

TABLE OF CONTENTS

DEMON MONITOR CHECKOUT ..... 1  
CHECKOUT INSTRUCTIONS ..... 3

APPENDICES

- I. HOW TO HAND-ASSEMBLE JOLT PROGRAMS
- II. NOTES, HINTS AND RECOMMENDATIONS FOR USING YOUR JOLT MICROCOMPUTER
- III. DETAILED DESCRIPTION OF DEMON
- IV. THE MCS 6502 INSTRUCTION SET
- V. CHART OF BRANCHES: DECIMAL TO HEXADECIMAL
- VI. LISTING OF DEMON MONITOR
- VII. LISTINGS OF DIAGNOSTIC PROGRAMS
  - SYSTEM EXERCISER
  - MEMORY ADDRESS TEST

### DEMON MONITOR CHECKOUT

You are now ready to check out your JOLT DEMON monitor. The instructions which follow assume that your JOLT is connected to a suitable power supply and a teletype or other serial computer terminal. A detailed description of the DEMON monitor is in Appendix III. Here is a summary of its features:

DEMON is the DEbug MONitor program for the JOLT Microcomputer. It is supplied in read-only memory (ROM) as part of the 6530 multi-function chip on the JOLT CPU board. Because the DEMON code is non volatile, it is available at system power-on and cannot be destroyed inadvertently by user programs. Furthermore, the user is free to use only those DEMON capabilities which he needs for a particular program. Both interrupt types, interrupt request (IRQ) and non-maskable interrupt (NMI) may be set to transfer control to DEMON or directly to the user's program.

DEMON communicates with the user via a serial full-duplex port (using ASCII codes) and automatically adjusts to the speed of the user's terminal. Any speed--even non-standard ones--can be accommodated. If the user's terminal has a long carriage return time, DEMON can be set to perform the proper delay. Commands typed at the terminal can direct DEMON to start a program, display or alter registers

and memory locations, set breakpoints, and load or punch programs. If available in the system configuration, a high-speed paper tape reader may be used to load programs through a parallel port on the 6530 chip. Programs may be punched in either of two formats--hexadecimal (assembler output) or BNPF (which is used for programming read-only memories). On loading or modifying memory, DEMON performs automatic read-after-write verification to insure that addressed memory exists, is read/write type, and is responding correctly. Operator errors and certain hardware failures may thus be detected using DEMON.

DEMON also provides several subroutines which may be called by user programs. These include reading and writing characters on the terminal, typing a byte in hexadecimal, reading from high-speed paper tape, and typing a carriage-return, line-feed sequence with proper delay for the carriage of the terminal being used. Program tapes loaded by DEMON may also specify a start address so that programs may be started with a minimum of operator action.

CHECKOUT INSTRUCTIONS

( ) 1. Turn power on, or if the power is on, perform a RESET operation. Type a carriage-return on the terminal. DEMON should respond with:

\* 7052 30 18 FF 01 FF

(Exact values may vary, although the first and last values should be as shown). If no response or a garbled response occurs, RESET and try again. In case of continued trouble, refer to the diagnostic section of the CPU Assembly Manual.

The "\* 7052 30 18 FF 01 FF" printout is DEMON's standard breakpoint message format. It consists of an asterisk "\*" to identify the breakpoint printout, followed by the CPU register contents in this order: PC, P, A, X, Y, and S, i.e., Program Counter, Processor Status, Accumulator, X index, Y index and Stack Pointer. Note that all DEMON inputs and outputs are in base 16 which is referred to as hexadecimal, or just hex. In hexadecimal, the "digits" are 0,1,2...,A,B,C,D,E,F. After printing the CPU registers, DEMON is ready to receive commands from you, the operator. DEMON indicates this "ready" status by typing the prompting character "." on a new line.

( ) 2. DEMON's response to RESET is to wait for a carriage-return and then print the user's registers. DEMON uses this carriage return-character to measure the terminal line speed. If you have a settable-rate terminal, change the



rate (any speed between 10 and 30 cps will work) and repeat Step 1. DEMON should respond at the new terminal speed.

( ) 3. The user's CPU registers may also be displayed with the R command. Type an R. The monitor should respond as above, but without the asterisk. Presence of the asterisk indicates that an interrupt or break instruction caused the printout.

```
.R 7052 30 18 FF 01 FF
```

( ) 4. Displayed values may be modified using the Alter (: ) command. To modify register contents, type a colon (: ) followed by the new values. For example:

```
.R 7052 30 18 FF 01 FF  
.: 0100 00 00 00 00 FF  
.R 0100 00 00 00 00 FF
```

Notice that DEMON automatically types spaces to separate data fields. (Note: Characters typed by you, the user, are underlined in this document for clarity. Everything else is typed by the computer.) Examine your registers (R command) to verify the changes.

Memory may be examined and modified, as above, using the M and : commands. Try this:

```
.M 0100 00 66 23 EE 01 A2 41 6E
```

The memory command (M) causes DEMON to type the contents of the first eight bytes of memory. (Memory data will be random on startup). Alter and verify these bytes using the Alter command, as above:

```
.M  0100  00  66  23  EE  01  A2  41  6E
.:  0100  00  01  02  03  04  05  06  07
```

If only part of a line is to be altered, items to be left unchanged can be skipped over by typing blanks, and carriage-return (↵). Try this:

```
.M  0100  00  01  02  03  04  05  06  07
.:  0100  FF  ___  FF  FF  ↵
.M  0100  FF  01  FF  FF  04  05  06  07
```

( ) 5. Try to alter a location in DEMON ROM:

```
.M  7000  85  F9  A9  23  D0  58  A9  16
.:  7000  00?
```

DEMON verifies all changes to memory. Since locations 7000 through 7007 are in read-only memory, alteration is not possible. DEMON signals write failure with a question mark. Similarly, the monitor will notify you of an attempt to alter a non-existent location:

```
.M  9000  90  90  90  90  90  90  90  90
.:  9000  00?
```

Note that attempts to read non-existent memory will normally yield the high-order byte of the address read.

( ) 6. There are three hardware facilities which may be used to stop a running (or run-away) program without the program itself calling DEMON. These are the hardware inputs RESET,

IRQ, and NMI. To test this feature enter the following program at location 0000:

<u>location</u>	<u>contents</u>	<u>instruction</u>
0000	4C	LOOP JMP LOOP
0001	00	
0002	00	

(Use the M and : commands.)

Now, set the program counter (PC) to this location using the R and : commands. Finally, tell DEMON to start executing your program using the GO (G) command:

```
.M 0000 FF 11 11 11 91 91 71 91
.: 0000 4C 00 00 ↓
.M 0000 4C 00 00 11 91 91 71 91
.R 0000 30 00 00 00 FF
.: 0000 ↓
.G
```

The computer should now be executing the program. It will continue to run until interrupted. Using the interrupt request line (IRQ), interrupt the processor. It should respond with:

```
* 0000 30 00 00 00 FF
```

Try the same experiment with non-maskable interrupt (NMI). The result should be the same except for a "#" character preceding, which identifies the NMI printout. Finally, try it with RESET. RESET, however, forces JOLT to branch to DEMON, losing the old PC and other register contents. Thus NMI is the preferred means for manually interrupting program execution. IRQ may also be

used unless it is required for other functions such as peripheral interrupts.

( ) 7. Use M and : to enter the following test program called CHSET because it prints the character-set on the terminal.

Note that Alter (:) commands may be repeated without intervening M commands to set sequential locations:

```
                ;CHECKOUT PROGRAM -- PRINT THE CHARACTER SET ON USER TERMINAL
                CRLF  = $728A                ;ADDRESS OF DEMCN CRLF ROUTINE
                WRT   = $72C6                ;ADDRESS OF DEMCN WRITE ROUTINE
                ;
0000            * = C                        ;VARIABLE STORAGE IN PAGE ZERO
                CHAR  * = * + 1            ;STORAGE FOR CHARACTER
                ;
0001            * = $0100                    ;PROGRAM STARTS ON PAGE ONE
                ;
0100 20 8A 72  CHSET JSR CRLF                ;DO CARRIAGE RETURN & LINE FEED
0103 A9 20                LDA #$20          ;FIRST CHAR IS A SPACE
0105 85 00                STA CHAR          ;INITIALIZE
                ;
0107 A5 00                LCCP  LDA CHAR     ;GET CHARACTER
0109 C9 60                CMP  #$6C        ;CHECK FOR LIMIT
010B F0 08                BEQ  DONE        ;DONE IF 60
                ;
010D 20 C6 72                JSR WRT        ;PRINT CHAR
0110 E6 00                INC CHAR        ;NEXT CHAR CCDE
0112 4C C7 01                JMP LCCP      ;CONTINUE
                ;
0115 00                DONE BRK           ;STOP & RETURN TO DEMCN MONITOR
                ;
0116 4C C0 01                JMP CHSET     ;DO IT AGAIN
```

```

.M  0100  20  8D  72  20  EC  72  8D  26
.:  0100  20  8A  72  A9  20  85  00  A5
.:  0108  00  C9  60  F0  08  20  C6  72
.:  0110  E6  00  4C  07  01  00  4C  00
.:  0118  01  ↓

```

Now run the program. Do this by setting the PC to 0100 and using the G command. The listing should look like this:

```

.R  0000  30  00  00  00  FF
.:  0100  ↓
.G
!'"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNØPQRSTUVWXYZ[\]^_`
*  0116  33  60  00  00  FF

```

The program may be continued, causing it to execute again, by typing G:

```

.G
!'"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNØPQRSTUVWXYZ[\]^_`
*  0116  33  60  00  00  FF
.G
!'"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNØPQRSTUVWXYZ[\]^_`
*  0116  33  60  00  00  FF
.G
!'"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNØPQRSTUVWXYZ[\]^_`
*  0116  33  60  00  00  FF

```

The CHSET program uses two DEMON monitor functions: CRLF is the DEMON function which causes a carriage-return and line-feed to be typed on the terminal. WRT is the routine which prints the character whose code is in the A register at the time of the call.

( ) 8. Save the CHSET program on paper tape (if your

terminal has a punch) as follows: First, punch some leader tape with the terminal in local mode. Then return to line mode and enter:

.WH 0100 0118 ↓

Turn the punch on after typing the second address, but before typing carriage-return. Then type carriage-return to punch the tape. When punching stops, turn the terminal back to local and type:

;00

and some blank trailer. This is a zero-length record which terminates your tape. See Appendix III for more information on tape formats.

( ) 9. Try re-loading your program using the LH command:

.LH

Now start the reader to load the program. The tape will be read and printed simultaneously. Stop the tape when the end is reached. (Before loading, you may wish to destroy the program in memory to verify that loading from tape actually works.)

( ) 10. Use the M and : commands to load the following program:

```

;CHECKOUT PROGRAM -- PRINT BINARY OF TYPED CHARACTER
;
;
0000      * = C          ;VARIABLE STORAGE IN PAGE ZERO
0000      BINARY * = * + 1 ;STORAGE FOR CHAR DURING DISSECTION
0001      CCOUNT * = * + 1 ;COUNT OF BITS REMAINING TO PRINT
;
0002      * = $0100      ;PROGRAM BEGINS ON PAGE ONE
;
CRLF      = $728A        ;DEMON CRLF ROUTINE
WRT       = $72C6        ;DEMON WRITE ROUTINE
RDT       = $72E9        ;DEMON READ ROUTINE
SPACE    = $7377        ;DEMON SPACE ROUTINE
;
0100      20 8A 72      PRIN   JSR CRLF          ;PRINT CARRIAGE RETURN & LINE FEED
0103      20 E9 72      JSR RDT           ;GET A CHARACTER
0106      85 CC          STA BINARY        ;SAVE FOR DISSECTION
0108      20 77 73      JSR SPACE         ;PRINT A SPACE
;
010B      A9 C8          LDA #8            ;INITIALIZE BIT COUNT
010C      85 01          STA COUNT
;
010F      A9 3C          PBLCCP LDA #0      ;ASSUME ZERO: LOAD ASCII "0"
0111      C6 C0          ASL BINARY        ;C=NEXT BIT
0113      BC C2          BCS PRINT        ;PRINT ZERO
;
0115      A9 31          LDA #'1         ;LOAD ASCII "1"
;
0117      20 C6 72      PRINT  JSR WRT      ;PRINT BINARY DIGIT
011A      C6 01          DEC CCOUNT       ;COUNT BIT PRINTED
011C      1C F1          BPL PBLCCP       ;GO NEXT BIT
;
011E      4C C0 01      JMP PRIN        ;DO IT ALL AGAIN
```

```
.M  0100 20 8D 72 A9 20 85 00 A5
.:  0100 20 8A 72 20 E9 72 85 00
.:  0108 20 77 73 A9 08 85 01 A9
.:  0110 30 06 00 B0 02 A9 31 20
.:  0118 C6 72 C6 01 10 F1 4C 00
.:  0120 01 ↓
```

The purpose of this program is to print the binary representation of an ASCII input character on the terminal. Run the program by starting it at location 0100. Try typing some characters:

```
.R  0116 33 60 00 00 FF
.:  0100 ↓
.G
U 101010101
B 101111011
l 110011101
```

There is obviously something wrong with the program. Bits which should be printed as 1's are 0's and vice versa. (Refer to your 6500 reference card for character codes.) Looking at the program, the problem is that the branch after PBLOOP goes the wrong way! It should be BCC, Branch if Carry Clear (or alternatively, the 1 and 0 loads could be interchanged). Thus, when a one-bit is shifted out of the character, a one should be printed.

Patch the program and try again ( the code for BCC is 90).



```
.M  0113  B0  02  A9  31  20  C9  72  C6
.:  0113  90  ↓
.R  7052  31  FC  FF  01  FF
.:  0100  ↓
.G
U  010101010
B  010000100
l  001100010
```

There is, alas, still an error--one too many bits is being printed. The cause of this is a little less obvious. (Do you see it?) To investigate the problem, set a breakpoint at location 011E. Do this by replacing the instruction there with a BRK (code of 00). Then run the program:

```
.M  011E  4C  00  01  EF  4C  00  01  00
.:  011E  00  ↓
.R  7052  31  FC  FF  01  FF
.:  0100  ↓
.G
U  010101010
*  011F  B0  00  00  AA  FF
```

Once the break has occurred, you are free to investigate the state of the program using DEMON. In particular, check the value in location COUNT:

```
.M  0000  00  FF  1B  2E  31  EA  FO  FA
```

Aha! Although COUNT starts out with a value of 8, it is going one step too far (FF is minus 1). This is because the test instruction, BPL PBLOOP is testing to see whether the count is

greater than or equal to zero. Replace it with BNE (code D0),  
replace your breakpoint with the original contents at that  
location, and try the program again.

.M 011C 10 F1 00 00 01 EF 4C

.: 011C D0 4C ↓

.R 011F B0 00 00 AA FF

.: 0100 ↓

.G

U 01010101

B 01000010

! 00110001

I 01001001

W 01010111

O 01001111

R 01010010

K 01001011

S 01010011

```

;CHECKOUT PROGRAM -- PRINT BINARY OF TYPED CHARACTER
;
;
CCCC      * = C          ;VARIABLE STORAGE IN PAGE ZERO
CCCC      BINARY * = +1 ;STORAGE FOR CHAR DURING DISSECTION
CC01      CCUNT * = +1  ;COUNT OF BITS REMAINING TO PRINT
;
0002      * = $0100     ;PROGRAM BEGINS ON PAGE ONE
;
CRLF      = $728A      ;DEMOM CRLF ROUTINE
WRT       = $72C6      ;DEMOM WRITE ROUTINE
RDT       = $72E9      ;DEMOM READ ROUTINE
SPACE     = $7377      ;DEMOM SPACE ROUTINE
;
C100      2C 8A 72     PBIN   JSR CRLF          ;PRINT CARRIAGE RETURN & LINE FEED
C103      20 E9 72     JSR RDT            ;GET A CHARACTER
C106      85 0C        STA BINARY          ;SAVE FOR DISSECTION
C108      2C 77 73     JSR SPACE          ;PRINT A SPACE
;
C10F      A9 C8        LCA #8              ;INITIALIZE BIT COUNT
C1CC      85 C1        STA CCUNT
;
C1CF      A9 3C        PBLCCP LCA #'0       ;ASSUME ZERO: LOAD ASCII "0"
0111      C6 C0        ASL BINARY          ;C=NEXT BIT
0113      90 C2        BCC PRINT          ;PRINT ZERO
;
0115      A9 31        LCA #'1           ;LOAD ASCII "1"
;
C117      2C C6 72     PRINT  JSR WRT          ;PRINT BINARY DIGIT
C11A      C6 01        DEC CCUNT          ;COUNT BIT PRINTED
C11C      DC F1        BNE PBLCCP        ;CC NEXT BIT
;
011E      4C 00 01     JMP PBIN          ;DO IT ALL AGAIN

```

CORRECTED PBIN PROGRAM

( ) 11. Save the corrected program using the WH command. Before punching the terminating record (as above in step 8), turn off the punch and set the PC to the start address of the program (0100). Then punch locations 00F6 and 00F7 on the tape, then the terminator (;00), and finally, some trailer:

```
.R 7052 30 37 FF 01 FF
.: 0100 ↓
.WH 00F6 00F7 ↓
;0200F6000101A2
.;00
```

The resulting tape can be loaded and then started as follows:

```
.LH
:
: (program loads in)
:
.G
```

Locations 00F6 and 00F7 contain the starting address for programs. You may assemble and load your starting address into these locations to make tapes which can be started with a minimum of operator action. The carriage-return delay time may also be set in this manner. See Appendix III.

( ) 12. It is also possible to punch BNPF-format tapes using DEMON. BNPF is the format used by some ROM programmers. The command is similar to that for writing hex tapes:

```
.WB 0100 0127 ↓
```

This command would punch the corrected PBIN program in BNPF

format. Try punching a BNPF tape. (Note that DEMON will not load tapes in this format--use hex format (WH) for saving programs for later loading into your JOLT.)

( ) 13. If you have a high-speed paper tape reader attached to your JOLT system, you can use it to load programs in hex format. The H command switches the load device to and from the high speed reader. If you have a high speed reader, try loading a tape as follows:

.H  
.LH

Note that control will not return to the user terminal until a terminator record (;00) is read.

THIS COMPLETES STEP-BY-STEP CHECKOUT  
OF THE DEMON MONITOR

HOW TO HAND-ASSEMBLE JOLT PROGRAMS

If you do not use an assembler to convert your JOLT programs to machine language (hexadecimal), you will have to convert your programs manually. Here is a suggested procedure.

The procedure consists of four steps:

STEP 1: Decide which variables and subroutines are to be placed in page zero and assign fixed locations to them.

STEP 2: Look up each instruction in the 6502 code chart and record the operation code in hexadecimal, noting how many bytes of memory are required by each instruction.

STEP 3: Determine the location in hexadecimal of each labelled instruction or variable.

STEP 4: Fill in all remaining values, using the locations determined in Step 3.

When writing a program for hand assembly, it is desirable to split your code into small routines which can be assembled separately. Since you will be loading and debugging your program by hand, you should leave some space for changes so that complete reassembly is not required to fix small problems.

By way of illustration, the PBIN program (used in the Monitor Checkout section) will be hand-assembled:

```

;CHECKOUT PROGRAM -- PRINT BINARY OF TYPED CHARACTER
;
;
;          *=0                ;VARIABLE STORAGE IN PAGE ZERO
BINARY   *=*+1              ;STORAGE FOR CHAR DURING DISSECTION
COUNT   *-*+1              ;COUNT OF BITS REMAINING TO PRINT
;
;          *=$0100            ;PROGRAM BEGINS ON PAGE ONE
;
CRLF     =$728A              ;DEMON CRLF ROUTINE
WRT      =$72C6              ;DEMON WRITE ROUTINE
RDT      =$72E9              ;DEMON READ ROUTINE
SPACE    =$7377              ;DEMON SPACE ROUTINE
;
PBIN     JSR CRLF             ;PRINT CARRIAGE RETURN & LINE FEED
          JSR RDT             ;GET A CHARACTER
          STA BINARY          ;SAVE FOR DISSECTION
          JSR SPACE           ;PRINT A SPACE
;
          LDA #8              ;INITIALIZE BIT COUNT
          STA COUNT
;
PBLOOP   LDA #'0              ;ASSUME ZERO: LOAD ASCII "0"
          ASL BINARY          ;C=NEXT BIT
          BCC PRINT           ;PRINT ZERO
;
          LDA #'1              ;LOAD ASCII "1"
;
PRINT    JSR WRT              ;PRINT BINARY DIGIT
          DEC COUNT           ;COUNT BIT PRINTED
          BNE PBLOOP          ;DO NEXT BIT
;
          JMP PBIN            ;DO IT ALL AGAIN

```

Step 1

Decide which variables and subroutines are to be placed in page zero and assign fixed locations to them.

Page zero contains locations 0000 to 00FF. The variables that are to reside in page zero must be identified prior to assembling the rest of the program since the mode and length of some instructions depend on whether their operands are in page zero. The sample program has two variables in page zero. They are simply assigned locations sequentially:

Loc    Contents    Instruction

			:	
			:	
0000	XX	BINARY **+1		;STORAGE FOR CHAR DURING DISSECTION
0001	XX	COUNT **+1		;COUNT OF BITS REMAINING TO PRINT
			:	
			:	

The program does not specify initial values of these locations, so the contents position is marked with X's, indicating that no values will have to be loaded there. In this example, there are no subroutines or other instructions to be assembled in page zero. It will be more convenient for hand assembly if such code, when it occurs, is placed after the variables. Then the position of variables will not depend on the length of preceding instructions.

Step 2

Look up each instruction in the 6502 code chart and record the operation code in hexadecimal, noting how many bytes of memory are required by each instruction.



The length and code of each instruction is determined by its mode. Some instructions, such as JSR and BNE, have only one possible mode, and thus present no difficulty. The mode for other instructions depends on the operand. For example, immediate mode is indicated by a pound sign (#) followed by a value. Instructions of this type use the operation code from the immediate columns of the code chart. The value following the pound sign is put in the second byte of the instruction. For example:

<u>Contents</u>	<u>Instruction</u>
A9 08	LDA #8
A9 31	LDA #'1 ;ASCII "1"

Instructions which have a zero page mode may be assembled in that mode if the operand is in fact in page zero:

85 01	STA COUNT
-------	-----------

The same operation with a non-zero page operand would occupy three bytes:

8D _ _	STA ADDR
--------	----------

Since symbols other than page zero (and certain pre-assigned addresses like WRT) do not have locations yet, we must leave blank spots to fill in later. Do mark the correct number of spaces for the unknown bytes, since the length of instructions determines the position of instructions which follow. Similarly, branch instructions will have their second bytes blank at this point:

D0 _	BNE PBLOOP
------	------------

Thus far, the partially assembled program looks like this:

<u>Location</u>	<u>Contents</u>	<u>Instruction</u>
		;CHECKOUT PROGRAM -- PRINT BINARY
		*=0
0000	XX	BINARY **+=1
0001	XX	COUNT **+=1
		*=\$0100
		CRLF = \$728A
		WRT = \$72C6
		RDT = \$72E9
		SPACE = \$7377
	20 8A 72	PBIN JSR CRLF
	20 E9 72	JSR RDT
	85 00	STA BINARY
	20 77 73	JSR SPACE
	A9 08	LDA #8
	85 01	STA COUNT
	A9 30	PBLOOP LDA #'0
	06 00	ASL BINARY
	90 ___	BCC PRINT
	A9 31	LDA #'1
	20 C6 72	PRINT JSR WRT
	C6 01	DEC COUNT
	D0 ___	BNE PBLOOP
	4C ___	JMP PBIN

### Step 3

Determine the location in hexadecimal of each labelled instruction.

It is now possible to fill in the location column, because the length of each instruction is known. Count in hex (0,1,2,...., 9,A,B,C,D,E,F) and write in the locations (of the first bytes)

of instructions and variables which have labels:

<u>Location</u>	<u>Contents</u>	<u>Instruction</u>
		;CHECKOUT PROGRAM -- PRINT BINARY
		*=0
0000	XX	BINARY *=*+1
0001	XX	COUNT *=*+1
		*=\$0100
		CRLF = \$728A
		WRT = \$72C6
		RDT = \$72E9
		SPACE = \$7377
0100	20 8A 72	PBIN JSR CRLF
	20 E9 72	JSR RDT
	85 00	STA BINARY
	20 77 73	JSR SPACE
	A9 08	LDA #8
	85 01	STA COUNT
010F	A9 30	PBLOOP LDA #'0
	06 00	ASL BINARY
	90 —	BCC PRINT
	A9 31	LDA #'1
0117	20 C6 72	PRINT JSR WRT
	C6 01	DEC COUNT
	D0 —	BNE PBLOOP
	4C — —	JMP PBIN

#### Step 4

Fill in the remaining values, using the locations determined in Step 3.

Locations of variables not already entered may now be filled in. Be sure to enter the low half first and the high half second.

For example, location PBIN is at address 0100. It is recorded as:

```
4C 00 01          JMP PBIN
```

Branches can now be completed by counting the number of bytes from the instruction to the target address. When going forward, count beginning with the first byte following the instruction, up to but not including the first byte at the target address. Thus, the boxed bytes are all that are counted in this example:

```
90  _          BCC PRINT
  A9 31        LDA #'1
20 C6 72 PRINT JSR WRT
```

When branching backwards, count those bytes from the end of the branch instruction itself (counting both bytes) to and including the first byte of the instruction at the target address. Thus:

```
                                STA COUNT
010F  A9 30  BPLOOP  LDA #'0
      06 00          ASL BINARY
      90 02          BCC PRINT
      A9 31          LDA #'1
0117  20 C6 72 PRINT JSR WRT
      C6 01          DEC COUNT
      D0  _          BNE PBLOOP
4C 00 01          JMP PBIN
```

Although you could count in hexadecimal, it is easier to count in decimal (base 10). When counting in decimal, count up whether going forward or backward, and look up the correct hexadecimal value on the Branch Chart shown on the next page and also in Appendix V. (If you do count in hexadecimal, backward counts need to be negated. Do this by subtracting the count from 100 hexadecimal. Forward hexadecimal counts may be used without modification.)

The assembly (page I-10) is now complete and ready to be loaded into JOLT.

CHART OF BRANCHES: DECIMAL TO HEXADECIMAL

FORWARD MSD →	0	1	2	3	4	5	6	7	
↓LSD↓	↑								
0	—	16	32	48	64	80	96	112	—
1	1	17	33	49	65	81	97	113	F
2	← 2	18	34	50	66	82	98	114	E
3	3	19	35	51	67	83	99	115	D
4	4	20	36	52	68	84	100	116	C
5	5	21	37	53	69	85	101	117	B
6	6	22	38	54	70	86	102	118	A
7	7	23	39	55	71	87	103	119	9
8	8	24	40	56	72	88	104	120	8
9	9	25	41	57	73	89	105	121	7
A	10	26	42	58	74	90	106	122	6
B	11	27	43	59	75	91	107	123	5
C	12	28	44	60	76	92	108	124	4
D	13	29	45	61	77	93	109	125	3
E	14	30	46	62	78	94	110	126	2
F	15	31	47	63	79	95	111	127	→ 1
—	16	32	48	64	80	96	112	—	0
									↑LSD↑
	↓ F	E	D	C	B	A	9	8	← MSD BACKWARD

<u>Location</u>	<u>Contents</u>	<u>Instruction</u>
		;CHECKOUT PROGRAM -- PRINT BINARY
		*=0
0000	XX	BINARY **+1
0001	XX	COUNT **+1
		*=\$0100
		CRLF =\$728A
		WRT =\$72C6
		RDT =\$72E9
		SPACE =\$7377
0100	20 8A 72	PBIN JSR CRLF
	20 E9 72	JSR RDT
	85 00	STA BINARY
	20 77 73	JSR SPACE
	A9 08	LDA #8
	85 01	STA COUNT
010F	A9 30	PBLOOP LDA #'0
	06 00	ASL BINARY
	90 <u>02</u>	BCC PRINT
	A9 31	LDA #'1
0117	20 C6 72	PRINT JSR WRT
	C6 01	DEC COUNT
	D0 <u>F1</u>	BNE PBLOOP
	4C <u>00</u> <u>01</u>	JMP PBIN

NOTE, HINTS, & RECOMMENDATIONS  
FOR USING YOUR JOLT MICROCOMPUTER

Storage Allocation

Some care in selecting locations for programs will save programming time and memory space. Page zero storage (0000 to 00FF) is a special resource in your system since it can be used for indirect references (to tables or routines) and since direct references to page zero locations require shorter instructions (2 instead of 3 bytes) for most instructions. Therefore, you will probably want to give priority to putting variables and data in page zero. Be sure to avoid locations at the high end of the page, however, since these are used by DEMON (00E3 to 00FF).

Code and data may also be placed in page one (0100 to 01FF). Be careful, however, to leave sufficient space for the stack (which, with DEMON's initialization, fills from the high end of the page downward, from location 01FF towards location 0100). You should allow at least three bytes for each level of nested subroutine call or interrupt possible in your program, plus space for all data you push onto the stack, plus an additional  $20_{10}$  bytes for DEMON. Failure to leave enough space may cause part of your program or data to be overwritten by the stack, with unpredictable results.



### 6502 Processor

1. Addresses for the 6502 processor are always stored low-order byte first, high-order byte second. Thus the lower part of an address is in the location having the lower-numbered address.

2. BRK acts as a two-byte instruction. When entered via BRK, DEMON adjusts the PC so as to make BRK in effect, operate as a one-byte instruction. Users who elect to handle BRK themselves (by changing the hardware IRQ interrupt vector) should be aware of this difference and program accordingly.

3. Certain undefined operation codes will cause the 6502 CPU to "hang up". When in this "hung up" state, the CPU can only be stopped with reset. NMI will not work. All other undefined codes may have unpredictable effects. Undefined codes should be avoided.

4. Attempting to read non-existent memory locations will usually yield the high-order part of the address as data. However, this is not true in all cases (indexed loads may respond differently), and should not be relied upon.

### The JOLT CPU Board

1. User PIA's are not fully address decoded, which means that each PIA uses 1K of address space. Thus, each PIA register appears every 4 locations in the 1K space used by that PIA. See the JOLT memory map in Appendix III.

2. Unless debouncing is provided for an NMI button, several interrupts can occur when this button is pressed. The result is that DEMON is interrupted in the process of servicing the original interrupt, and the users CPU registers are lost. With proper debouncing, however, CPU registers printed by DEMON after NMI will correctly reflect the state of the machine when the first interrupt occurred.

DETAILED DESCRIPTION OF DEMONDEMON Commands<sup>†</sup>

<u>Command</u>	<u>Description</u>
<u>↓</u>	Set line speed. After RESET, a carriage return is typed to allow DEMON to measure the line speed.
<u>.R</u>	Display user registers. The format is: PC P A X Y S where: PC is the program counter P is the processor status A is the A (accumulator) register X is the X (index) register Y is the Y (index) register S is the stack pointer low byte (high byte is always 01)
<u>.G</u>	Go. Begin execution at user PC location (see R command).
<u>.M <u>addr</u></u>	Memory examine. DEMON will display the eight bytes beginning at address <u>addr</u> .
<u>.: ADDR <u>data</u></u>	Alter registers or memory. DEMON allows the user to alter registers (if R command precedes) or memory (if M command precedes). Values for registers or memory locations which are not to be changed need not be typed

<sup>†</sup> Characters typed by the user are underlined. All other characters are typed by the computer. ↓ means carriage return.

—these fields may be skipped by typing spaces instead of data. The remainder of the fields in a line may be left unchanged by typing carriage return. The : command may be repeated to alter subsequent memory locations without the necessity of typing intervening M commands. Note that DEMON automatically types spaces to separate data fields.

Load Hexadecimal. DEMON responds with carriage return, line-feed and loads data in assembler output format from the terminal or high-speed paper tape reader. The format is:

Zero or more leading characters except  
";" (usually blank leader)

Any number of records of the form:  
;ccaaaadddd....ddssss

where:

cc is the number of bytes in the record in hex

aaaa is the hex address to store the first byte of data

dddd....dd is the data (two hex digits per byte)

ssss is the check-sum, which is the arithmetic sum, to 16 bits, of all the count, address and data bytes represented by the record

A terminating record of zero length,  
either: ;00 or ;↓

Note that read-after-write and check-sum tests are performed. An error will result in a "?" being typed at the point the error occurred. Data from records with bad check-sums is deposited in memory as received, prior to the error stop.

.H

High-speed/low-speed reader switch. This command switches the load device from the user's terminal to the high-speed reader or vice versa.

.WH addl addh↓

Write Hexadecimal. An assembler-format tape is generated at the user's terminal. Format is as described above in the LH command description. Note that the address range must be specified with the lower address first. As in the Alter command, DEMON types the space between the address fields.

.WB addl addh↓

Write BNPF. A BNPF format tape is generated at the user's terminal. Format is one or more records as follows:

```
aaaa BdddddddF BdddddddF BdddddddF BdddddddF
```

where:

aaaa is the address of the first of the four bytes specified in the record.  
(Note: BNPF conventions require that the letter "B" never occur in the address field. Blanks are substituted by DEMON.)

B is the letter "B", meaning begin data.  
 dddddddd is eight data bits—P for logical true, N for logical false.  
 F is the letter "F", meaning finish.

Note that the BNPF format is output as multiples of four bytes. Thus, a multiple of four bytes will always be punched even if a non-multiple of four bytes is specified.

!! Cancel Command. While typing any command, its further effect may normally be terminated by typing one or two carriage returns, as required. During alter (:), carriage return means that no further bytes (or registers) are to be altered.

### DEMON Interrupt and Breakpoint Action

#### BRK

The BRK instruction causes the CPU to interrupt execution, save PC and P registers on the stack, and branch through a vector at locations FFFE and FFFF. DEMON initializes this vector to point to itself on RESET. Unless the user modifies this vector, DEMON will gain control when a BRK instruction is executed, print an asterisk "\*" and the registers (as in R command), and wait for user commands. Note that after a BRK which vectors to DEMON, the user's PC points to the byte following the BRK; however, users who choose to handle BRK instructions themselves

should note that BRK acts as a two-byte instruction, leaving the PC (on return via RTI) two bytes past the BRK instruction.

### IRQ

Interrupt Request is also vectored through location FFFE. The CPU traps (as with BRK) through this vector when IRQ goes low, provided interrupts are not inhibited. Since this vector is the same as for BRK, DEMON examines the BRK bit in the P register after this type of interrupt. If a BRK did not cause the interrupt, then DEMON will pass control through the UINT vector. Users should normally put the address of their interrupt service routine in the UINT vector location. If an IRQ occurs and UINT has not been set by the user, DEMON reports the unexpected interrupt in the same way as an NMI (see below).

### NMI

Non-Maskable Interrupts vector through location FFFA. DEMON initializes this vector at RESET to point to itself. If an NMI occurs, a pound-sign character (#) precedes the asterisk and CPU registers printout. This action is the same for IRQ's if the user has not set this vector to point to his own routine.

### RESET or POWER-UP

On RESET or POWER-UP, DEMON takes control, initializes itself and the system, sets defaults for interrupt vectors and waits for a carriage-return input from the user to determine terminal line speed. After carriage-return is typed, control is passed to the user as in BRK.

DEMON Monitor Calls and Special Locations

<u>Call</u>	<u>Address</u>	<u>Action</u>	<u>Arg.</u>	<u>Result</u>	<u>Notes</u>
JSR WRT	72C6	Type a character	A	None	A,X cleared Y preserved
JSR RDT	72E9	Read a character	None	A	X cleared Y not preserved
JSR CRLF	728A	Type CR-LF and delay	None	None	A,X cleared Y preserved
JSR SPACE	7377	Type a space character	None	None	A,X,Y preserved
JSR WROB	72B1	Type a byte in hex	A	None	A,X cleared Y preserved
JSR RDHSR	733D	Read a character from high-speed paper tape reader	None	X—char read A—char trimmed to 7 bits	Y preserved

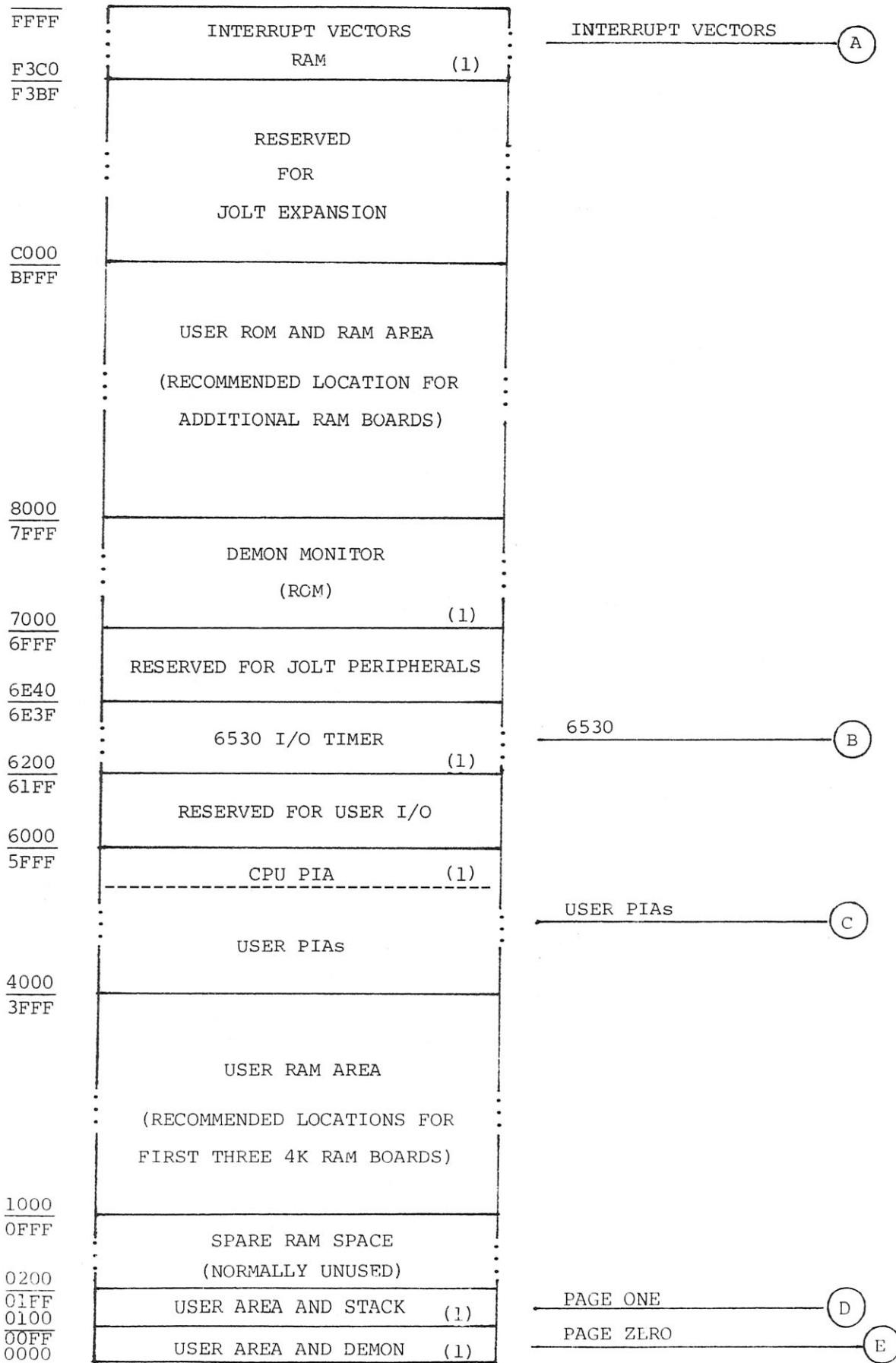
<u>Function</u>	<u>Locations</u>	<u>Notes</u>
Start Address	00F6,00F7	Set with hex tape on load
CR-LF Delay	00E3	Set on load or with user program (in <u>bit times</u> , minimum of 1. Zero means 256 bits-time delay).
UINT	FFF8	User IRQ vector
NMI Vector	FFFA	Hardware NMI vector
RESET Vector	FFFC	Hardware RESET vector
IRQ Vector	FFFE	Hardware IRQ vector



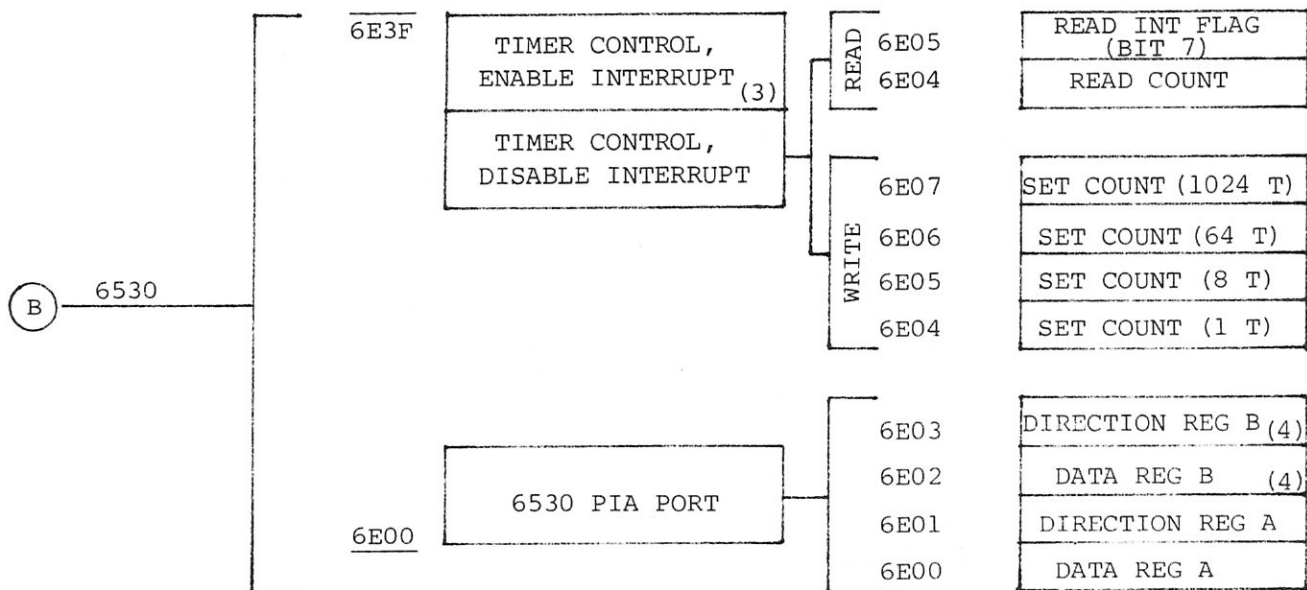
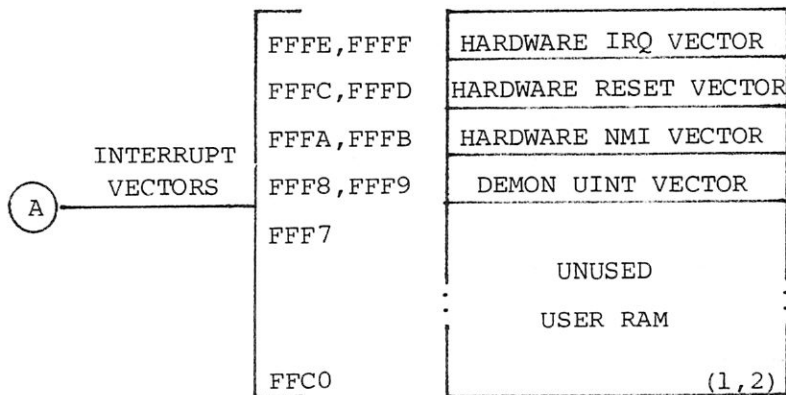
## JOLT SYSTEM MEMORY MAP

The memory map on the following pages explains what functions have been assigned to each segment of the JOLT address space. It is recommended that users respect this space allocation when adding memory and peripherals to their JOLT systems. Space has been reserved for 32K bytes of user RAM or ROM, seven additional PIA devices, and up to 512 user I/O device registers. Other areas are reserved for JOLT expansion, i.e., new JOLT peripherals and memory options will use these spaces. Users are advised to not use JOLT expansion space unless absolutely necessary.

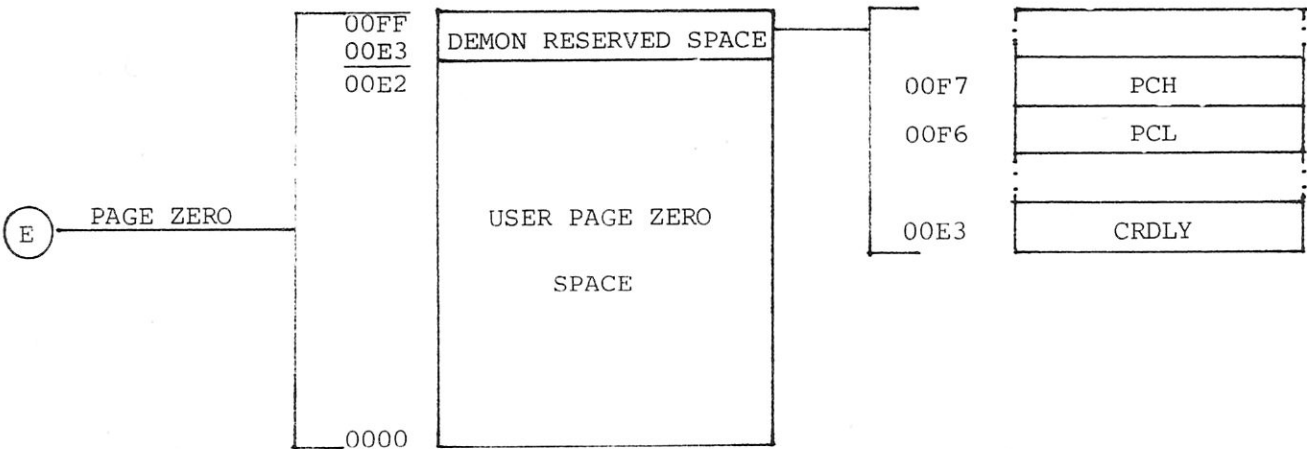
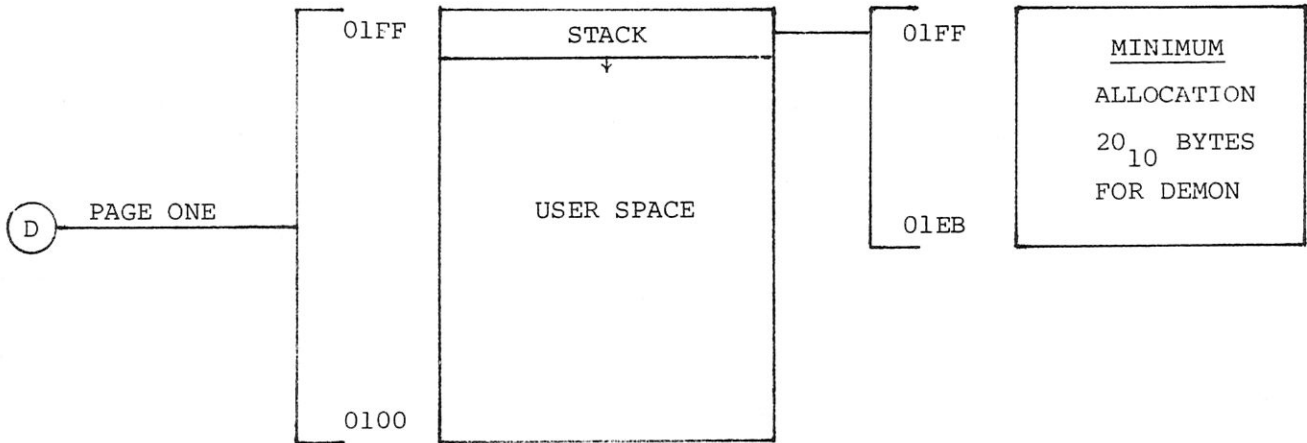
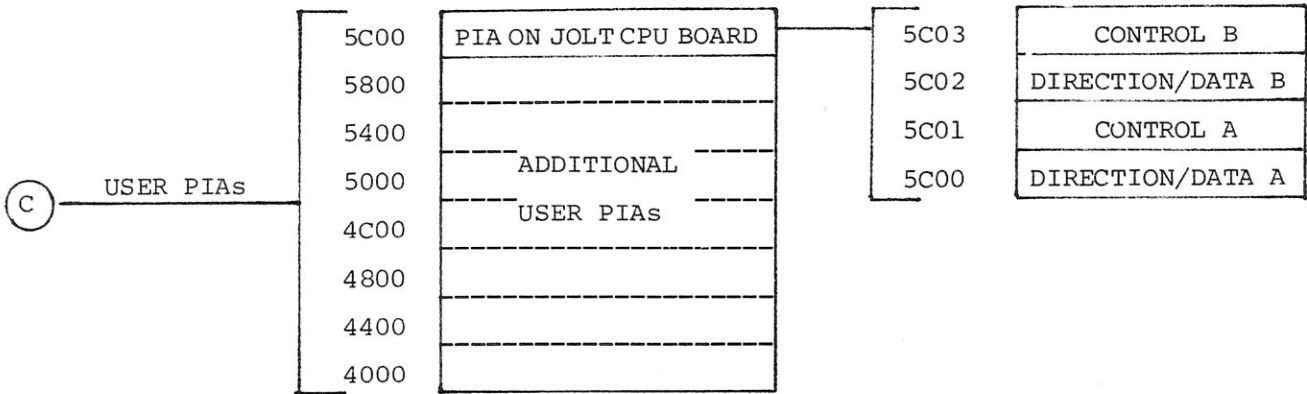
Note that some areas used by the JOLT CPU board and PIA boards have more space indicated than there are registers or locations in the device occupying them. This is because these devices do not decode all address bits, or use some of the address bits for special functions. For example, the 6530 timer determines the time scale and interrupt enable/disable by the address used to access it. Thus, these "partly filled" areas are actually entirely used and are not available for other uses.



(1) Standard on JOLT CPU board.



- (1) Standard on JOLT CPU board.
- (2) Available to user—not used by DEMON.
- (3) To get enable-interrupt address, add  $0008_{16}$  to disable-interrupt address with corresponding functions.
- (4) Reserved for DEMON use—TTY control and reset functions.



DEMON Memory Usage

DEMON uses the top  $29_{10}$  bytes of page zero (locations 00E3 through 00FF). The user is advised to avoid these locations, except as noted above, if return to DEMON or use of DEMON sub-routines is required before RESEtting the processor. DEMON also uses the hardware stack when it is in control. Provided the user does not alter the stack pointer during a break, and provided the stack does not overflow, DEMON will restore the stack to its original status before returning to the user's program. The user is advised to use page 1 (the stack page) cautiously, leaving at least  $20_{10}$  bytes for DEMON use during a break or when using other DEMON functions.

MCS 6502 INSTRUCTION SET SUMMARY

The following symbols are used in this summary:

A	Accumulator
X,Y	Index Registers
M	Memory
P	Processor Status Register
S	Stack Register
L,LOC	Absolute Location
Z	Zero-Page Location
*	Affected
-	Not Affected
+	Sum
$\wedge$	Logical AND
-	Difference
$\vee$	Logical Exclusive Or
$\uparrow$	Transfer From Stack
$\downarrow$	Transfer To Stack
$\rightarrow$	Transfer To
$\leftarrow$	Transfer To
V	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
#	Immediate Addressing Mode

MCS 6502 INSTRUCTION SET SUMMARY

Addressing Modes

Mode	Description	# Bytes	Example
IMPLIED	The operation performed is implied by the instruction.	1*	TAX AA Code for transfer A to X
ACCUMULATOR	The operation is performed upon the A register.	1	ROL A 2A Code for rotate left A
IMMEDIATE	The data accessed is in the second byte of the instruction.	2	LDA #3 A9 Code for load A immediate 03 Constant to use
ZERO PAGE	The address within page zero of the data accessed is in the second byte of the instruction.	2	LDA Z A5 Code for load A zero page 75 Low part of address on page zero
ZERO PAGE INDEXED BY X	The second byte of the instruction plus the contents of the X register (without carry) is the address on page zero of the data accessed.	2	LDA Z,X B5 Code for zero page indexed by X 75 Base address on page zero
ZERO PAGE INDEXED BY Y	The second byte of the instruction plus the contents of the Y register (without carry) is the address on page zero of the data accessed.	2	LDX Z,Y B6 Code for zero page indexed by Y 75 Base address on page zero
ABSOLUTE	The address of the data accessed is in the second and third bytes of the instruction.	3	LDA L AD Code for load A absolute 47 Low part of address 02 High part of address

\*Except BRK which is two bytes when not using DEMON.

Mode	Description	# Bytes	Example
INDEXED BY X	The address in the second and third bytes of the instruction, plus the contents of the X register is the address of the data accessed.	3	LDA L,X BD 47 02
INDEXED BY Y	The address in the second and third bytes of the instruction, plus the contents of the Y register is the address of the data accessed.	3	LDA L,Y B9 47 02
INDIRECT PRE-INDEXXED BY X	The second byte of the instruction plus the contents of the X register (without carry) is the address on page zero of the two-byte address of the data accessed.	2	LDA (Z,X) A1 75
INDIRECT POST-INDEXXED BY Y	The contents of the page zero two-byte address specified by the second byte in the instruction, plus the contents of the Y register is the address of the data accessed.	2	LDA (Z),Y B1 75
RELATIVE BRANCH	The second byte of the instruction contains the offset (in bytes) to branch address.	2	BEQ LOC F0 07
INDIRECT JUMP	The address in the second and third bytes of the instruction is the address of the address to which the jump is made.	3	JMP (LOC) 6C 47 02



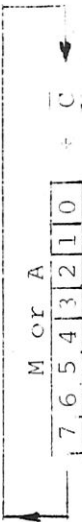
MCS 6502 INSTRUCTION SET SUMMARY

Instr	Description	Mode										Condition Codes											
		IMP	ACC	IMM	Z	Z, X	Z, Y	ABS	L, X	L, Y	(L, X)	(L, Y)	REL	IND	N	Z	C	I	D	V	B		
ADC	A + M + C → A, C Add memory to accumulator with carry			69	65	75	7D	70	79	61	71				*	*	*	-	-	*	-		
AND	A ∧ M → A "AND" memory with accumulator			29	25	35	2D	3D	39	21	31				*	*	-	-	-	-			
ASL	$C \leftarrow [7 6 5 4 3 2 1 0] \leftarrow 0$ Shift left one bit (memory or accumulator)		0A		06	16	0E	1E							*	*	*	-	-	-	-		
BCC	Branch on C = 0 Branch on carry clear												90		-	-	-	-	-	-			
BCS	Branch on C = 1 Branch on carry set												B0		-	-	-	-	-	-			
BEQ	Branch on Z = 1 Branch on result zero												F0		-	-	-	-	-	-			
BIT	A ∧ M, M7 → N, M6 → V Test bits in memory with accumulator				24		2C								M7	*	-	-	-	M6			
BMI	Branch on N = 1 Branch on result minus												30		-	-	-	-	-	-			
BNE	Branch on Z = 0 Branch on result not zero												D0		-	-	-	-	-	-			
BPL	Branch on N = 0 Branch on result plus												10		-	-	-	-	-	-			
BRK	Forced interrupt PC ← PC + 1 Force break	00													-	-	-	-	-	1	-	-	1





MCS 6502 INSTRUCTION SET SUMMARY

Instr	Description	Mode										Condition Codes									
		IMP	ACC	IMM	Z	Z,X	Z,Y	ABS	L,X	L,Y	(I,X)	(I,Y)	REL	IND	N	Z	C	I	D	V	B
LSR	$0 \rightarrow [7][6][5][4][3][2][1][0] \cdot [C]$ Shift right one bit (memory or accumulator)	4A	4E	5E											0	*	-	-	-	-	-
NOP	No Operation	EA													-	-	-	-	-	-	-
ORA	$A \vee M \cdot A$ "OR" memory with accumulator		09	05	15	0D	ID	19	01	11					*	-	-	-	-	-	
PHA	A Push accumulator on stack	48													-	-	-	-	-	-	
FHP	P Push processor status on stack	08													-	-	-	-	-	-	
PLA	A Pull accumulator from stack	68													-	-	-	-	-	-	
PLP	P Pull processor status from stack	28													-	-	-	-	-	-	
ROL	 $[7][6][5][4][3][2][1][0] \cdot [C]$ Rotate one bit left (memory or accumulator)		2A	26	36	2E	3E							*	*	*	-	-	-	-	
RTI	P, PC + 1 Return from interrupt	40																			
RTS	PC, PC + 1 Return from subroutine	60													-	-	-	-	-	-	

MCS 6502 INSTRUCTION SET SUMMARY

Instr	Description	Mode										Condition Codes									
		IMP	ACC	IMM	Z	Z,X	Z,Y	ABS	L,X	L,Y	(L,X)	(L,Y)	REL	IND	N	Z	C	I	D	V	B
SBC	A - M - $\overline{C}$ → A Note: $\overline{C}$ = Borrow Subtract memory from accumulator with borrow			E9	E5	F5	ED	FD	F9	E1	F1				*	*	*	-	-	*	-
SEC	1 → C Set carry flag	38													-	-	1	-	-	-	-
SED	1 → D Set decimal mode flag	F8													-	-	-	-	1	-	-
SEI	1 → I Set interrupt disable flag	78													-	-	-	1	-	-	-
STA	A → M Store accumulator in memory				85	95	8D	9D	99	81	91				-	-	-	-	-	-	-
STX	X → M Store index X in memory				86										-	-	-	-	-	-	-
STY	Y → M Store index Y in memory				84	94									-	-	-	-	-	-	-
TAX	A → X Transfer accumulator to index X	AA													*	*	-	-	-	-	-
TAY	A → Y Transfer accumulator to index Y	A8													*	*	-	-	-	-	-
TSX	S → X Transfer stack pointer to index X	BA													*	*	-	-	-	-	-

MCS 6502 INSTRUCTION SET SUMMARY

Instr	Description	Mode											Condition Codes									
		IMP	ACC	IMM	Z	Z,X	Z,Y	ABS	I,X	I,Y	(L,X)	(L,Y)	REL	IND	N	Z	C	I	D	V	B	
TXA	X → A Transfer index X to accumulator	8A													*	-	-	-	-	-	-	
TXS	X → S Transfer index X to stack pointer	9A													-	-	-	-	-	-	-	
TYA	Y → A Transfer index Y to accumulator	98													*	-	-	-	-	-	-	

CHART OF BRANCHES: DECIMAL TO HEXADECIMAL

FORWARD MSD →	0	1	2	3	4	5	6	7	
↓LSD↓									
0	—	16	32	48	64	80	96	112	—
1	1	17	33	49	65	81	97	113	F
2	2	18	34	50	66	82	98	114	E
3	3	19	35	51	67	83	99	115	D
4	4	20	36	52	68	84	100	116	C
5	5	21	37	53	69	85	101	117	B
6	6	22	38	54	70	86	102	118	A
7	7	23	39	55	71	87	103	119	9
8	8	24	40	56	72	88	104	120	8
9	9	25	41	57	73	89	105	121	7
A	10	26	42	58	74	90	106	122	6
B	11	27	43	59	75	91	107	123	5
C	12	28	44	60	76	92	108	124	4
D	13	29	45	61	77	93	109	125	3
E	14	30	46	62	78	94	110	126	2
F	15	31	47	63	79	95	111	127	1
—	16	32	48	64	80	96	112	—	0
									↑LSD↑
	F	E	D	C	B	A	9	8	← MSD BACKWARD

Forward Branches

Count in decimal from the end of the branch instruction to target address. Do not count bytes in either the branch or target instruction. Find the count in the center of the chart. Use the Most-Significant-Digit at the top of the column, and the Least-Significant-Digit at the left of the row.

Reverse Branches

Count in decimal from the end of the branch to the beginning of the target instruction. Count all bytes in both instructions. Find the count in the center of the chart. Use the Most-Significant-Digit at the bottom of the column, and the Least-Significant-Digit at the right of the row.

Example

Forward 10 gives 0A. Backward 10 gives F6.

Chart Idea Credit

Ray Boaz, Homebrew Computer Club Newsletter, Volume 1, Number 7, September 1975.