Computer Sciences Department
The University of Wisconsin
Madison, Wisconsin 53706

A MINI-COMPUTER SUPPORT SYSTEM

by

E. J. Desautels

Technical Report #116

February, 1971

Introduction

While teaching Computer Sciences 536, Programming Systems and Computer Organization, the author launched a class project which aroused the interest of others besides the participants. In order to document what was done and what remains to be done, and to whom credit is due, the author has prepared this report.

Acknowledgments

The Digital Equipment Corporation, in the person of Mr. Craig Zamzow, was generous in allowing us the use of a PDP8/L for a period of two months. Having access to a real mini-computer contributed greatly to the enthusiasm of the participants, and the success of their efforts.

The University of Wisconsin Computing Center assisted in permitting us to place the PDP8/L in one of their rooms. The quality of the UWCC 1108 software, in particular the recently released catalogued file facility, greatly eased our task.

The Space Sciences and Meteorology Computing Facility allowed us the use of their Raytheon 440, which provided us with the required paper tape punch facility.

Mr. Mark Birnbaum, a graduate student in the Computer Sciences Department, presented the class with a lecture on the PDP8, a sample program, and on-line debugging tools. He also made valuable suggestions to the instructor.

Finally, credit should go to those who did the work - the students enrolled in CS536, Fally, 1970. These individuals (ranked alphabetically, without regard to the magnitude of their contributions) are: M. L. Athavale, V. Chow, G. Eichler, V. Erickson, A. Finger, S. Gilheany, J. Hicks, S. Jacobs, L. Jensen, R. Kenison, R. Kenyon, R. Koch, J. Lanuti, C. Madsen, R. Mills, D. Nissen, J. Oakley, D. Praizler, P. Reid, K. So, S. Stone, W. Tan and E. Zenk.

The task

Assume that you have access to a large computing facility, such as the UWCC's 1108 system. Then assume that you have (or are contemplating having) access to a mini-computer. Several questions then arise:

1. In what manner can the large and the small systems complement each other?
2. What are the practical difficulties in using the large system to develop software for the small system?

3. What are the economics of the tradeoffs, and what pitfalls is one likely to encounter?

In considering these and related questions, one comes to the notion of a "support system".

Definition: a support system is that set of tools which facilitate or improve the ultimate use of a given computer, which involve the use of a larger, not necessarily related computer.

In the sense of the above definition, support systems already exist for many machines, such as the CDC peripheral processors (supported by and used with CDC 6400/6500 and 6600 processors) and the recently announced IBM System 7 which has support systems on the 360 and 1800 and 1130. Locally, one finds that the UNIVAC 1108 Assembler has characteristics which might facilitate using it as an assembler for a different machine. However UNIVAC itself appears not to be using the facility (i.e., the UNIVAC 9200 and 9300's are supported by themselves, not by the 1108).

There are other proprietary support systems in use, as in the case of Applied Data Research (ADR), Princeton, N. J., which supports PDP-8's on a PDP-10. There exist partial support systems, such as a PDP-8 simulator on the Michigan Time-Sharing System.

Our objective was to produce a set of tools consistent with the above definition, and in keeping with the nature of the course. Fortunately, the one fits in very well with the others. As the project developed, the following tools described in this report were developed:

1. PDP-8 simulator
2. A machine operator simulator
3. PDP-8 PAL III assembler
4. ASCII and Fieldata code conversion packages
5. Paper tape to Fastrand storage routines
6. Fastrand storage to punched paper tape routines
7. Tape and core dump interpreter
8. Absolute program de-assembler

9. A quick-load facility
10. A BASIC-like compiler

These tools will be described later.

Ground rules

The participants were encouraged to use the most advanced features of the programming languages they chose to use, as well as advanced features of the operating system. Thus some of the programs written in Fortran V use features peculiar to Fortran V such as PARAMETER, DEFINE, END and ERR contingencies, reread, ENCODE/DECODE and FLD functions. Some of the programs were written in SNOBOL; a few were written in 1108 assembler language.

Items such as word size, memory size, size of the instruction set and other characteristics peculiar to the target machine (i.e. the PDP8) were to be treated as parameters. Thus the system was implemented in a manner which would facilitate adapting it for use with other mini-computers.

To the extent that the class had no prior experience with mini-computers, it was not always clear which items could meaningfully be parameters.

In some cases, individuals independently constructed routines with the same function and in other cases, small teams worked together on a task. Thus we actually wound up with two different and successful simulators (plus one which was not), and several assemblers, whereas the compiler was a team effort involving 2 teams of 3 people each.

The PDP-8 simulator

The memory and processor instructions are implemented, along with the input and output instructions relating to the Teletype terminal and its low-speed paper tape reader and punch. Information may also be entered via the simulated console switches.

Instructions relating to interrupts and to other input and output devices are not implemented. The implementation framework would make the inclusion of new I/O instructions straight-forward.

Interrupt handling is not implemented due to fundamental uncertainty relating to PDP-8 user program design. We could simulate interrupts by using a simulated-time clock set to interrupt 100 milliseconds worth of memory cycles after an output instruction - but this is very expensive on the 1108 (i.e. slaving the 1108 to a teletype). If we attempt to speed things up by forcing interrupts as if I/O were instantaneous, then we risk interrupting a user program at a time it was never anticipated because the programmer wrote timing-dependent programs. It might prove a valuable use of the simulator that it detect violations of the standard and proper use of enabling and disabling rules for interrupt programming.

The simulator can run in trace mode, providing a printout which is more than equivalent to the information one gets from hands-on use at the console. It would be a simple matter to include a breakpoint and snap-dump facility beyond that available with the real machine, because of the availability of 256,000 words of 1108 core.

Input to the simulator may be punched paper tape in RIM, BIN or ASCII format, or punched card in octal (for RIM and BIN) or BCD (for ASCII). Output may be produced on paper tape or punched cards. Use of magnetic tape merely involves changing a control card. The mechanisms for dealing with paper tape and cards is described in later sections.

Time tests which have been run indicate that the simulator performs a PDP8/L instruction in approximately 4.1 microseconds (in the no-trace mode). Recall that PDP8/L instruction times range from 1.6 to 4.25 microseconds, but that input and output are performed at 10 cps (on the Teletype).

Machine operator simulator

On small machines, console operations play a significant role in application programs, such as specifying program options, input data, load transfer addresses and so on. In the 1108 batch operation, it is necessary to simulate the role of the small machine operator. This is done using a "command language" which describes how the console switches and buttons are to be manipulated. Program loading and initiation begins by consulting the command language stream. Program halts cause further interpretation of the command language stream.

The commands which are needed to simulate the operator have a simple form:

```
$<command>                  <octal operand, if pertinent>
```

The commands are:

$SWR=	operand	(set console switch register = operand)
$LAD		(set the program counter from switches)
$DEP	operand	(store operand according to program counter)
$STRT		(push START button)
$SDS	operand	(combines SWR and DEP)

PDP-8 PAL III Assembler

This assembler accepts source statements prepared in accordance with Digital Equipment Corp. specifications. It is written in Fortran V for the 1108, and it accepts PAL III programs on punched cards or paper tape. Its output can be in BIN format, or it can be loaded directly into the simulator memory for an assembler-load-run test.

Timing tests indicate that 60 to 100 statements will assemble in one second of 1108 time.

It would be a reasonably straightforward effort to provide a macro-capability. In fact in an earlier class project, macro-preprocessors were written and tested, and these could have been used as a front-end to the PAL III assembler, since they are essentially machine independent.

ASCII and Fieldata code conversion

The PDP-8 users ASCII codes for alphanumeric information. The 1108 uses Fieldata codes. Conversion routines were written, subject to some reasonable compromises. For instance, when ASCII output is to be printed, characters which have no 1108 equivalent are mapped into the symbol delta.

Paper tape to Fastrand storage

The UWCC 1108 has no on-line paper tape input facility. It has off-line facilities for transferring paper tape to magnetic tape. Paper tapes in BIN or RIM or ASCII format are recorded on magnetic tape in binary mode. This magnetic tape is then read and stored either on Fastrand or punched cards, in a format 16(1X,04). Thus each frame of paper tap (8 bits) is mapped into a 12 bit representation.

Generation of punched paper tape

The UWCC 1108 has no on-line paper tape punch. Nor does the UWCC have any off-line paper tape punch facility. It is however possible to generate random binary information and to record it on magnetic tape or punched cards. These are then processed by a simple program which drives a paper tape punch attached to a Raytheon 440.

When demand processing becomes available on the 1108, one can use the Teletype paper tape reader and punch, but it is likely that the 1108 software will not accept or produce random binary. Thus it will still be necessary to use other means to accept or produce BIN tapes, for instance.

Tape and core dump interpreter

A routine has been written which will accept an arbitrary paper tape and produce three interpretations for each frame:

1) octal
2) mnemonic (as a PDP 8 op-code)
3) ASCII characters

The same routine can be applied to simulator core content, providing the same interpretations.

Absolute program de-assembler

This program is specifically designed to produce an assembler-like printout, using the simulator core content as input. It could easily be adapted to provide this output given the paper tape (in BIN format) which created the simulator core content.

Quick-load facility

Assembler output is absolute code punched on paper tape in a reasonably compact manner (BIN format). When such programs are to be tested in the simulator, they are loaded using the BIN loader. Since the BIN loader runs in the simulator, loading takes an appreciable amount of time. Having done it once, the quick-load facility can be used to copy the core image and save it. This core image can subsequently be placed back into simulator core, entirely by-passing the BIN loader.

BASIC-like language compiler

A language very similar to BASIC was specified and a compiler was implemented for it. The language was given the name PL/1.5 and its specifications are given in an appendix. In order to divide the effort, an intermediate language called UNCOL was specified (also in an appendix) and two distinct phases were provided:

1) PL/1.5 translation to UNCOL
2) UNCOL translation to PAL III source

The necessary support routines, such as software multiply and divide routines, were also written.

This aspect of the support system is probably of less interest to non-participants, since the PL/1.5 language is not standard, but it was an important part of the project in terms of understanding all aspects of software.

Conclusions

The project provided experience with and increased awareness of the importance of

1) clearly specified interfaces
2) correct and current documentation
3) media conversion problem
4) code conversion problems
5) mini-computer organization
6) software system organization

It demonstrated that in a period convering two calendar months (interrupted by the Christmas recess) a group of able students were able to construct a viable system.

It reinforced the instructor's belief that hands-on access to a real computer is necessary in a systems course. There is no substitute for this experience. Among the intangible but important benefits it contributes is the self-confidence it imparts, and the high level of motivation it inspires (i.e. real enthusiasm).

The software is now being used in the course CS436 which leads up to CS536, the course in which the software was developed.

What remains to be done

The author of this report is cooperating with other parties to make the software more effective in a production sense, and further documentation is in preparation.

It would be particularly interesting and challenging to either adapt the existing code, or develop new code, for support systems including a larger class of peripheral devices, and other mini-computers. No severe difficulty is anticipated in supporting systems with 8, 12, 16, 24 or 32 bit words.

Appendix 1

RIM, BIN and PDP 8 Bootstrap Loader.

```
channels
8 7 6 5 4 : 3 2 1 octal      interpretation
#         :       200        leader tape signal
#         :       200        leader tape signal
  # #     :   #   122  }
      # # : #     034  }     location 2234
    # # # : # # # 077  }
      # # : #     034  }     c(2234) is to be 7734
  #     # :   #   112  }
      # # : #     034  }     location 1234

#         :       200        trailer tape signal
```

RIM (read-in-mode) assumes for each 12 bit word to be loaded into core, that its 12 bit address will precede it on the tape. Channels 6 to 1 of pairs of frames are used to represent 12 bit words or addresses. Channels 8 and 7 have a control function as shown above.

BIN (binary tape format) is similar to RIM except that locations (i.e. addresses) are punched only when essential - when a break in the normal sequence occurs.

The following bootstrap loader is used to read in RIM tapes. In particular it is used to load the BIN loader, which is punched in RIM format.

The following instructions are loaded (by hand) starting at location 7756.

6032, 6031, 5357, 6036, 7106, 7006, 7510, 5357, 7006, 6031, 5367, 6034, 7420, 3776, 3376, 5356.

Exercises

1. De-assemble the above program, using PAL conventions.

2. Trace the execution of the above program, as applied to the same tape above.

3. Write a mnemonic program which can load RIM tapes.

4. Hand assemble the first ten instructions of the preceding program.

5. Discuss two techniques for increasing the redundancy in the load tape formats, and their suitability for the PDP 8.

Appendix 2

PAL III (Program Assembly Program) Highlights.

Sample source program:

```
*200          /*200 sets location counter=200 octal, need not start in col. 1.
MULT,         CLA CLL / "CLA CLL" is a combined
                        operate microinstruction
              TAD NUMBER / typical 1-address instr
              CLL RAL    / another com-op-micro
              TAD 207    / another reference to NUMBER (because of *200)
              SZA
              JMP .-2    / . refers to current location counter value
              DCA NUMBER
NUMBER,    7416  /  an octal constant (by default)
$                        / $=END of assembly code
```

Notes

1. / introduces comments, terminates line scan.

2. All labels are followed by a comma.

3. Identifiers as in Fortran, except that names I and Z are reserved for marking indirection (or lack of it), as in TAD I .37.

4. All constants are interpreted as octal, 12 bits, unless they follow a DECIMAL pseudo-op. Its effect can be cancelled by use of the OCTAL pseudo-op.

5. Expressions may have the form <u>sym op sym</u>, where <u>sym</u> is a constant or an identifier (which may be the name of an operation), and <u>op</u> may be a + or - or a space (for logical or).

6. Since this is a terminal oriented language, it has a free format, and allows multiple statements per line, using the semicolon as a separator.

7. Literals are not supported by PAL. MACRO, PAL's successor, uses the left paren to introduce a current-page literal, as in <u>TAD (22</u>, and it uses a left-square bracket for a page-0 literal, as in TAD [-123. ASCII (ANSII) single-character literals are indicated as in ("H for the letter H. The pseudo-op TEXT introduces text strings, with the first nonblank character following TEXT is expected to be the string terminator.

8. PAL uses the equal sign instead of the usual EQU. Thus TAD 207 in the above example could be written TAD ABC ; ABC=207.

Exercises

1. Study the procedure for using the PAL III assembler. How many passes does it require?

2. Design a PAL III assembler, in Fortran or Snobol, which runs on the 1108, but produces BIN formatted absolute programs for the PDP 8.

3. Discuss the extensions to 2 above which would provide for relocatable output, and specify an appropriate relocatable loader.

Appendix 3

PL/1.5   CS 536

v := variable name, as in Fortran

e := integer arithmetic expression, with +-*/

a := v, e, string

string := 'any char except quote'

n := line number

r := LT, EQ, LE, GT, GE, NE

b := array specification, as in A(10), B(3,4)

i := integer

Executable statements

[LET] v = e

$\left\{\begin{matrix}\text{READ}\\\text{INPUT}\end{matrix}\right\}$ v1, ..., vn

PRINT a1, ..., an

GOTO n

IF e1 r e2 THEN n

FOR v = e1 TO 32 STEP e3 ... NEXT v

GOSUB n ... RETURN

STOP

Declarations

DIM b1, ..., bn

DATA i1, ..., in

DEF $\left\{\begin{matrix}\text{v(v1)}\\\text{v(v1,v2)}\end{matrix}\right\}$ = e

Miscellaneous

REM, END

Intrinsics

ABS, SGN, TAB (TAB(i):= PLACE NEXT OUTPUT (CHAR IN COL I)

Sample program

```
 10 REM COMPUTED WEIGHTED AVERAGE
 20 DATA 10,15,-2,27,3,1
 30 INPUT N
 40 FOR I=1 TO N
 50 INPUT X
 60 READ WEIGHT
 70 LET SUM=SUM+X"WEIGHT
 80 IF SUM GT 2047 THEN 120
 90 NEXT I
100 PRINT 'WEIGHTED AVERAGE=' SUM/N
110 STOP
120 PRINT 'OVERFLOW'
125 RESTORE ("REWINDS" DATA)
130 GO TO 30
140 END
```

Appendix 4

PL/1.5 → UNCOL

These are guidelines, to be modified by mutual agreement

(PL/1.5) → 3 – address code

1) a = b + c = + a b c

2) a = b = ← a b

3) READ A,B(I),C(K,L) = R unit n –
```
0  0  0 A
1  0  I B
2  K  L C
```

4) IF e_1 r e_2 THEN n = $T_1 \leftarrow e_1$

$T_2 \leftarrow e_2$

IF T_1 r T_2

0 0 0 a(n)

5) PRINT $e_1, \ldots, e_n$ = $T_1 \leftarrow e_1$

$\wr$

$T_n \leftarrow e_n$

P unit count 0

0 0 0 T_1

$\sim\sim\sim$

0 0 0 T_n

6) GOTO n = GO a(n) – –

7) DIM $v_1(s_1)$, $v_2(s_2,s_3)$ = DIM v_1 s_1 0

DIM v_2 s_2 s_2

8) DATA $n_1, n_2, \ldots, n_n$ = DATA count 0 0

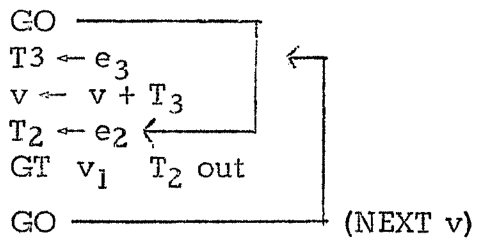
__________ n_1

$\wr$

__________ n_n

9) FOR v = e_1 TO e_2 STEP e_3

= $v \leftarrow e_1$



```
GO
T3 ← e3
v ← v + T3
T2 ← e2
GT v1  T2 out
GO               (NEXT v)
```

CODES FORMAT

CODE	a1	a2	a3
2	6	6	6

NOP	00
LET	01
READ	02
INPUT	03
PRINT	04
DATA	05
IF	06
GOTO	07
FOR	08
NEXT	09
GOSUB	10
RETURN	11
DIM	12
RESTORE	13
DEF	14
LT	30
LE	31
EQ	32
GE	33
GT	34
NE	35
Function call	50
label	60
REM	97
STOP	98
END	99