AN INTEL 8080 MICROPROCESSOR DEVELOPMENT SYSTEM

P.J. Horne
Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l’autorisation écrite du Directeur général du CERN, titulaire du droit d’auteur. Dans les cas appropriés, et s’il s’agit d’utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s’opposer à toute revendication qu’un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

© Copyright CERN, Genève, 1977

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use. If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.
ABSTRACT

The INTEL 8080 has become one of the two most widely used microprocessors at CERN, the other being the MOTOROLA 6800. Even though this is the case, there have been, to date, only rudimentary facilities available for aiding the development of application programs for this microprocessor. An ideal development system is one which has a sophisticated editing and filing system, an assembler/compiler, and access to the microprocessor application. In many instances access to a PROM programmer is also required, as the application may utilize only PROMs for program storage.

With these thoughts in mind, an INTEL 8080 microprocessor development system was implemented in the Proton Synchrotron (PS) Division. This system utilizes a PDP 11/45 as the editing and file-handling machine, and an MSC 8/MOD 80 microcomputer for assembling, PROM programming and debugging user programs at run time. The two machines are linked by an existing CAMAC crate system which will also provide the means of access to microprocessor applications in CAMAC and the interface of the development system to any other application.
CONTENTS

1. INTRODUCTION .................................................. 1

2. IMPLEMENTATION .................................................. 2

3. OPERATION OF THE LINK ......................................... 2
   3.1 System commands and error messages ......................... 2
   3.2 User command messages ................................... 3
      3.2.1 Memory Read/Write operations ................... 3
      3.2.2 Paper-tape control .................................. 4
      3.2.3 Program control ..................................... 5
      3.2.4 PROM programming .................................. 5

4. MCS RESIDENT ASSEMBLER ....................................... 6
   4.1 Installing the assembler .................................. 6
   4.2 Assembling a program .................................... 6
   4.3 Programming with macros .................................. 7
   4.4 Assembler errors .......................................... 7

5. USER PROGRAM CONTROL OF INPUT/OUTPUT (I/O) ................. 8
   5.1 Termination of the program ................................ 9
   5.2 MCS utility programs ..................................... 9

6. AN EXAMPLE OF USING THE DEVELOPMENT SYSTEM ............... 10

APPENDIX I ................................................................ 16
   1.1 Starting the MCS monitor ................................ 16
   1.2 I/O device assignment ...................................... 16
   1.3 Starting the PDP 11/45 ..................................... 17

APPENDIX II ................................................................ 18
   INTEL 8080 INSTRUCTION SET ................................. 18
1. INTRODUCTION

Program development is the most time-consuming (and therefore expensive) section of the majority of microprocessor applications. To keep this time to a minimum, it is necessary to have as good a development system as possible to aid the user in implementing his application programs for a particular microprocessor (in this case the INTEL 8080). On the arrival of the INTEL 8080 on the market, the only development system available was the MCS 8/MOD 80 microprocessor development system. This microcomputer was acquired by the Booster Group for aiding the development of software for a particular application. However, it soon became apparent that if any extension of the application was required or any other application arose, a more sophisticated development system would be needed.

By program development is meant the following sequence of steps:

1) specify the required actions in the form of a flow chart or high-level language;
2) translation of this chart or high-level language into the assembler language of the particular microprocessor;
3) "edit" the program into a source file;
4) "assemble" the source file to obtain the binary file;
5) "on-line" debugging of the program.

The time required to perform steps 1 and 2 is independent of the development system (assuming no high-level language compiler is available), while steps 3, 4, and 5 are system-dependent.

A good development system is one that gives good support to the user during steps 3, 4, and 5 of the program development, i.e. it has good file-handling, editing, assembling, and debugging facilities. The MCS system has a good assembler and debugging facility but a poor editor and no file-handling (as it has no mass storage device such as a disk) except that of paper-tape reading and punching. There was, however, an extremely good development system for the PDP 11 minicomputers near at hand, which has sophisticated file-handling and editing facilities. It was therefore decided to "link" the MCS to the PDP 11/45 to obtain a more sophisticated microprocessor development system (see Fig. 1). As the PDP 11/45 MCS was already connected to a CAMAC crate system, its interface to the MCS was relatively simple.

With this link implemented, program development on the MCS could be improved (time-wise) by at least a factor of 50 and would enable the user to utilize all the I/O devices of the PDP 11/45 (which are: two DEC packs, two DEC tapes, a teletype, and a line printer) thus providing a powerful microprocessor development tool achieved by the use of existing equipment and a very simple interface module.

It is hoped to implement this microprocessor development system with a NORD 10 mini-computer, in place of the PDP 11/45, in such a way that it may be used in a time-sharing mode. The NORD 10 being the software development machine for the new PS control system, the change from the PDP 11/45 to the NORD 10 will require no change in hardware because of the use of CAMAC as the interface medium.

On the software front the incorporation of a high-level language (e.g. CORAL 66, BCPL, PL/M) for microprocessors into the development system is part of the on-going implementation. An ideal language should be easy for the non-computer scientist to learn. It should
aid him in writing structured programs and, of course, be available on a minicomputer. It should also be possible to include into the language algorithms that are specific to the particular application [domain specific features]. For example, for CAMAC applications the compiler must know the keywords LAM, CAMAC, C, N, A, F, etc., and generate the required object code.

This report is intended to be an instruction manual and therefore includes all the information necessary for the user to develop an application program for the INTEL 8080. However, to prevent the report from becoming too large, reference to other instruction manuals will be made where necessary.

2. **IMPLEMENTATION**

The MCS contains a built-in operating system [MONITOR VERSION 3.0] on EPROM, which may be started by a simple bootstrapping via its front panel. This monitor allows the user to access the memory via dump or display commands, control a user program and program EPROMs, type 1702A, 2704 8 2708 INTEL EPROMs. A list of all the available commands is given in Section 3.1 of this report (copied from the INTELLEC 8/MOD 80 Operators Manual, Ref. 9).

It was decided to utilize this monitor -- and not treat the MCS as a special memory extension to the PDP 11/45 -- in such a way that all data transfers between the two machines would be in the form of ASCII strings which the monitor should interpret before giving a response. Each byte transferred is echoed by both machines in order to make synchronization of the link software to the monitor easy and provide some basic error-checking facility.

The software to implement the link was written in PL 11 and runs as a privileged task under the RSX 11D operating system on the PDP 11/45. PL 11 was chosen as the language for writing the link program because of its I/O support library and its speed of operation with the CAMAC system. As a block-structured language (based on PL 1) it has many similarities with NORD PL, making software transfer to that machine simpler than if PDP 11 assembly had been used.

3. **OPERATION OF THE LINK**

A series of preliminary operations must be performed prior to using the software link (all details are listed in Appendix I). An example of running this program is given in Section 6.

3.1 **System commands and error messages**

- **Starting the MCS**

  The user must first power-up the MCS and enable its monitor as described in Appendix I. Having started the system, the user should assign the list output, reader input, reader output, and teletype output to the CAMAC interface (see Appendix I).

  Make sure the CAMAC crate power supply is on.

- **Starting the PDP 11/45 and the link software**

  The link program must now be enabled. To do this the user should load disk pack DK12 (Microprocessor Development System) in drive 0 (DK0). The operating system RSX 11D on DK0
should be started by the usual bootstrap (Appendix I.3) and the link program PIL run, by
typing PIL (CR) on the console (DEC writer).

It announces itself by typing a header and then prints the prompt message `COMMAND>.
This is printed whenever the MCS monitor is ready to receive a command.

*Note:* If the user has forgotten to start the MCS, the program will abort and should be
restarted after the MCS has been started.

- **Transmission errors**

1) **Byte error**: This error can be due to a hardware fault or a mistype by the user in one
   of the commands. In either case the message "BYTE ERROR" is printed on the console
   and the program will attempt to make a recovery back to the command
   level. If this is not possible, the link program will be aborted and the
   user must restart the system. The program will also be aborted if there
   have been over five byte errors during one program development pass.

2) **Time out**: A time out will occur when the link program PIL has no reply from the MCS
   monitor within a predefined time. This may be due to CAMAC or the MCS
   being inadvertently switched off or to an error in typing a command message.
   In most instances there is no possible recovery from this error and the
   program will abort. However, if a recovery is possible, the message
   "TIME-OUT ERROR, PRESS CARRIAGE RETURN" is printed on the console, and
   the user need only type Carriage Return.

- **End of program**

To terminate the link program the user should type CONTROL AND Z (ɔ 2) after the prompt
message `COMMAND>.

3.2 **User command messages**

All commands to the MCS monitor consist of a single character followed by 0, 1, 2, or 3
hexadecimal parameters and terminated by either "SP" or "SP' & "CR". These commands may be
used only when the link software has been correctly enabled, as described at the beginning
of this section, and are made from the console.

*Note:* These commands may also be performed from the teletype connected to the MCS (see
Fig. 1) if the I/O assignment has not been performed.

3.2.1 **Memory Read/Write operations**

D ... Display memory in hexadecimal format (D low address, high address). Memory from
low address to high address is printed in hexadecimal format on the console.

F ... Fill memory (F low address, high address, data).
Memory from low address to high address is filled with the value specified by the
variable "data".

M ... Move (M low address, high address, destination address).
A block of memory from low address to high address is moved to location "destination
address".
Fig. 1 INTEL 8080 development system block diagram

S ... Substitute \([S \text{ address (SPACE) (CR)}]\).
Memory at address is printed on the console and may be modified by typing the new data. Termination with Space and Carriage Return opens the next location whilst only Carriage Return terminates the command.

L ... Load (L offset address).
A hexadecimal formatted file (Ref. 9), generated by the assembler program during pass A, is loaded into memory. The load address may be altered, from that specified on the file, by setting "offset address" to the required value.

3.2.2 Paper-tape control

R ... Read a hexadecimal tape (R offset address).
A hexadecimal formatted tape (Ref. 9) is read from the high-speed reader (on the PDP 11/45) into memory. The address where the data will be stored can be altered, from that specified on the tape, by setting the offset address to the required value.
W ... Write hexadecimal (W low address, high address).

Memory from low address to high address is punched in hexadecimal format on the slow-speed punch (connected to the MCS teletype).

E ... End (E address).

An end-of-file mark is punched on the slow-speed punch followed by 60 null characters. If address is non-zero the monitor will transfer control to the specified address after loading the tape by the R command.

N ... Punch null (N).

60 null characters are punched on the slow speed punch.

3.2.3 Program control

G ... 070 (G address, BKPT1, BKPT2).

Program control is transferred to address specified, and breakpoints are set at BKPT1 and BKPT2. When a breakpoint is encountered, the address is printed and control returned to the command level.

X ... Examine registers (X register ident. or "CR").

Register contents are displayed on the console and may be modified if a register ident. (A, B, C, D, E, H, L, SP, PC) is typed followed by "SP" & "CR". If only X followed by "CR" is typed, all the register contents will be printed on the console.

3.2.4 PROM programming

a) 256-byte PROMs:

P ... Program from (P low address, high address, PROM address).

A 1602A or 1702A PROM is programmed with data, from memory location low address through high address, beginning at PROM address.

T ... Transfer from PROM to memory (T address).

The contents of the PROM are loaded into memory starting at address and continuing for the next 256 bytes.

C ... Compare PROM with memory (C address).

256 bytes of memory, beginning at the specified address, are compared with a PROM in the programming socket, and miscompares are printed.

b) 512- and 1024-byte PROMs:

The programmer for these two types of PROMs (2708 and 2704) has its own monitor which must be started before programming can commence. The initialization of this monitor is performed by tapping "PROM" after the prompt message COMMAND. One now has a further five commands to aid in programming PROMs.

P ... Program PROM (P low address, high address, PROM address).

This command is the same as for the 256-byte PROM, except that the PROM must be in the 512/1024 programming socket.

T ... Transfer from PROM to memory (T <PROM>, address).

Transfer data from the PROM in the 512/1024 programming socket to the specified memory address. <PROM> must equal 2704 or 2708.
C ... Compare PROM contents with memory (C <PROM>, address).
    Compare the 512 or 1024 bytes of data starting at address with the contents of
    the PROM.

E ... Check for an empty PROM (E <PROM>).
    Read the contents of the PROM and print out the number of already programmed
    locations.

Q ... "Quit"; leave this programming monitor and return.

4. MCS RESIDENT ASSEMBLER

    To assemble an INTEL 8080 assembler language program\textsuperscript{111}, a certain sequence of operations
    must be adhered to. First, the source program must be present on the system disk in a file
    under: UIC:[200,200]\textsuperscript{112}. The user can input this program to the system by means of the RSX 11D
    text editor (Ref. 12, chapter 6), having written the source file, the assembler must now be
    installed in the MCS memory.

4.1 Installing the assembler

    To install the assembler, the link program must be operational (see Section 3) and, when
    the prompt message occurs, the user should type ASS(CR). This initiates the loading of the
    assembler, stored in file DKO:[200,200] ASSEMBLER. INT, into the MCS. When it is ready the
    assembler announces itself by printing ASSEMBLER> on the console, inviting the user to input
    the complete file name (DKO:[200,200] FILENAME. INT) of the stored program. This file name
    is written by the user on entry to the text editor (see Section 6). All programs to be as-
    sembled by this system must be terminated with four stars (****).

4.2 Assembling a program

    The first pass of the program through the assembler is automatic and will set up the
    symbol table. The user is then invited to type a number which will control the second pass,
    e.g.

    i) If the user types 2(CR), the program will be assembled and a listing sent to the line
       printer.

    ii) 3(CR) will again cause the program to be assembled, but this time a hexadecimal for-
        matted paper tape will be punched on the slow-speed punch (the user should remember to
        switch this on).

    iii) Typing 4(CR) instructs the link program to assemble the specified program and generate
         a hexadecimal formatted file (named TEMP. INT) on disc DKO.

    iv) If 0(CR) is typed, the assembler program is terminated and the message RESET MCS &
        REASSIGN OUTPUTS, PRESS CR WHEN READY is printed.

    The following is a listing of a possible run of the link program when an assembly is
    required:
MCR> PIL (CR)
   PDP 11/45 TO MCS 8/MOD 80 SOFTWARE LINK
Command> ASS (CR)
Assembler> DKO:[200,200] TESTFT.INT (CR)
END OF PASS 1
   PASS = 2(CR)
   PASS = 4(CR)
   PASS = 0(CR)
RESET MCS & REASSIGN OUTPUTS, PRESS CR WHEN READY (CR)
COMMAND>.
PROGRAM TERMINATED

The control and Z (or Z) is the means of exiting from the program LINK/MC.

A brief description of the errors that are detected by the assembler now follows. These occur only during pass 2 of the assembler and are printed as single-letter codes by the line printer.

4.3 Programming with macros

The INTEL 8080 assembler includes the facility of implementing macros in the user program (Ref. 11, pp. 43-48). A macro is the means of generating a group of instructions from a symbol (the macro name) appearing in the code field. In this way one can introduce domain specific instructions into the assembler language.

To use this facility with the development system a macro "call" must be preceded by the symbol '#', e.g.

\[ \text{MNAME macro : Macro name = MNAME} \]
\[ \text{: List of instructions} \]
\[ \text{ENDM : End of macro MNAME} \]
\[ \text{PROG: ... : Main program commences here} \]
\[ \text{: List of instructions} \]
\[ \text{#MNAME : Load macro MNAME here.} \]

The '#' symbol informs the link program to expect multiple line replies from the assembler running in the MCS memory.

4.4 Assembler errors

A ... Address error.
   This indicates that the address referenced by a JUMP or CALL instruction is not in the range 0 to 65535.

B ... Balance error.
   This error indicates that parentheses in an expression are unbalanced, or that the quotes in an ASCII string are unbalanced.

E ... Expression error.
   This usually occurs because of a missing operator, omitted comma, or misspelt op-code.
F ... Format error.
   This may be caused by a missing operand or an extraneous operand.

I ... Illegal character.
   This indicates that an invalid ASCII character is present in the statement. It is
   also caused by a numeric character which is too big for the base of the number in
   which it occurs.

M ... Multiple definition.
   Two or more labels exist which are identical or not unique in the first five
   characters.

N ... Nesting error.
   An IF, ENDF, MACRO, or ENDM statement is improperly nested.

P ... Phase error.
   The value of an element being defined changes between passes 1 and 2 of the assembly.

Q ... Questionable syntax.
   Caused usually by omitting an op-code.

R ... Register error.
   This indicates that a register specified for an operation is invalid for that
   operation.

S ... Stack overflow.
   This indicates that the assembler's internal expression evaluation stack became
   too large and overflowed the memory available to the assembler. It may be caused
   by using extremely long character strings, too many nested macros, too many nested
   IF statements, or expressions which are too complex.

T ... Table overflow.
   The assembler's symbol table space has been exhausted, usually caused by too many
   symbols. (The MCS has only 8K of RAM which set an upper limit on the number of
   symbols at ~ 52.)

U ... Undefined identifier.
   A symbol used in an operand field has not been declared.

V ... Illegal value.
   The value of an operand or expression is out of range.

5. USER PROGRAM CONTROL OF INPUT/OUTPUT (I/O)

   When a user wishes to have I/O to the console from his INTEL assembler program running
on the MCS, there is a certain protocol he must follow.

   There are four characters used by the link software for message transfer control. These
are the "!", ".", "-" and ">" characters. The first three control characters cannot in gen-
eral be used by the user for controlling his I/O because of their utilization by the system.

   When a ":" or "." occurs after a line feed, the link software will assume End of Message
and issue the prompt message COMMAND>. The "." character is used by the MCS monitor and the
":" is generated by the 512/1024 PROM programming monitor.
The character "-" is used during the S and X monitor commands for inputting variables (see Sections 3.1.1 and 3.1.3, respectively).

The user-controlled I/O character is a ">"; this allows him to have an interactive mode of operation with the console and his program. Whenever the user program generates this character for output, control is returned to the console and an input expected. An example of using this control character is given in Section 6.

5.1 Termination of the program

The user should always end his program by one of the following commands:

i) A direct jump to the monitor restart address (3C43H), i.e. JMP 3C43H.
ii) Use of the control character ">" to demand an input, and then type ":CR:" which will initiate a return to the command level.

In both of these cases, a "-" followed by the prompt message COMMAND> will be printed on the console, informing the user of the termination of his program.

5.2 MCS utility programs

To aid the user with arithmetic and I/O operations, a number of utility programs are available. These utility programs are resident in the MCS and may be invoked by a call to the relevant addresses as described below.

DIV ... (Address = 30A3H). Divide a floating-point number in B-H-L by one in C-D-E and put result in C-D-E. If A = 1 on return, the C-D-E number was equal to zero.

ADD ... (Address = 3140H). Add a floating-point number in B-H-L to one in C-D-E and place result in C-D-E.

SUBB ... (Address = 3131H). Subtract a floating-point number in C-D-E from one in B-H-L and place result in C-D-E.

FLOT ... (Address = 311BH). Convert an unsigned 16-bit number in C-D-E to a floating-point number in C-D-E where

i) C contains a 6-bit exponent (0-5) and two sign bits, bit 6 for exponent (SE = 0 for +ve) and bit 7 for the mantissa (SM = 0 for +ve);

ii) D contains the MSB of the mantissa (as .MSB);

iii) E contains the LSB of the mantissa.

FIX ... (Address = 31DEH). Fix a floating-point number in C-D-E to a 16-bit unsigned number in D-E. A = 0 if result is all right.

FNEG ... (Address = 3133H). Negate a floating-point number in C-D-E.

FBCD ... (Address = 329CH). Convert a floating-point number in C-D-E to a BCD number in C-D-E as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>C</td>
<td>SM SE EH EH EL EL EL</td>
</tr>
<tr>
<td>D</td>
<td>X4 X4 X4 X4 X3 X3 X3 X3</td>
</tr>
<tr>
<td>E</td>
<td>X2 X2 X2 X2 X1 X1 X1 X1</td>
</tr>
</tbody>
</table>
where SM is the sign of the mantissa (0 is positive) and SE is the sign of the exponent (0 is positive). The largest number is therefore 9.999 E39, where X4 = X3 = X2 = X1 = 1000 (9), EH = 11 (3), EL = 1001 (4).

PDEC ... (Address = 33CAH). Print a BCD floating-point number in C-D-E on the console device.

RDEC ... (Address = 3340H). Read a decimal number in the range 0-32767 from the console and convert to a binary number in H-L. The input may be terminated by either "space", *, /, +, or "CR" in which case the C register contains 0, 1, 2, 4 or 8, respectively. If no number is input and only Carriage Return (CR) typed, control is returned to the monitor.

BNBCD... (Address = 342FH). Convert a binary number in D-E to five BCD digits pointed at by H-L. On entry to the routine, the H-L register pair must point at the storage space for the BCD digits.

CO ... (Address = 3809H). Console output routine: a value in the C register is printed on the assigned console device.

CI ... (Address = 3803H). Console input routine: a character received from the selected console device is returned to the caller in the A register.

PO ... (Address = 380CH). Punch output routine: a character in C is punched on the slow-speed punch.

FMPY ... (Address = 3000H). Floating-point multiply: a number in C-D-E is multiplied by a floating-point number in B-H-L and the result placed in C-D-E.

EXIT ... (Address = 3C45) Restarts the MCS monitor and returns program control to the command level.

6. AN EXAMPLE OF USING THE DEVELOPMENT SYSTEM

The example described in this section illustrates the use of the development system for writing an INTEL 8080 assembler language program, and its subsequent assembling and running on the MCS.

The sequence of events is shown in the form of a flow chart (Fig. 2) and a listing with comments referenced by the number at its side (see below).

RSX-004A

| MCP>LOG *** MICROPROCESSOR DEVELOPMENT SYSTEM *** |
| 1 MCP>ED1 |
| 2 EDI>TESTFT.INH |
| [CREATING NEW FILE] |
| INPUT |
| 3 | PROGRAMME TO READ TWO DECIMAL NUMBERS AND PERFORM |
| | THE REQUIRED FLOATING POINT ARITHMETIC OPERATION. |
| | E.G USER TYPES 8*8 AFTER PROMPT CHARACTER '.' AND |
| | THE RESULT = G.400 001 IS TYPED BY PROGRAM. |
| | FMPY EQU 3000H ;FLOATING POINT MULTIPLY ROUTINE |
| | FDIV EQU 3043H ;FLOATING POINT DIVISION ROUTINE |
| | FADD EQU 3140H ;FLOATING POINT ADDITION |
| | FSUB EQU 3130H ;FLOATING POINT SUBTRACTION |
| | FBCD EQU 323CH ;FLOATING POINT TO BCD CONVERSION |
| | FLOT EQU 3110H ;INTEGER TO FLOATING POINT |
CO EDU 3C55H :CONSOLE OUTPUT ROUTINE
CRLF EDU 3C00H :PRINT 'CR', 'LF' ON CONSOLE
PDEC EDU 33CAH :PRINT BCD NUMBER ON CONSOLE
RD EC EDU 334BH :READ A DECIMAL NUMBER FROM CONSOLE
EXIT EDU 3C43H :MONITOR RESTART ADDRESS

ORG 20H :PROGRAM LOADS AT THIS ADDRESS
START: LXI SP, 1500H :SET STACK
CALL PROMPT :WAKE UP CONSOLE FOR INPUT
CALL RDEC :READ A DECIMAL NUMBER FROM CONSOLE
MOV A,C
PUSH PSW
PUSH H
CALL RDEC :READ SECOND NUMBER
POP D
CALL FLOT :FLOAT NUMBER INTO C-D-E
PUSH D
PUSH B
CALL FLOT :FLOAT SECOND NUMBER
MOV A,C
POP B
MOV B,C
POP H
MOV C,A :FIRST FLOATING POINT NUMBER
IN H-L & SECOND IN C-D-E
POP PSW :GET OPERATOR VALUE (*, +, -, /)
CP 1 1 :TEST VALUE AND BRANCH TO POP CALL SEQUENCE
JNZ ST1
CALL MPY
JMP ST4
ST1: CP 1 2
JNZ ST2
CALL FDIV
JMP ST4
ST2: CP 1 4
JNZ ST3
CALL FADD
JMP ST4
ST3: CP 1 5
JNZ ST4
CALL FSUB
CALL FBOD :CONVERT RESULT TO BCD IN REGISTERS C-D-E
MOV B,C
MVI C,'*' :PRINT '*' ON CONSOLE
CALL CO :PRINT '*' ON CONSOLE
MOV C,B
CALL PDEC
CALL CRLF
CALL EXIT :EXIT FROM PROGRAM TO COMMAND LEVEL

; ROUTINE TO WAKE UP CONSOLE AND PROMPT USER FOR INPUT
; PROMPT:
MVI C,'>' :PRINT '>' ON CONSOLE
PET
END

PDP 11/45 TO MFS & MDO TO SOFTWARE LINK:

COMMAND: AS5
7 ASSEMBLER: D0: (200.008 TEST.T, INT

END OF PASS 1
0 PASS= 2
9 PASS= 4
10 PASS= 0
RESET MFS & REASSIGN OUTPUTS, PRESS CR WHEN READY
11 COMMAND>
12 COMMAND>D20.2F
  8020 31 00 15 CD 76 00 CD 40 33 79 F5 E5 CD 40 33 D1
13 COMMAND>G20
   >8*8
  = 6.400  E01
14 *
15 COMMAND>^Z

PROGRAM TERMINATED

MCR>

1) Run the editor program by typing EDI(CR).
2) Input the file name of the program to be written [DKI: TESTFT.INT]. The extension name .INT should be used to inform the user that the program is written in INTEL 8080 assembler code.
3) Type in program, terminating each line by Carriage Return.
4) Terminate program by **** (CR) and then another Carriage Return to terminate input.
5) Exit from the editor by typing EX (CR)
6) Run the link program by typing PIL (CR).
7) Load the assembler into the MCS; when prompted by the message ASSEMBLER> type the file name (with UIC = [200,200]).
8) An assembly listing is printed on the line printer (see assembly listing).
9) A hexadecimal formatted file is generated on disc.
10) Terminate the assembler and cause the message following to be printed.
11) Read the hexadecimal formatted file (from DKO:[200,200] TEMP.INT) and load into the MCS memory.
12) Print memory locations 20 to 2F.
13) Start program at location 20.
14) Program was terminated by a call to the monitor restart address; see program listing for comments.
15) The link program is terminated by typing control Z.
Fig. 2 A flow chart example of using the development system

ASSEMBLY LISTING OBTAINED DURING PASS 2

1 : PROGRAMME TO READ TWO DECIMAL NUMBERS AND PERFORM
   THE REQUIRED FLOATING POINT ARITHMETIC OPERATION.
   E.G USER TYPES 0#8 AFTER PROMPT CHARACTER '=>' AND
   THE RESULT -6.480 E01 IS TYPED BY PROGRAM.
   
2
3
4
5
6

7 3000 FMPY EQU 3000H : FLOATING POINT MULTIPLY ROUTINE
8 30A3 FDIV EQU 30A3H : FLOATING POINT DIVISION ROUTINE
9 3140 FADD EQU 3140H : FLOATING POINT ADDITION
10 313D FSUB EQU 313DH : FLOATING POINT SUBTRACTION
11 329C FBCD EQU 329CH : FLOATING POINT TO BCD CONVERSION
12 311B FLOT EQU 311BH : INTEGER TO FLOATING POINT
Acknowledgements

I would like to thank Dr. K.H. Reich and Mr. G. Baribaud for their support of this work, the PS/CCI group for the use of their PDP 11/45, and Mr. T. Lingjaerde for his useful comments on the form the report should take.
REFERENCES

1) Software becomes the challenge, Electronics 49 (No 8), 104-108 (1976).

2) R.D. Catterton and G.S. Carilli, Universal development system is aim of master-slave processors, Electronics 49 (No. 19), 91-96 (1976).


4) P.J. Horne, Implementation of a varilog for the PSB multipole power supplies using a microprocessor, CERN PS/BR Note 76-24 (1976).


7) Intel Corporation, 8008 and 8080 PL/M programming manual (Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051, USA, 1975).


9) Intel Corporation, INTELLEC 8/MOD 80 operators manual (Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051, USA, 1974).


11) Intel Corporation, INTEL 8080 Assembly language programming manual (Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051, USA, 1974).

APPENDIX I

1.1 Starting the MCS monitor

To start the monitor the user must load the JUMP 3800H instruction into memory at location 0 and press the RESET switch. The sequence of operations is as follows:
   i) Set the MEM ACCESS switch.
   ii) Set all the ADDRESS switches to 0, indicating memory location 0.
   iii) Press the LOAD switch.
   iv) Set the ADDRESS/INSTRUCTION/DATA switches to C3H (11000011 B).
   v) Press the DEP switch.
   vi) Press the INCR switch.
   vii) Set the ADDRESS/INSTRUCTION/DATA switches to 00H.
   viii) Press the DEP switch.
   ix) Press the INCR switch.
   x) Set the ADDRESS/INSTRUCTION/DATA switches to 38H.
   xi) Press the DEP switch
   xii) Reset the MEM ACCESS switch.
   xiii) Press the RESET switch.

1.2 I/O device assignment

After the last operation has been accomplished, the monitor will print 8080 VER 3.0 on the teletype (if it has been switched on!). If the user wishes to use the teletype for all communication with the MCS, he should use the commands as described in Chapter 4 of the Operators Manual (Ref. 9). However if the user wishes to utilize the PDP 11/45 to the MCS software development link, he must assign certain I/O devices to the CAMAC interface driver (which resides on an EPROM), by typing

```
.AP = 1(CR)  
.AR = 1(CR)  
.AL = 1(CR)  
.AC = 1(CR)  
```

(the monitor prints the dot).

After the last command the teletype is software-disconnected from the monitor (except the slow-speed punch) and can only be restarted by pressing the RESET switch on the front panel.)
I.3 Starting the PDP 11/45

i) Put disk DK12 in drive 0 and press LOAD switch.

ii) Press HALT switch on front panel.

iii) Set ADDRESS switches to 173030₄.

iv) Press LOAD ADDRESS switch.

v) Reset HALT switch.

vi) Press START switch.
APPENDIX II

INTEL 8080 INSTRUCTION SET*}

SILICON GATE MOS 8080

INSTRUCTION SET

The accumulator group instructions include ARITHMETIC and LOGICAL OPERATORS with DIRECT, INDIRECT, AND IMMEDIATE addressing modes.

MOVE, LOAD, and STORE instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using DIRECT, INDIRECT, and IMMEDIATE addressing modes.

The ability to branch to different portions of the program is provided with JUMP, JUMP CONDITIONAL, and COMPUTED JUMPS. Also the ability to CALL to and RETURN from subroutines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as STACK manipulation and DOUBLE ADD instructions extend both the arithmetic and interrupt handling capability of the 8080. The ability to INCREMENT and DECREMENT memory, the six general registers and the accumulator is provided as well as EXTENDED INCREMENT and DECREMENT instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to ROTATE the accumulator LEFT or RIGHT through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080 instruction set.

The following special instruction group completes the 8080 instruction set: the NO-OP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

Data and Instruction Formats

Data in the 8080 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

\[ \text{DATA WORD} \]

\[ D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \]

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

One Byte Instructions

\[ D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \]

OP CODE

TYPICAL INSTRUCTIONS

Register to register, memory reference, arithmetic or logical, rotate return, PUSH, POP, ENABLE or DISABLE INTERRUPT INSTRUCTIONS

Two Byte Instructions

\[ D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \]

OP CODE

OPERAND

Immediate mode or I/O instructions

Three Byte Instructions

\[ D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \]

OP CODE

JUMP, CALL or DIRECT LOAD

LOW ADDRESS OR OPERAND 1 AND STORE INSTRUCTIONS

HIGH ADDRESS OR OPERAND 2

For the 8080 a logic "1" is defined as a high level and a logic "0" is defined as a low level.

* Copied from the INTEL 8080 Microcomputer System User's Manual)
## INSTRUCTION SET

### Summary of Processor Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Instruction Code(1)</th>
<th>Clock(2)</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Instruction Code(1)</th>
<th>Clock(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVX, A, M</td>
<td>Move register to register</td>
<td>0 1 0 0 0 0 5 5 5 5</td>
<td>1 0 0</td>
<td>RZ</td>
<td>Return on zero</td>
<td>0 1 0 0 0 0 0 0 5 11</td>
<td></td>
</tr>
<tr>
<td>MOVY, A</td>
<td>Move register to memory</td>
<td>0 1 1 0 0 0 5 5 5 7</td>
<td>0 0 0 0</td>
<td>RNZ</td>
<td>Return on not zero</td>
<td>0 1 1 0 0 0 0 0 5 11</td>
<td></td>
</tr>
<tr>
<td>MOVX, M</td>
<td>Move register to register</td>
<td>0 1 1 1 0 0 5 5 5 7</td>
<td>0 0 0 0</td>
<td>RP</td>
<td>Return on positive</td>
<td>0 1 1 1 0 0 0 0 5 11</td>
<td></td>
</tr>
<tr>
<td>HL</td>
<td>Move to memory</td>
<td>1 1 1 1 1 1 0 5 5 3</td>
<td>0 0 0 0</td>
<td>RM</td>
<td>Return on minus</td>
<td>1 1 1 1 1 0 0 0 5 11</td>
<td></td>
</tr>
<tr>
<td>N0</td>
<td>Move to register</td>
<td>0 1 0 0 0 0 0 0 1 0</td>
<td>1 0 0 0</td>
<td>RPE</td>
<td>Return on parity even</td>
<td>0 1 0 1 0 1 1 0 5 11</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>Move to register</td>
<td>0 1 0 0 0 0 1 0 1 0</td>
<td>1 0 0 0</td>
<td>RPD</td>
<td>Return on parity odd</td>
<td>0 1 1 0 0 0 0 0 5 11</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>Increment register</td>
<td>0 0 0 0 0 0 0 1 0 5</td>
<td>1 0 0 0</td>
<td>RST</td>
<td>Reset</td>
<td>1 1 1 A A 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>DCR</td>
<td>Decrement register</td>
<td>0 0 0 0 0 0 0 1 1 5</td>
<td>1 0 0 0</td>
<td>INH</td>
<td>Inhibit</td>
<td>1 1 1 0 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>INH</td>
<td>Increment memory</td>
<td>0 0 0 1 0 1 0 1 0 10</td>
<td>1 0 0 0</td>
<td>OUT</td>
<td>Output</td>
<td>1 1 1 0 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>Add immediate to A</td>
<td>0 1 0 0 0 1 0 5 5 4</td>
<td>1 0 0 0</td>
<td>LX</td>
<td>Load immediate register</td>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract register from A</td>
<td>0 1 0 0 0 1 0 5 5 4</td>
<td>1 0 0 0</td>
<td>LX</td>
<td>Load immediate register</td>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>SBB</td>
<td>Subtract register from A</td>
<td>0 1 0 0 1 1 1 5 5 4</td>
<td>1 0 0 0</td>
<td>LX</td>
<td>Load immediate register</td>
<td>1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

### FOOTNOTES

1. DD5 or DD5 - 000 B - 001 C - 010 D - 011 E - 100 H - 101 L - 110 Memory - 111 A.
2. Two possible cycle times, 0/11 indicate instruction cycles dependent on condition flags.
**JUMP DECREMENT**

- C3 JMP 05 DCR B
- C2 JNZ 00 DCR C
- CA JZ 15 DCR D
- D2 JNC 1D DCR E
- DA JC 25 DCR H
- E2 JP 2D DCR L
- EA JPE 35 DRM D
- F2 JP 30 DCR A
- FA JM 32 DCX SP

**MOVE IMMEDIATE**

- 06 MVI B, D8 DCX B
- 0E MVI C, 1B DCX D
- 16 MVI D, 2B DCX H
- 1E MVI E, 3B DCX SP
- 26 MVI H, 08 DCX B
- 2E MVI L, 1B DCX D
- 36 MVI M, 2B DCX H
- 3E MVI A, 3B DCX SP

**LOAD IMMEDIATE**

- 04 INX B 00 NOP
- 0C INX C 09 DAD B
- 14 INR D 19 DAD D
- 24 INR H 29 DAD H
- 34 INR L 39 DAD SP

**LOAD/STORE**

- 0A LDAX B 02 STAX B
- 1A LDAX D 12 STAX D
- 2A LHLD ADR 22 SHTLD ADR
- 3A LDA 32 STA ADR

**ADD**

- C6 ADI C7 RST 0
- CE ACI CF RST 1
- D6 SUI D7 RST 2
- DE SBII DF RST 3
- E6 ANI E7 RST 4
- EE KRI EF RST 5
- F6 ORI F7 RST 6
- FE CPI RR RST 7

**ACCUMULATOR**

- 80 ADD A F3 D1
- 81 ADD C FB EI
- 82 ADD D F8 E1
- 83 ADD E 84 ADD H
- 85 ADD L 86 ADD M
- 87 ADD A 88 ADD C
- 89 ADD D 8A ADD E
- 8B ADD H 8C ADD L
- 8D ADD M 8E ADD A
- 8F ADD C 0BDH

**STACK**

- 58 MOV E,B 59 MOV E,C
- 5A MOV E,D 5B MOV E,E
- 5C MOV E,F 5D MOV E,FL
- 5E MOV E,M 5F MOV E,A
- 60 MOV H,B 61 MOV H,C
- 62 MOV H,D 63 MOV H,E
- 64 MOV H,F 65 MOV H,H
- 66 MOV H,M 67 MOV H,A
- 68 MOV L,B 69 MOV L,C
- 6A MOV L,D 6B MOV L,E
- 6C MOV L,F 6D MOV L,L
- 6E MOV L,M 6F MOV L,A
- 70 MOV M,B 71 MOV M,C
- 72 MOV M,D 73 MOV M,E
- 74 MOV M,F 75 MOV M,L
- 76 MOV M,A 77 MOV M,M
- 78 MOV M,C 79 MOV M,D
- 7A MOV M,H 7B MOV M,A
- 7C MOV M,A 7D MOV M,A
- 7E MOV M,A 7F MOV M,A

** operators**

- 98 SBB E 9C SBB H
- 9D SBB L 9E SBB M
- 9F SBB A
- A0 SBB B AND C
- A1 SBB C OR C

**specials**

- 27 DAA* 2F CMA
- 37 STC1 6B MOV LE
- 6C MOV LH
- 60 MOV LL
- 6B ORA B 6E MOV LM
- 6F MOV LA
- 70 MOV MB 71 MOV MC
- 72 MOV MD 73 MOV ME
- 74 MOV MH 75 MOV ML
- 77 MOV MA 79 MOV AC
- 76 MOV OB 78 MOV AB
- 7A MOV AD 7B MOV AE
- 7C MOV AH 7D MOV AL
- 7E MOV AM 7F MOV AA

** PSEUDO INSTRUCTION**

- 00 NOP 09 DAD B
- 10 HLT 19 DAD D
- 1A RET 29 DAD H
- 1B DCX B 39 DAD SP

**CALL**

- 0B CALL 07 RLC
- 1B CALL 0F RRC
- 2B CALL 17 RAL
- 3B CALL 1F RAR
- 4B CALL 21 LXI H
- 5B CALL 25 LXI D
- 6B CALL 31 LXI SP
- 7B CALL 00 NOP
- 8B CALL 09 DAD B
- 9B CALL 19 DAD D
- AB CALL 29 DAD H
- AC CALL 39 DAD SP

**DECIMAL**

- D6 D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity.
- DB = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity.
- * = al Flags (C,Z,S,P) affected
- t = only CARRY affected

- Adr = 16 bit address
- ** = all Flags except CARRY affected.
- (exception: INX & DCX affect Flags)

- 105D
- 11011B
- 00110B
- 'A' B

- 720
- 720
- 1A