating logical addresses into physical addresses, plus certain protection functions. In the Z280 MPU, memory management is integrated into the chip.

memory-mapped I/O: An I/O device accessed in the memory address space.

miss: A miss occurs when an associative memory is searched for a value and no match is found.

nonmaskable interrupt: The highest priority interrupt; cannot be disabled.

page: A logical memory unit mapped by the memory management mechanism to a physical memory frame.

paged translation: A method of address translation in which the logical and physical address spaces are divided into fixed, equal-sized units called pages and frames, respectively. During address translation, a logical page is mapped to an arbitrary physical frame.

physical address: The 24-bit address required for accessing memory and peripherals, obtained by the CPU's address translation hardware.

pipeline: A computer design technique in which an instruction is executed in a sequence of stages by different functional units. The functional units can be operating on several different instructions simultaneously, similiar to an automobile assembly line.

prefetching: Ability of the CPU to fetch an instruction or operand before the previous instructions have been completed.

privileged instruction: An instruction that performs I/O operations, accesses control registers, or performs some other operating system function. Privileged instructions execute in system mode only.

Program Counter (PC): One of the two Program Status registers; it contains the address of the current instruction.

Program Status registers: The two registers (Program Counter and Master Status register) that contain the Program Status. The Program Status is automatically saved during exception processing.

protection: See access protection.

read access: The type of memory access used by the CPU for fetching data operands other than those specified by Immediate addressing mode.

refresh: To restore information that fades away if left alone. For example, dynamic memories must be refreshed periodically in order to retain their contents.

Register (R) addressing mode: In this mode, the operand is in a general-purpose register.

Relative Address (RA) addressing mode: In this mode, the displacement in the instruction is added to the contents of the Program Counter to obtain the effective address.

relocation: The process of mapping a logical address to a different physical address, so that multiple processes can use the same logical address for distinct physical memory locations.

request: A signal or message used by a device to indicate the need for some action or resource.

reset: A CPU operating state or exception that results when a reset request is signaled on the RESET line.

responder: The device to which bus transactions transfer data.

self-modifying program: A program that stores to a location from which a subsequent instruction is fetched.

semaphore: A storage location used as a Boolean variable to synchronize the use of resources among multiple programming tasks. A semaphore ensures that a shared resource is allocated to only one task at any given time.

service routine: Program code that is executed in response to an interrupt or trap.

Short Index addressing mode: In this mode, the contents of the IX or IY register are added to an 8-bit displacement contained in the instruction to obtain the effective address of the operand.

slave processor: A processor, such as a Direct Memory Access transfer controller, that performs dedicated functions asynchronously to the CPU.

source: The register, memory location, or device from which data are being read.

spatial locality: The characteristic of program behavior whereby consecutive memory references often apply to closely located addresses.

stack: An area of memory used for temporary storage and subroutine linkages. A stack uses the first-in, last-out method for storing and retrieving data; the last data written onto the stack will be the first data read from the stack.

Stack Pointer (SP): A register indicating the top (lowest address) of the processor stack used by Call and Return instructions for linking procedures. User and system modes of operation use separate Stack Pointers, the User Stack Pointer (USP) and System Stack Pointer (SSP).

system mode: A CPU mode of operation, used for operating system functions. In this mode, the CPU can execute privileged (and all other) instructions.

System Stack Pointer (SSP): The Stack Pointer used while the CPU is in system mode. User mode programs cannot access the SSP.

tag hit: On a memory reference, a tag hit occurs when the cache address tags are searched for the referenced address and a match is found.

tag miss: On a memory reference, a tag miss occurs when the cache address tags are searched for the referenced address and no match is found.

temporal locality: The characteristic of program behavior whereby memory references often apply to a location that has been referred to recently.

tightly coupled CPUs: CPUs that execute independent instruction streams and communicate through shared memory on a common (global) bus.

transaction: A basic bus operation involving the transfer of one byte or word of data between the CPU and a memory or peripheral device.

trap: An exception that occurs when certain conditions, such as an access protection violation, are detected during execution of an instruction.

unaligned address: An address that is not a multiple of an operand's size in bytes. Odd addresses are unaligned for words.

user mode: A CPU mode of operation, generally used for application programs. In this mode, the CPU cannot execute privileged instructions or access protected memory locations.

User Stack Pointer (USP): The Stack Pointer used while the CPU is in user mode. System mode programs can access the USP with the Load Control instruction.

vectored interrupt: A interrupt that uses the low-order byte of the identifier word as a vector to an interrupt service routine; can be disabled.

virtual memory: A memory management technique in which the system's logical memory address space is not necessarily the same as, and can be much larger than, the available physical memory.

wait state: A clock period that is added to a memory or I/O transaction due to an active WAIT signal. Wait states are used to prolong memory and I/O transactions to devices with long access times.

word: A data item containing sixteen contiguous bits.

write access: The type of memory access used by the CPU for storing data operands.

Index

NOTES

NOTES

Zilog

READER COMMENTS

Your comments concerning this publication are important to us. Please take the time to complete this questionnaire and return it to Zilog.

Title of Publication: _____

Document Number: _____

Your Hardware Model and Memory Size: _____

Describe your likes/dislikes concerning this document:

Technical Information: _____

Supporting Diagrams: _____

Ease of Use: _____

Your Name: _____

Company and Address: _____

Your Position/Department: _____

03-8224-01

Scanned by

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA
Agoura 818–707–2160
Campbell 408–370–8016
Costa Mesa 714–261–1281

COLORADO
Boulder 303–494–2905

FLORIDA
Largo 813–585–2533

GEORGIA
Atlanta 404–451–8425

ILLINOIS
Schaumburg 312–885–8080

MASSACHUSETTS
Burlington 617–273–4222

MINNESOTA
Edina 612–831–7611

NEW JERSEY
Hasbrouck Heights 201–288–3737
Mt. Laurel 609–778–8070

OHIO
Seven Hills 216–447–1480

TEXAS
Richardson 214–231–9090

INTERNATIONAL SALES OFFICES

CANADA
Toronto 416–673–0634

GERMANY
Munich 49–89–612–6046

JAPAN
Tokyo 81–3–587–0528

HONG KONG
Kowloon 852–3–723–8979

R.O.C.
Taiwan 886–2–731–2420

UNITED KINGDOM
Maidenhead 44–628–39200

Zilog, Inc. 210 Hacienda Ave., Campbell, California 95008-6609
Telephone (408)370-8000 TWX 910-338-7621

03–8224–02 Printed in USA