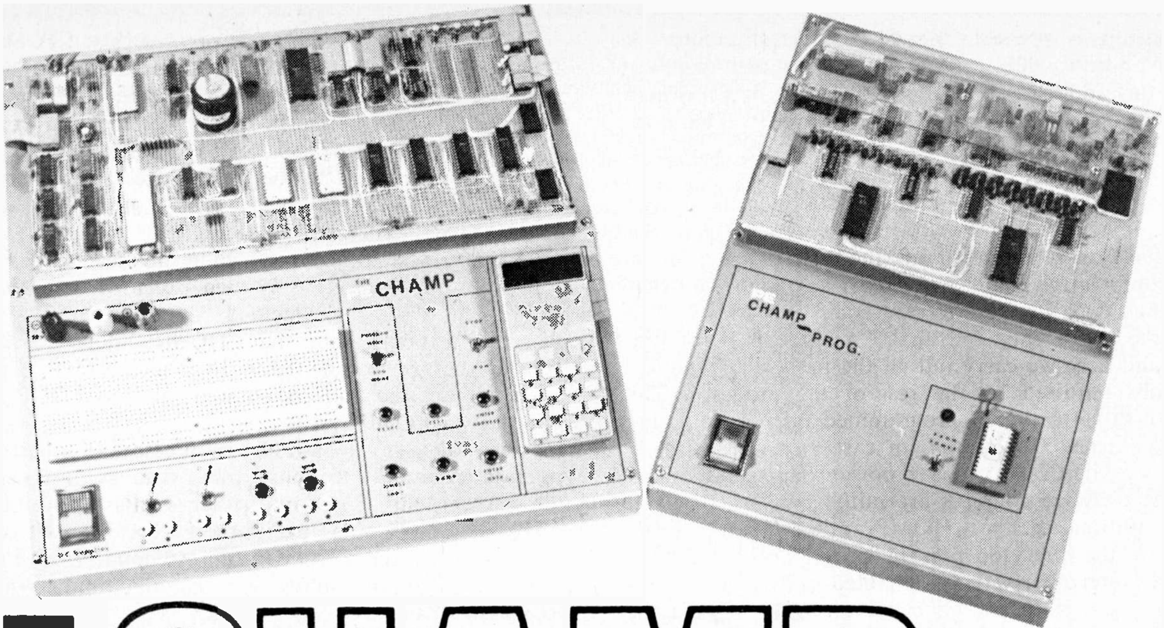


PE CHAMP

R.W. COLES
B. CULLEN

PART SIX

Now that the "hardware" description of CHAMP is complete, we can move on to consider that magic ingredient, "software". As you will no doubt recall, the program which makes CHAMP work is called CHOMP (CHamp Operating system and Monitor Program), and this month we shall examine the program.

COMMERCIAL KITS

Most commercial microprocessor development kits provide the user with only a simple listing of their operating programs, and ploughing through these listings to gain an understanding of how the system operates can be a painful experience.

CHAMP is for hardware oriented people; not the software genius; so we have done more than just provide a simple listing of the code you will need in PROM chip zero to get CHAMP to work. We cannot, for space reasons, give an intimate description of *every* line in the program, but we will be discussing the overall program flow chart. As an introduction to programming techniques, we will be showing how segments of the overall flow chart are converted first into more detailed flow charts, and then into hexadecimal code. In this way we hope to use CHOMP not only as an essential part of CHAMP, but also as a sort of software training ground for fledgling CHAMP programmers!

Constructors are advised to spend some time developing a familiarity with this program, and also of course with the 4040 instruction set which it uses.

4702A PROM

CHOMP should, strictly speaking, be called a firmware program, because is resides not on paper tape or magnetic cassette, but in a ROM or Read Only Memory. The type of ROM used is in fact an eraseable and reprogrammable type using the FAMOS technology, and these devices are more properly described as EPROMS, or just PROM for short. The actual device used is the 4702A chip which contains 256 eight bit words, has supply requirements compatible with the 4040, and can be erased by means of exposure to short wave ultra violet light. The 4702A is a selection from the 1702A family, characterised to work on +5V and −10V supplies over the full temperature range, instead of the usual +5V and −9V of the 1702A. The 4702A is also a less speedy device than the 1702A, having a 1·7μs maximum access time. The only extra requirement the 4702A has, is for that extra volt on the supply rails, and in fact it is virtually certain that any 1702A chip will work well in the CHAMP circuit, at least over the usual domestic temperature range. This has been tried on the prototype with complete success, and opens up the possibility of using the low cost 1702As now being advertised. Of course, it is not possible for us to *guarantee* success with anything other than the 4040 manufacturers' recommended 4702A devices.

CHOMP uses 248 locations out of the 256 available in a 4702A, and the PROM containing CHOMP has to be plugged into the CHIP ZERO location, i.e. the left hand PROM socket.

MAIN FLOW CHART

Figure 6.1 shows the main flow chart of CHOMP, and this is in effect an overview of the whole program in a much simplified form. We have chosen to use just four symbols to draw the flow chart:

(a) Circles represent the beginning and end of events.
(b) Oblong boxes represent actions to be performed.
(c) Diamonds represent decision points with two possible exits.
(d) Square arrows represent "Jumps" to other pages of memory.

When power is first applied to CHAMP, or when the

RESET button is pressed, the 4040 address counter is cleared to address 000H, and it fetches its next instruction from this address, which is of course the first location in chip zero, and the beginning of CHOMP. The flow chart can be traced from this RESET point which is located at the top left of Fig. 6.1.

The first box is not very exciting; it simply tells us that we must jump past address 003H, because this is the program location which contains the first instruction of the Interrupt routine, and we only want to go to *that* address when a hardware interrupt is acknowledged.

Box three represents the first "meaty" part of the program, and here we carry out all the preliminary housekeeping jobs required by the rest of the program. The 4265 INPUT/OUTPUT chip is programmed into mode 9; the switch flag latches are cleared (in case any were already indicating a switch closure when power was applied), and the various software counters are initialised to a required starting condition (i.e. the CHOMP address counter is set to point to the first location in program RAM, 200H). Finally, the interrupt system is enabled so that any interrupt signal from now on will cause the 4040 to save the current address on its internal stack, and jump to 003H, the interrupt vector. The only source of interrupt recognised by CHOMP itself is the keyboard, but for the moment let's assume that no interrupt has been received and continue on to box 4.

After initialisation, the CHOMP address counter holds 200H, and this box is present to load that address value into the display buffer register, so that we can see it on the right hand three display digits. Notice that box 4 is also entered via LOOP 2, and in this case the current address value (whatever it is) will be displayed.

Box 5 performs the vital job of refreshing the l.e.d. display. Each time this box is entered, a new eight bit word is presented to the segment lines and the display shift register is stepped on one position. Eight entries are required to refresh the complete display, and to ensure regular use, box 5 is made part of LOOP 1, through which the 4040 cycles continuously as long as no control switches are pressed.

Box 6 is also part of LOOP 1, and the main purpose of this box is to read into the 4040 accumulator register the state of the four control switches, so that the state of these may be checked and appropriate action taken. The interrupt system is again disabled at this point to prevent interference with switch responses. The INTERRUPTS RECOGNISED zone is quite extensive enough for a prompt response to any key press, and making the rest of the program interruptible would be an unnecessary complication.

Box 7 is a decision based not upon the switch flags, but upon the separate 4040 TEST input. If the TEST button is pressed, box 7 ensures a jump to the start of Chip 1, address 100H. Chip 7 is normally used for the PROMPT programmer software of course, but if the programmer is not in use, any 4702A resident program can be started by pressing TEST.

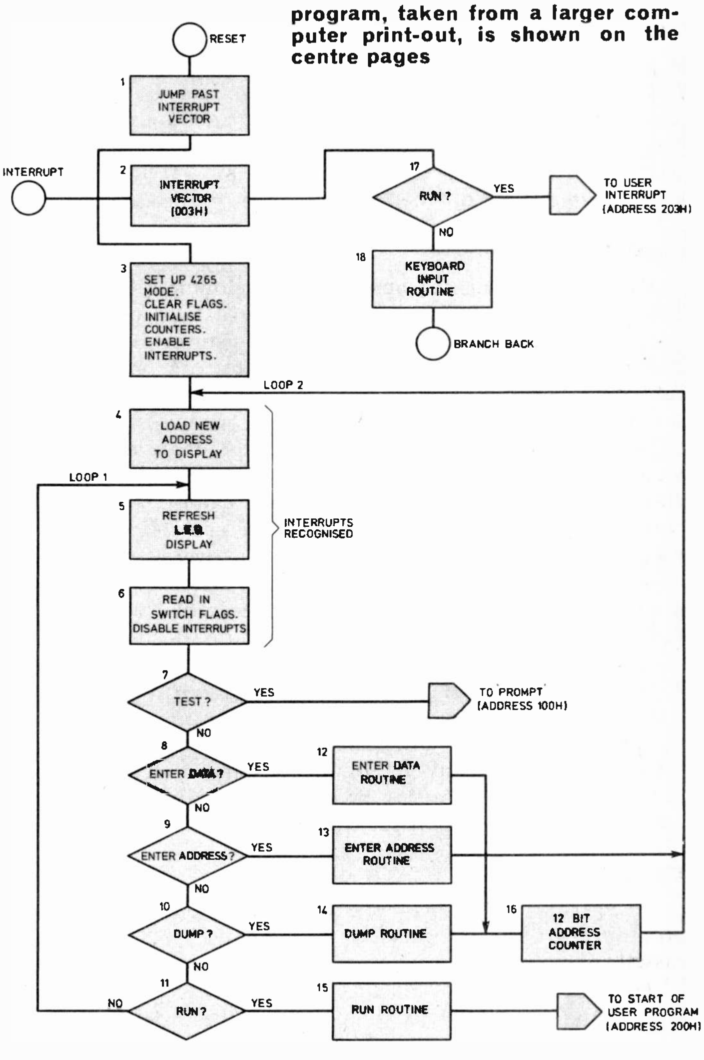
Boxes 8, 9, 10 and 11, check each of the switch flag bits in the accumulator in turn, by shifting them into the carry flip-flop and performing a JCN instruction. If no switches are pressed at box 6 time, then LOOP 1 is completed, and is in fact repeated indefinitely, refreshing the display and checking the switches on each pass. Needless to say, CHAMP spends most of its time in this loop when CHOMP is running, only leaving it intermittently, to respond to control switch closures.

If the ENTER DATA switch is pressed then CHAMP exits from LOOP 1 at box 8. Box 12 represents a routine which takes data previously entered via the keyboard and stores that data (8 bits) in the program RAM location pointed to by the CHOMP address counter, before passing on to box 16 to increment to the next address in sequence. The new address is displayed by means of box 4, and then LOOP 1 is re-entered.

If the ENTER ADDRESS switch is pressed, then LOOP 1 is left at box 9. Box 13 is then executed, and this loads the three digit hexadecimal data previously entered via the keyboard into the CHOMP address counter to replace the previous contents. In this case there is no need to increment the address counter, and so LOOP 1 is re-entered via LOOP 2.

When the DUMP switch is pressed, a sequence of operations similar to those for ENTER DATA takes place, although in this case box 14 represents a routine which reads data (8 bits) from the program RAM location pointed to by the CHOMP address counter, and loads it into the display buffer for examination. When the PROGRAM MODE/RUN MODE switch is in the RUN position box 15 is entered, and a routine is executed to cause an unconditional jump to the start of the user program RAM at address 200H. From this point onwards of course, CHOMP has relinquished its control of CHAMP facilities to whatever user program is resident in RAM.

Fig. 6.1. CHOMP main flow chart. The complete CHOMP program, taken from a larger computer print-out, is shown on the centre pages



```
0042   AC     LD 0CH
0043   B2     XCH 2
0044   AE     LD 0EH
0045   B4     XCH 4;                  RELOAD COUNTER WITH 12 BIT
                                        ADDRESS
0046   5097   JMS CLRF;               CLEAR SWITCH FLAGS
0048   400E   JUN LOOP 2
004A   2C00   DUMP: FIM 0CH, 00H
004C   2800   FIM 8, 00H
004E   29     SRC 8
004F   A4     LD 4
0050   E1     WMP;                    SELECT RAM CHIP
0051   23     SRC 2;                  ADDRESS BYTE
0052   0E     RPM;                    GET LS NIBBLE
0053   BC     XCH 0CH
0054   0E     RPM;                    GET MS NIBBLE
0055   BE     XCH 0EH
0056   2A00   FIM 0AH, 00H;           CLEAR KBD COUNT
0058   2850   FIM 8, 50H
005A   29     SRC 8
005B   E2     WRR;                    CLEAR FLAGS BUT NOT  KBD
005C   50CE   JMS LOKY                DISPLAY DUMP BYTE
005E   730E   COUNT: ISZ 3, LOOP 2
0060   720E   ISZ 2, LOOP 2
0062   740E   ISZ 4, LOOP 2
0064   400E   JUN LOOP 2;             ADDRESS COUNTER
0066   0B     INTER: SB1
0067   B6     XCH 6
0068   F7     TCC;                    SAVE AC AND CARRY
0069   B7     XCH 7
006A   2240   FIM 2, 40H
006C   23     SRC 2
006D   EA     RDR;                    GET PROG/RUN SWITCH
006E   F5     RAL;                    PUT IN CY
006F   1276   JC PROMO;               ARE WE IN PROG MODE?
0071   A7     LD 7;                   USER IR SO RESTORE STATUS
0072   F6     RAR
0073   A6     LD 6
0074   4203   JUN 203H;               GO TO USER IR
0076   2480   PROMO: FIM 4, 80H;      SELECT 4265
0078   25     SRC 4
0079   EC     RDO;                    GET KBD BCD
007A   BF     XCH 0FH;                PUT IN KBD TEMP
007B   6B     INC 0BH;                BUMP TABLE INDEX
007C   6B     INC 0BH
007D   D8     LDM 8
007E   BA     XCH 0AH;                MS TABLE INDEX NIBBLE
007F   AF     LD 0FH;                 PUT KBD IN ACC
0080   3B     JIN 0AH;                BRANCH VIA TABLE
0082          0RG 082H
0082   4088   TABLE: JUN FIRST
0084   408B   JUN SECON
0086   408E   JUN THIRD
0088   BE     FIRST: XCH 0EH;         FIRST KBD DIGIT INRE
0089   4091   JUN TERM
008B   BC     SECON: XCH 0CH;         SECOND TO RC
008C   4091   JUN TERM

00C9   78C9   LOOP 3: ISZ 8, LOOP 3
00CB   79C9   ISZ 9, LOOP 3
00CD   C0     BBL 0H
00CE   2800   LOKY: FIM 8, 00H;       SET UP ADDRESS
00D0   AE     LD 0EH
00D1   B1     XCH 1;                  GET LOW FOUR
00D2   50DD   JMS HEXL;               CONVERT TO SEVEN SEG
                                        CODE
00D4   AC     LD 0CH
00D5   B1     XCH 1;
00D6   50DD   JMS HEXL;               CONVERT TO SEVEN SEG
                                        CODE
00D8   AD     LD 0DH
00D9   B1     XCH 1
00DA   50DD   JMS HEXL;               CONVERT TO SEVEN SEG
                                        CODE
00DC   C0     BBL 0H
00DD   DF     HEXL: LDM 0FH;          SEVEN SEG TABLE LOOKUP
00DE   B0     XCH 0;                  TABLE BASE IN RP 0
00DF   30     FIN 0;                  GET SEG CODE FROM TABLE
00E0   29     SRC 8
00E1   A1     LD 1;                   FIRST FOUR TO 4002
00E2   E0     WRM
00E3   69     INC 9;                  BUMP NIBBLE POINTER
00E4   29     SRC 8
00E5   A0     LD 0;                   LAST FOUR TO 4002
00E6   E0     WRM
00E7   69     INC 9;                  BUMP NIBBLE POINTER
00E8   C0     BBL 0
00F0          ORG 0F0H
00F0   7E     DB 07EH
00F1   0C     DB 00CH;                LOOKUP TABLE
00F2   B6     DB 0B6H
00F3   9E     DB 09EH
00F4   CC     DB 0CCH
00F5   DA     DB 0DAH
00F6   FA     DB 0FAH
00F7   0E     DB 00EH
00F8   FE     DB 0FEH
00F9   DE     DB 0DEH
00FA   EE     DB 0EEH
00FB   F8     DB 0F8H
00FC   72     DB 072H
00FD   BC     DB 0BCH
00FE   F2     DB 0F2H
00FF   E2     DB 0E2H
```

SYMBOL TABLE

CLRF	0097	COUNT	005E	DATO	00C5	DDRV	00B1
DUMP	004A	ENTAD	0040	ENTDA	0032	FIRST	0088
HEXL	00DD	INTER	0066	LADR	00A2	LOKY	00CE
LOOP 1	0010	LOOP 2	000E	LOOP 3	00C9	PASS	00C7
PROMO	0076	RUN	0028*	SECON	008B	SKIP	001D
TABLE	0082*	TERM	0091	THIRD	008E		

Fig. 6.2.
Complete CHOMP program

```
000     00      NOP
0001    4005    JUN s + 4;            SKIP INTERRUPT
0003    4066    JUN INTER;            INTERRUPT VECTOR
0005    2880    FIM 8, 80H
0007    29      SRC 8
0008    D9      LDM 9
0009    E1      WMP;                  SET UP 4265 MODE
000A    5097    JMS CLRF;             CLEAR SWITCH FLAGS
000C    2428    FIM 4, 28H;           SET MS ADDR. COUNT AND
                                        DDRV COUNT
000E    50A2    LOOP 2: JMS LADR;     LOAD ADDRESS TO DISPLAY
0010    0C      LOOP 1: EIN;          ENABLE INTERRUPTS
0011    50B1    JMS DDRV;             DISPLAY DRIVER
0013    2840    FIM 8, 40H
0015    29      SRC 8
0016    EA      RDR;                  READ IN SWITCHES
0017    0D      DIN;                  DISABLE INTERRUPTS
0018    F6      RAR;                  FIRST FLAG TO CY
0019    111D    JNT SKIP;             JUMP TO CHIP 1 IF TEST SET
001B    4100    JUN 100H
001D    1A32    SKIP: JNC ENTDA;      ENTER DATA?
001F    F6      RAR;                  NEXT FLAG TO CY
0020    1A40    JNC ENTAD;            ENTER ADDRESS
0022    F6      RAR;                  NEXT FLAG TO CY
0023    1A4A    JNC DUMP;             DUMP?
0025    F6      RAR;                  LAST FLAG TO CY
0026    1210    JC LOOP 1;            RUN OR BACK AGAIN
0028    5097    RUN: JMS CLRF
002A    2880    FIM 8, 80H
002C    29      SRC 8
002D    F0      CLB
002E    E5      WR1;                  BLANK DISPLAY
002F    E6      WR2
0030    4200    JUN 200H;             JUMP TO USER PROG IN
                                        CHIP 2
0032    2800    ENTDA: FIM 8, 00H
0034    29      SRC 8
0035    A4      LD 4
0036    E1      WMP;                  SELECT PROGRAM RAM CHIP
0037    23      SRC 2;                ADDRESS BYTE
0038    AC      LD 0CH
0039    E3      WPM;                  WRITE LEAST SIG NIBBLE
                                        TO RAM
003A    AE      LD 0EH
003B    E3      WPM;                  WRITE MOST SIG NIBBLE
003C    5097    JMS CLRF;             CLEAR SWITCH FLAGS
003E    405E    JUN COUNT;            BUMP ADDRESS COUNT
0040    AD      ENTAD: LD 0DH;        PUT KBD IN COUNTER
0041    B3      XCH 3
```

```
008E    BD      THIRD: XCH 0DH;       THIRD IN RD
008F    2A00    FIM 0AH, 00H;         CLEAR KBD CHAR COUNT
0091    50CE    TERM: JMS LOKY;       PUT NEW KBD DIGIT IN 4002
0093    A7      LD 7
0094    F6      RAR;                  RESTORE STATUS
0095    A6      LD 6
0096    02      BBS
0097    2850    CLRF: FIM 8, 50H;     SELECT ROM O.P. PORT 5
0099    29      SRC 8
009A    E2      WRR;                  CLEAR SWITCH FLAGS
009B    2A00    FIM 0AH, 00H;         CLEAR KBD REGISTERS
009D    2C00    FIM 0CH, 00H
009F    2E00    FIM 0EH, 00H
00A1    C0      BBL 0H
00A2    280A    LADR: FIM 8, 0AH;     FETCH 4002 SRC START ADD
00A4    A4      LD 4
00A5    B1      XCH 1
00A6    50DD    JMS HEXL;             CONVERT TO SEVEN SEG
                                        CODE
00A8    A2      LD 2
00A9    B1      XCH 1
00AA    50DD    JMS HEXL;             CONVERT TO SEVEN SEG
                                        CODE
00AC    A3      LD 3
00AD    B1      XCH 1
00AE    50DD    JMS HEXL;             CONVERT TO SEVEN SEG
                                        CODE
00B0    C0      BBL 0H
00B1    27      DDRV: SRC 6;          DISPLAY DRIVER ROUTINE
00B2    E9      RDM;                  LOW FOUR FROM 4002
00B3    2880    FIM 8, 80H
00B5    29      SRC 8;                LOW FOUR TO 4265 PORT X
00B6    E5      WR1
00B7    67      INC 7;                BUMP NIBBLE POINTER
00B8    27      SRC 6
00B9    E9      RDM;                  HIGH FOUR FROM 4002
00BA    29      SRC 8
00BB    E6      WR2;                  LOW FOUR TO 4265 PORT Y
00BC    67      INC 7;                BUMP NIBBLE POINTER
00BD    75C5    ISZ 5, DATO;          INCREMENT SHIFT COUNTER
00BF    DF      LDM 0FH;              FETCH WRM CODE
00C0    E0      WRM;                  BIT SET 4265 Z3 HIGH
00C1    D8      LDM 08H;              PRESET SHIFT COUNTER
00C2    B5      XCH 5
00C3    40C7    JUN PASS
00C5    DE      DATO: LDM 0EH;        FETCH WRM CODE
00C6    E0      WRM;                  BIT SET Z3 LOW
00C7    2880    PASS: FIM 8, 080H;    SLOW DOWN MULTIPLEX
                                        RATE
```

INTERRUPT

If CHOMP is running and a keyboard switch is pressed, one interrupt is latched by IC10 and CHOMP responds (from the INTERRUPTS RECOGNISED zone) with a jump to box 2 (address 003H) which is called the interrupt vector.

Box 2 contains another jump to the start of the interrupt routine proper, which just happens to be elsewhere in chip zero (actually at address 066H). Before the keyboard handler routine is entered, CHOMP makes a check to see whether it is actually in PROGRAM MODE. Interrupts to RUN MODE user programs are also vectored to address 003H, so this check is essential, and is represented by box 17. If PROGRAM MODE *is* current, then box 18 is entered and a routine executed to read-in a single four bit hexadecimal digit from the keyboard, and store it away in a 4040 register. The keyboard routine also updates the display buffer so that each digit appears on the left hand side of the display as it is entered.

User interrupts are re-vectored to address 203H, so that the RAM resident program can define how a response is to be made. If you want to use the keyboard interrupt routine in your own program, simply carry out a JUN (Jump UNconditional) to address 066H from address 203H. Remember to use a BBS (Branch back and SRC) at the end of any "custom" interrupt routines you write!

CHOMP LISTING

Figure 6.2 is a complete listing of the CHOMP program, showing hexadecimal address data (column 1), hexadecimal instruction code data (column 2), mnemonic instruction codes (column 3) and comment lines (column 4).

The listing of Fig. 6.2 is the output of an assembler program which runs not on CHAMP, but on a much larger computer. Before anyone cries *cheat!* let me hasten to point out that CHOMP *was originally* written without the benefit of any such sophisticated facilities, directly in hexadecimal code. The reasons for eventually putting CHOMP into this form are simple:

(a) The assembler program does produce nice neat output listings which are useful for publication purposes.

(b) Since we are indeed saying that you do not *need* assembler programs when writing CHAMP software, we thought it only fair to show you what you are doing without!

When entering programs into an assembler, you have to enter columns 3 and 4 of Fig. 6.2 via a teletype terminal. From these the assembler produces columns 1 and 2 which tell you *what* hexadecimal code to enter *where* in program memory. The advantages of using an assembler program are firstly that the mnemonic instruction codes are all you have to remember, and that is fairly easy: and secondly, that instead of having to specify addresses in hexadecimal code you can use labels (i.e. names) instead. The assembler program will turn instruction mnemonics and address labels directly into hexadecimal code, and produce neat listings like the one shown here.

These sort of facilities sound very useful of course, and we would be the first to agree that with more complicated micros such as the Z80 or the 6800 they are very helpful indeed. The disadvantages are of course that you have to have lots of RAM available to store all those useful comments, and you also need a teletype or a V.D.U. The authors have assembler facilities available to them, but even so we prefer to write our 4040 programs directly in hex, with a pencil and paper; an exercise which is quite simple after a little practice!

Before leaving the subject of assemblies, let me explain a few things about the output listing shown in Fig. 6.2 which may be puzzling some readers:

(a) ORG and END are pseudo instructions, nothing to do with the 4040 but understood by the assembler.

(b) Some lines in column 2 have four hexadecimal digits. These involve two line instructions such as JUN, and will of course occupy two consecutive bytes in program memory.

(c) Some lines are field separators required by the assembler program.

(d) Notation. The assembler requires hexadecimal data to start with a decimal digit (don't ask us why!), and to be followed by an H. This means that FF hex is written 0FFH, while 2F hex is written 2FH.

(e) Register references can be made in a variety of ways, but we referred to them using hexadecimal, or decimal where this was equivalent.

Putting this information together, refer to Fig. 6.3 which explains how a complete assembler line is made up.

To get a CHOMP PROM from Fig. 6.2, all you have to do is step through the PROM addresses (column 1) entering the hexadecimal instruction codes from column 2. To do this you need a PROM programmer of course, and since most constructors will not have access to such a unit, arrangements have been made for the provision of a CHAMP programming service which will carry out the programming for you. Details next month.

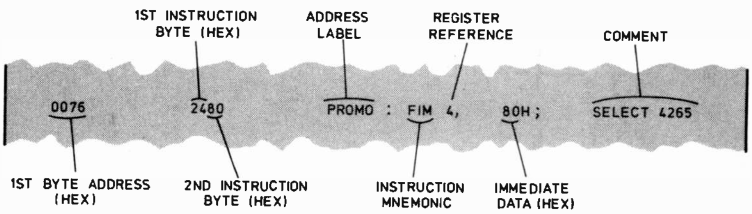


Fig. 6.3. One assembler output line and what it means

DETAILED FLOW CHARTS

No doubt many readers who felt reasonably happy with the overall flow chart in Fig. 6.1 had second thoughts when they tried to relate it to the program listing of 6.2. This is inevitable, because there is a missing link between the two, namely the detailed flow charts of each separate section of the program. Figures 6.4 to 6.8 show some of the detailed flow charts needed, but lack of space makes it impossible to reproduce all of them, so a certain amount of "unravelling" will still be necessary if any reader wishes to trace the operation of the complete program.

Let us start off with something easy, and have a look at how box 16 of Fig. 6.1 is turned into a 4040 program segment. Box 16 is a software implemented 12 bit binary counter routine which is updated each time the ENTER DATA or DUMP switches are pressed. The current count value is used during the ENTER DATA or DUMP program segments as a program memory address, and is displayed on the rightmost three display digits in hexadecimal.

Counters are implemented in 4040 software by using the ISZ (Increment and Skip if Zero) instruction which has the effect of incrementing the value of an internal four bit 4040 register by one, and jumping to a specified address if the contents of the register are not zero. If they *are* zero, the jump does not take place, and the next instruction in sequence is fetched. Figure 6.4 shows the implementation of the 12 bit address counter using ISZ,

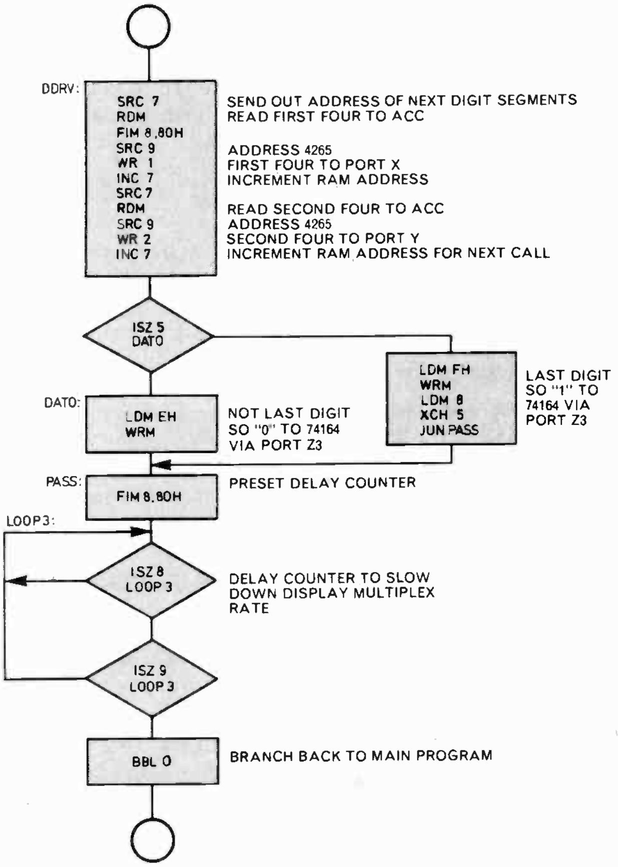


Fig. 6.5. Display driver subroutine. Refer to box 5 in CHOMP main flow chart, and address 00B1H on CHOMP program listing

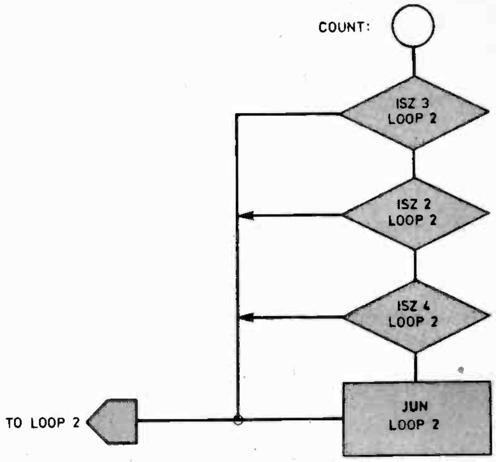


Fig. 6.4. Twelve bit address counter flow chart. Refer to box 16 in CHOMP main flow chart, and address 005EH. The CHAMP address counter should not be confused with the 4040 address counter

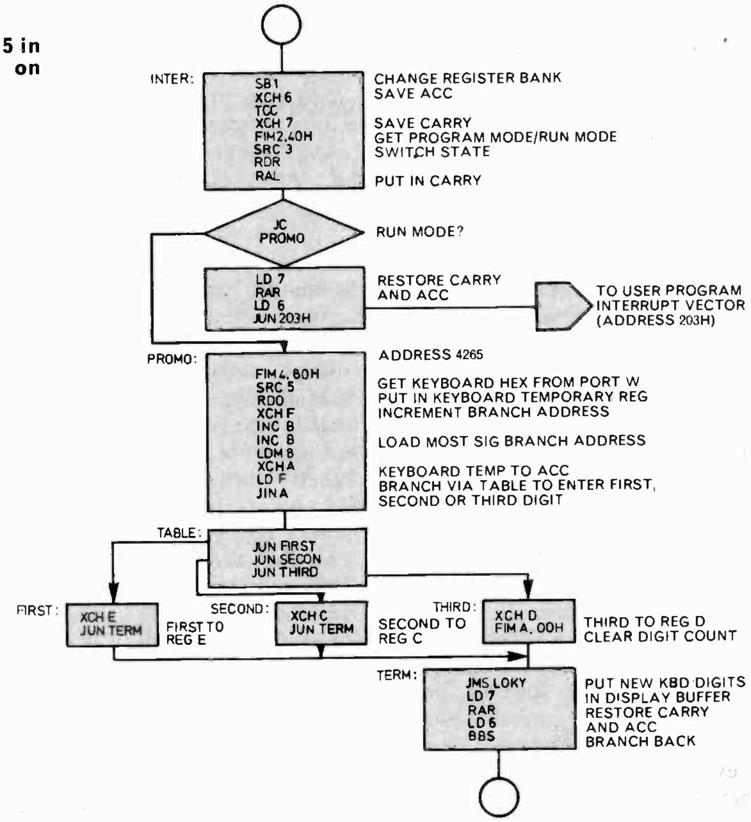


Fig. 6.6. Interrupt routine. Refer to boxes 17 and 18 in CHOMP main flow chart, and address 066H in CHOMP program listing

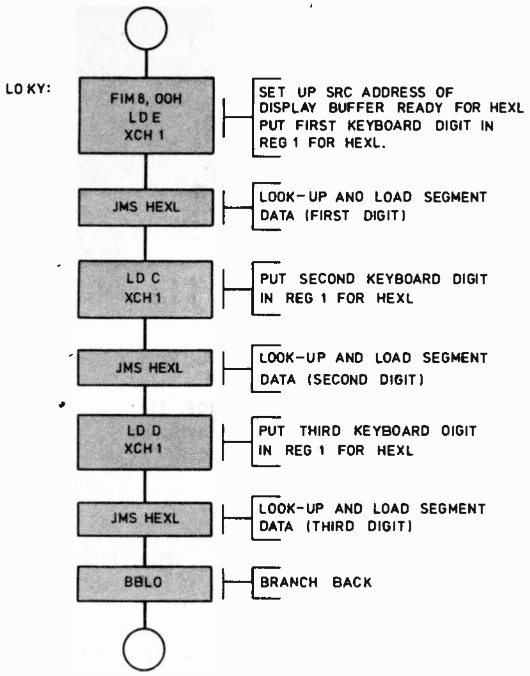


Fig. 6.7. Load keyboard subroutine. Refer to address 00CEH in CHOMP program

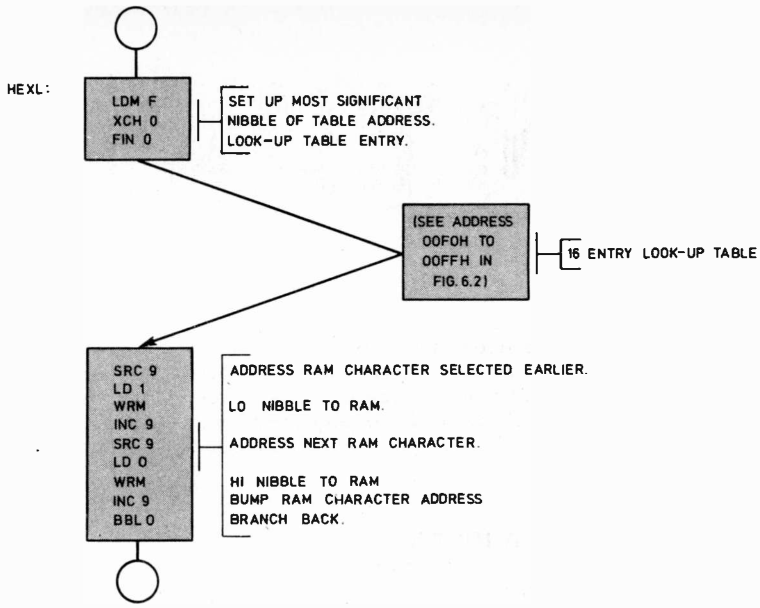


Fig. 6.8. Seven segment from hex look-up subroutine. Refer to address 00DDH in CHOMP program

the registers used being 3, 2, and 4 *in that order*. (The order is important because the high order address bits during an SRC instruction are taken from the lower register of a pair, and of course we use the lower eight bits of the counter as a SRC value when addressing program memory, before using RPM or WPM instructions.) The required 12 bit length of the counter is arranged by using three cascaded ISZ instructions, each with a common jump address, namely LOOP 2. You can probably see that Register 3 is incremented 16 times more often than Register 2, which itself is incremented 16 times more often than Register 4, in traditional binary counter fashion.

DISPLAY DRIVER

The subroutine DDRV is the full version of box 5 in Fig. 6.1, and its detailed flow chart is shown in Fig. 6.5.

This subroutine increments a counter (Register 7) twice each time it is called, and uses the counter contents as part of a SRC address to the data RAM display buffer (RAM chip 0, register 0). On each call it reads two four bit locations from the 4002, and sends their contents to the 4265 output ports X and Y which control the display segment lines. After doing this it increments another counter (Register 5) which it uses as a digit counter. This counter is preset to 8 hex (using LDM) when it reaches zero, and a logic one is placed on the 74164 shift register DATA INPUT via 4265 output Z3, using the WRM command. If this counter does not reach zero during a call then a logic zero is placed on the shift register data input.

You can probably see how this subroutine displays eight digits, one per call; and how it recycles to repeat the process over and over again. On seven out of eight calls it shifts a logic zero into the register, but on the eighth it generates a new "digit strobe" for the display, to replace the one which has just "dropped off the end" of the 74164.

INTERRUPT ROUTINE

Figure 6.6 shows the interrupt routine, INTER, which is boxes 17 and 18 on the overall flow chart of Fig. 6.1. The main thing of interest here is the use of a "Branch Table" accessed using the JIN (Jump Indirect) instruction to route the program flow to the correct segment depending on whether the current keyboard digit entry is the first, second, or third in sequence. Notice also that at the start of the routine the current accumulator and carry flip-flop contents are saved in registers 6 and 7 of Bank 1, to be restored at the end of the routine so that the main program flow can continue normally. A subroutine LOKY is used to enter the newly entered keyboard data into the display buffer.

The subroutine LOKY is itself shown in Fig. 6.7. It takes the contents of the three keyboard registers (E, C, and D) and converts their hexadecimal data into seven segment code using another subroutine HEXL.

HEXL itself is shown in Fig. 6.8, and as you can see it uses a FIN (Fetch Indirect) instruction to access a look-up table with sixteen entries. To convert hex to seven segment code, the hex is used as part of an indirect address so that the correct segment data can be "looked up" in the table. Table look-up is a powerful and simple technique which is very useful when converting data from one format to another. HEXL also loads the seven segment data into the 4002 RAM buffer register, at the appropriate address passed to it in registers 8 and 9 by the subroutine LOKY.

There are several other detailed flow charts required for a full understanding of CHOMP, and it would be excellent practice for CHAMP users to try and draw these up for themselves using Figs. 6.1 and 6.2 for reference. Don't be discouraged if it takes a while for the flash of inspiration to arrive, programming a microprocessor takes some getting used to, and is invariably a frustrating business at first, particularly for us "hardware people".

NEXT MONTH: Putting CHAMP to work