1058

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication. (See GEI 13.9-3)

Title- **C-Language Oriented Microprogram for the HP-21MX**

Date- **February 27, 1975**

TM- **75-1273-1**

Other Keywords-

Author	Location	Extension	Charging Case- **39199**
A. G. Fraser	**MH 2C520**	**3685**	Filing Case- **39199-11**
D. M. Ritchie	**MH 2C517**	**3770**	

ABSTRACT

The UNIX time-sharing system is presently implemented on the DEC PDP11/40 and PDP11/45 computers. Since the operating system itself and much of its software is written in the C language, an implementation of a UNIX-based system on another computer would be much easier, and much more efficient, if the new machine were particularly congenial to the requirements of C. Drawing on a study of the code generated for a sample of PDP11 C programs, this paper proposes an instruction-set architecture designed for implementation on the Hewlett-Packard 21MX micro-programmed processor. The design, which includes the crucial parts of the actual micro-program, is detailed enough to determine that the resulting machine would be a bit slower than that of the PDP11/40, and the size of compiled C programs would be smaller.

Pages Text 13	Other 0	Total 13	
No. Figures 0	No. Tables 3	No. Refs. 3	

BELL TELEPHONE LABORATORIES, INC.

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

AHO,A V
ANDERSON,M M
ARTHURS,EDWARD
BARTLETT,WADE S
BASEIL,RICHARD J
BAUGH,C R
BECKETT,J T
BERGLAND,G DAVID
BERRYMAN,R D
BIREN,MRS IRMA B
BOYCE,W M
BRILEY,B E
BROWNE,T E
BROWN,W STANLEY
CARROLL,J J
CHANG,HERBERT Y
CHANG,S-J
COHEN,HARVEY
COLSON,J S JR
COOK,ROBERT W
COOPER,DENNIS W
CRUME,LARRY L
DE LUGISH,BRUCE G
DENTON,R T
DRISCOLL,PATRICK J
FLEISCHER,HERBERT I
FOUGHT,B T
FRASER,A G
FREEMAN,K GLENN
GAY,FRANCIS A
GEYLING,F T
GITHENS,JOHN A
GOLDSTEIN,A JAY
GORDON,BRIAN G
GRAHAM,R L
GRAVEMAN,R F
GREENLAW,RICHARD L
HACHENBURG,V
HAGELBARGER,D W
HAHN,J R JR
HAIGHT,R C
HALLIN,THOMAS G
HAMILTON,PATRICIA
HAMMING,R W
♦HANNAY,N B
HARTWELL,WALTER T
IPPOLITI,O D
IRVINE,M M
IVIE,EVAN L
JANIK,JOHN JR
JENSEN,O C

COMPLETE MEMORANDUM TO

JOEL,A E JR
JOHNSON,STEPHEN C
KANE,JOHN M
KEESE,W M
KERNIGHAN,BRIAN W
KROPFL,WALTER J
LA MARCHE,ROBERT E
LEHRMAN,WILLIAM
LINNEWEH,LOUIS H
LIU,T K
LUDERER,GOTTFRIED W R
MARANZANO,JOSEPH F
MC EOWEN,JAMES R
MC GILL,ROBERT
MC ILROY,M DOUGLAS
MC PHERSON,ALAN F
♦MCDONALD,H S
MENON,P R
MICHAEL,ROBERT E
MILNE,D C
MOLLENAUER,J F
MORGAN,DENNIS J
MORGAN,S P
MORRIS,ANDREW A JR
MUGLIA,PATRICIA M
NASSON,NORMAN D
O SHEA,W T
OSSANNA,J F JR
PETERSON,RALPH W
PILLA,MICHAEL A
PINSON,ELLIOT N
♦PRIM,ROBERT C
RALEIGH,THOMAS M
RAY,F DUANE
REPSHER,WILLIAM G
ROBERTS,CHARLES S
RODRIGUEZ,ERNESTO J
SATZ,L R
SCHURTER,W H
SEDLMAYR,E R
SEYMOUR,ROBERT F
SHICHMAN,HAROLD
SHIVELY,RICHARD R
SHORTER,J W
SIPES,J D
SLANA,M F
SNARE,R C
SPANG,THOMAS C
SPIRES,R J
STEIGERWALT,R A
STONE,R C
STUART,J C JR
TABLOSKI,THEODORE F
TERRY,M E
THOMPSON,K
THOMSON,M-L
TJADEN,G S
TOWNSEND,RICHARD L JR
TOY,W
TUTELMAN,DAVID M
VERMA,SHIV P
WAGNER,BRUCE D
WALKER,MISS E A

COMPLETE MEMORANDUM TO

WARNER,JACK L
WARNER,TERRY L
WASSERMAN,MRS Z
WATSON,D S
WEBB,FRANCIS J
WENTWORTH,BENNING
WILD,J CHRISTIAN
WILLIAMS,R D
WO,YIU K
YAMIN,MRS E E
ZUK,P
ZYDNEY,HERBERT M
126 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ABRAHAN,STUART A
AHRENS,RAINER B
ALCALAY,DAVID
ALEXANDER,E J
ALLEN,JAMES R
ALLES,DAVID S
ALMQUIST,R P
AMORY,R W
AMRON,I
ANDERSON,L G
ANGNER,RONALD J
ARMSTRONG,DOUGLAS B
ARNOLD,THOMAS F
AXON,S L
BAECHLER,D O
BAKER,B S
BALDWIN,G L
BARNETT,M P
BARRESE,A L
BERING,D E
BERNSTEIN,LAWRENCE
BEYER,JEAN-DAVID
BILINSKI,D J
BILOMOS,RICHARD N
BLINN,JAMES C
BLUE,J L
BLY,JOSEPH A
BODEN,F J
BOHACHEVSKY,I O
BONANNI,LORENZO E
BOURNE,STEPHEN R
BOWEN,F W
BOWERS,J L
BRAUN,EDWIN J
BREEN,ROBERT S
BREITHAUPT,ALLAN R
BRENDZEL,HENRY T
BRONZO,J A
BUCHSBAUM,S J

COVER SHEET ONLY TO

BULFER,ANDREW F
BURROWS,F A
BUTLER,THOMAS T
BYRNE,EDWARD R
CANADAY,RUDD H
CARON,LIONEL
CASEY,JOSEPH P
CAVINESS,JOHN D
CHAMBERS,J M
CHAMBERS,MRS B C
CHAPPELL,S G
CHERRY,MS L L
CHODROW,MARK M
CHRIST,C W JR
CHUNG,FAN K R
CLAYTON,D P
COBEN,ROBERT M
COLE,LOUIS M
COLTON,JOHN R
COPELAND,JOHN A
COPP,DAVID H
CORNELL,R G
COULTER,J REGINALD
CRANE,B A
CUTLER,C CHAPIN
DAVIDSON,CHARLES L
DEUTSCH,DAVID N
DI PIETRO,RICHARD S
DINMICK,JAMES O
DOLOTTA,T A
DOMPIERRE,J A
DRAPER,DON R
DREIZLER,HOWARD K
DUBNOWSKI,JOHN J
DUFFY,FRANCIS P
EHRLICH,N
EITELBACH,DAVID L
ELLIOTT,R J
EMMOTT,J T
ENGRAM,R L
ESSERMAN,ALAN R
FABISCH,MICHAEL P
FARGO,GEORGE A
FELDMAN,STUART I
FELS,ALLEN M
FETTE,CHARLES J
FISCHER,H B
FLEMING,T W
FONG,KENNETH T
FORT,JAMES W
FOWLER,C F
FOX,R T
FRANK,MISS A J
FRANK,RUDOLPH J
FRESNY,S L
FULTON,ALAN W
GATES,G W
GEER,EUGENE W JR
GIBB,KENNETH R
GILLETTE,DEAN
GILMER,J L
GIMPEL,JAMES F
GLASSER,ALAN L

COVER SHEET ONLY TO

GLUCK,F
GNANADESIKAN,R
GOETZ,FRANK M
GOLABEK,MISS R
GORMAN,JAMES E
GREENBAUM,H J
GREENE,MRS DELFA A
GREENWOOD,T SPENCER
GUERRIERO,JOSEPH R
GUMMEL,HERMANN K
HAFER,E H
HALL,ANDREW D JR
HALL,W G
HANDSPICKER,BRIAN P
HARKNESS,MRS CAROL J
HARUTA,K
HAWKINS,DONALD T
HAYNIE,G D
HEATH,SIDNEY F III
HERBST,ROBERT T
HERMAN,KENNETH M
HEROLD,JOHN W
HESS,MILTON S
HILDRETH,RONALD D
HOALST,BLAINE C
HOEHN,MISS MARIE J
HONIG,W L
HOPPERT,D J
HORLACHER,ROBERT L
HORNBACH,THOMAS S
HOYT,WILLIAM P
HUDSON,E T
HUMCKE,D J
HUNNICUTT,CHARLES F
HYMAN,B
JACKOWSKI,D J
JACKSON,JAMES H
JACOBS,H S
JAMES,DENNIS B
JENNESS,R V
JENSEN,PAUL D
JESSOP,WARREN H
JUDICE,CHARLES N
KAISER,J F
KANE,J RICHARD
KAPLAN,N M
KARAFIN,B J
KAUFELD,J C JR
KAYEL,R G
KELLY,L J
KEMP,C W
KENNEDY,ROBERT A
KERTZ,DENIS R
KILLMER,JOHN C JR
KLAPPROTH,F F
KLEBER,J J
KLEIN,MISS R L
KNOLLMAN,DIETER J H
KNOLL,RONALD L
KNOWLTON,KENNETH
KNUDSEN,DONALD B
KNUDSON,M J
KOHN,MELINDA C

...
354 TOTAL

♦ NAMED BY AUTHOR

> CITED AS REFERENCE SOURCE

MERCURY DISTRIBUTION.....

COMPLETE MEMO TO:
127-SUP
COHALD = COMPUTING/HARDWARE/LOGIC DESIGN, MACHINE ORGANIZATION

COVER SHEET TO:
12-DIR 13-DIR 127
COHA = COMPUTING/HARDWARE/SURVEY PAPERS ONLY

FRASER,A G
MB 2C520

TM-75-1273-1
TOTAL PAGES 13

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

Subject: **C-Language Oriented Microprogram for the HP-21MX**
Case- 39199 -- File- 39199-11

date: **February 27, 1975**

from: **A. G. Fraser**
D. M. Ritchie

TM: **75-1273-1**

MEMORANDUM FOR FILE

Introduction.

It is currently both practical and economically attractive to construct a general-purpose computer by micro-programming a simpler special purpose machine. The time may soon come when a corporation such as Bell Laboratories can standardize upon a machine/language design and yet have the option of buying the hardware from several sources. The limited extent to which that might now be possible is illustrated by an investigation into using Hewlett-Packard hardware in place of the Digital Equipment machine on which UNIX software normally runs.

The UNIX operating system[1] currently runs on PDP11/40 and PDP11/45 computers. Most of the operating system and applications software are written in the language C [2]. The language encourages but does not attempt to enforce the portability of programs between machines. Recent experience in moving fairly substantial UNIX programs, written in C, to the IBM 370 and the HIS 6070 suggests that most of the effort of transporting programs lies not in the code itself but in the interface to the operating system, in spite of the fact that the hardware environments of the three machines involved are quite diverse. Nevertheless, for efficiency reasons, the following architectural features should be provided.

a) UNIX strategy relies heavily upon multi-programming and so address mapping hardware must be available to allow several processes to occupy memory simultaneously.

b) Byte variables, pointers to bytes and arithmetic on byte pointers are heavily used and should be provided for by hardware.

c) C programs make heavy use of procedure and function calls so that reasonable efficiency can be obtained only by using a call sequence with low overhead.

Aside from user-mode software, of course, much of the operating system itself would clearly have to be re-written for any new machine in order to accommodate its peripherals and process management hardware.

The work of transferring UNIX software to a Hewlett-Packard 21MX computer[3] would be minimized if the 21MX could be micro-programmed to emulate the PDP11. But the 21MX was not designed as a base for emulating other machines and it does a very inefficient job of emulating a PDP11. An alternative, which we describe in this memorandum, is to define a new machine whose architecture is similar to that of the PDP11 but whose instruction formats can be handled with minimum overhead by 21MX microprogram.

The need to define a new 16-bit machine code to handle C object programs provides an opportunity to improve upon the PDP11 machine code for this purpose. We have chosen to design a new machine, called the 'C machine,' so as to minimize the size of compiled C programs. To that end we obtained some statistics from programs compiled for the PDP11.

Statistics of C Programs

The PDP11 provides eight addressing modes.

R	Register R
(R)	Memory pointed to by R
(R++)	As (R) but increment R after addressing
((R++))	Memory pointed to by (R++)
(--R)	As (R) but decrement R before addressing
((--R))	Memory pointed to by (--R)
X(R)	Memory address X indexed by R
(X(R))	Memory pointed to by X(R)

The machine provides eight registers which are used by C programs as follows.

R0,R1	Temporary results
R2-R4	Local variables
R5	Stack frame pointer, FP
R6	Stack pointer, SP
R7	Program counter, PC

Some address forms have special uses. They are as follows.

X(FP)	Local variable or function argument
(PC++)	Literal constant
((PC++))	Literal pointer
(--SP)	Push item onto the stack
(SP++)	Pop item from the stack
X(PC)	Static variable

Table 1 contains some statistics obtained by scanning 21516 words of compiled C programs. In these programs, all branches, data-generating, pseudo-operation, and other non-data instructions were cast out. The special cases referred to above occur with the following frequencies.

X(FP)	18	per 100 instructions
(PC++)	31	per 100 instructions
((PC++))	1.8	per 100 instructions
(--SP)	9.4	per 100 instructions
(SP++)	7.6	per 100 instructions
X(PC)	15	per 100 instructions

If one excludes these cases one finds that most other address forms fit into one of the three patterns: R, (R), and X(R). Those that do not fit these patterns occur with the following frequencies.

(R++)	1.4	per 100 instructions
((R++))	0.04	per 100 instructions
(--R)	0.17	per 100 instructions
((--R))	0	per 100 instructions
(X(R))	2.0	per 100 instructions

The 21516 compiled words of program contained 11869 instructions and 9647 words of literals. Table 2 shows how the values of these literals were distributed. It will be seen that 4110 literals were in the range −7 to +7 so that some economy could be obtained by allowing small integer literals to appear in the instruction proper rather than in separately compiled words.

The C Machine

The 16-bit C machine is based upon the above statistics. Its central processor has 8 registers that are used as follows.

R0-R4 General-purpose registers; R0,R1 are used for all double-length and floating point operations.

R5 Stack frame pointer, FP; set equal to the stack pointer after a procedure call.

R6 Stack pointer, SP; points to the 'top' of the stack; as in the PDP11 a stack push decreases SP, and SP is used to save registers and control information during a subroutine call.

R7 This register reads as 0 and cannot be written into. In hardware it is the program counter, but it is not directly addressible.

A general address can take four different forms. It is a 4-bit field subdivided into a 1-bit indirection flag, I, and a 3-bit register address, R. The addressing modes are as follows.

I=1, R<7	register R
I=1, R=7	literal constant X
I=0, R<7	memory X(R)
I=0, R=7	memory X

Depending upon the instruction format the value X can either be in a 4-bit field of the instruction or can be in a word following the instruction.

The 21MX does not have a byte organized memory so that byte addressing in the PDP11 way would be quite expensive (in execution time) to implement. Byte addresses in the C machine have a word address in their least significant 15 bits and the most significant bit indicates which of two bytes in the word is intended. Thus the C machine address is equal to the PDP11 address after rotating the PDP11 address 1 bit to the right. To make this acceptable the C machine has separate instructions for doing pointer arithmetic. Although there are a few programs which make use of the particular structure of the PDP11 address word, this change in representation is not expected to affect program compatibility to any noticable degree. It also leads to more efficient programs when integer arrays are used.

There are five instruction formats. They are listed in detail in appendix A together with the operation codes available with each format.

a) Double operand instructions, with four fields:

A 4-bit function
A 4-bit general destination address
A 4-bit general source address
A 4-bit literal used to augment the source address

If the literal equals 15 then the value X used to compute the source address is obtained from (PC++) otherwise the value is the literal X−7.

b) Register/operand instructions, with four fields:

A 5-bit function
A 3-bit destination register address
A 4-bit general source address
A 4-bit literal used to augment the source address

c) Single operand instructions, with three fields:

An 8-bit function
A 4-bit general address
A 4-bit literal used to augment the general address

d) Jump instructions, with three fields:

A 7-bit function
A 2-bit type code
A 7-bit jump address

Depending upon the type code, this is either a 7-bit literal J, or is a 3-bit register R, and a 4-bit literal XX.

The type code can take three values:

1 jump to (PC)+J

2 jump to X(R), where X is (PC++) if XX=15 otherwise X is XX−7.

3 jump to (PC)−J

e) Miscellaneous instructions, which have only a function code in the least significant eight bits of the 16-bit word.

Implementation on the 21MX

Key samples of the microcode required to implement the C machine on the 21MX have been written; they are given in Appendix B. The general strategy is to use the switch jump micro-instructions to decode instruction fields. This leads to a large micro-program but a fast one. (It is not expected that the cost of micro-store will be significant.) It also means that we rely upon Hewlett-Packard's offer to use plug-in sockets for their jump table read-only memories. We would then replace the HP versions with our own. The main instruction interpretation loop has been written together with sample entries for the various switch jump tables. No code for interrupts is included but it is thought that that would be quite straight forward and not markedly different from the HP version. The same applies to the code required to handle the front panel.

The strategy used for Input/Output operations is to provide one C machine instruction whose operand is a legitimate HP I/O instruction. The micro-program simply transfers that operand to the instruction register and uses the HP micro-program sequence.

Memory addressing and protection have been cobbled into the 21MX in various places (presumably in an attempt to speed up the 2100 machine emulation). In particular there is hardware to handle the following.

a) addresses 0 and 1
These are used by HP to refer to the central processor registers A and B. If the micro-instructions that use the BAF and AAF flags are avoided, addresses 0 and 1 can be used normally. But they cannot be write-protected.

b) base page addressing
There are two 'fences:' one in the protection unit and one in the memory mapping unit. If they are both set to zero and mapping 'above the fence' is selected all addresses can be mapped.

c) illegal instructions
The memory address protection unit also checks for certain illegal instructions. Hewlett-Packard provide a strapping option on the protection board that can turn this function off. When that is done the micro-code must check for illegal use of I/O and HALT instructions.

It is apparent that Hewlett-Packard intend to encourage micro-programming by their 21MX users. Apart from the documentation, which is reasonably complete, they provide a writable micro-program memory module that can be loaded through the 21MX peripheral bus. Thus the most practical way of bringing up the micro-program for the C machine is to put two 21MX side-by-side. One 21MX would load the micro-program for the other. Eventually a HP provided read-only memory 'zapper' can be used to write the proper micro-program memory.

To further assist in the process of developing the micro-code HP have indicated that we could obtain one of their 'suitcase test units' which are really plug-in front panels for the micro-machine. There is a connector on the back of the CPU that allows one to operate the micro-machine directly. For example, from that connector one can load micro-code to be obeyed and one can alter the speed of the clock.

Conclusion

We have successfully designed a machine that can be used as a vehicle for transferring UNIX software to the Hewlett-Packard 21MX hardware. The software effort required to effect the transfer of C programs is quite minimal and that required for PDP11 assembly code would probably not be a major undertaking. Our experience in writing code for the 21MX indicates that that is not a difficult problem although it does require more than ordinary care. (Other related experiences suggest that prior thought pays unusually handsome dividends, by reducing debugging time, for this type of work.)

For word operations, the speed of the C machine would be somewhat less than a PDP11/40 but its speed for byte operations would be less than half that of the PDP11/40. Compiled C programs would be about 20% shorter on the C machine. A comparison of some C machine and PDP11/40 times appears in Table 3.

During the course of this study it has become evident how sensitive the performance of an emulated machine is to the way in which the micro-machine handles sub-fields of the instruction register. It is apparent that a versatile micro-machine must be able to extract and suitably shift arbitrary fields of an instruction. It must then be possible to deliver those fields to a central machine register and also to do a switch jump on the extracted value. These are expensive requirements and may mean that a general-purpose micro-machine is also an expensive machine. However, the comparative success of our study indicates that standardization at a language level only slightly above machine code may well leave one with sufficient latitude to plan on alternative hardware support for even quite low level software.

A. G. Fraser

MH-1273-$\frac{AGF}{DMR}$-UNIX

D. M. Ritchie

Att.
References
Tables 1-3
Appendices A, B

References

[1] Ritchie, D. M. and Thompson, K. The UNIX Time-sharing System. C. ACM 17, 7, (July, 1974), pp 365-375.

[2] Ritchie, D.M. C Reference Manual. TM-74-1273-1.

[3] "Microprogramming 21MX Computers: Operating and Reference Manual." Hewlett-Packard (#02108-90008), 1974.

TABLE 1
Frequencies of Addressing Modes for C Programs

The following table gives the numbers of instructions, in a sample of 11869 instructions, that used the indicated addressing mode and indicated register. The sample consisted exclusively of data-manipulating operations (moves, adds, and so forth); branches, calls, and the like were excluded.

mode	R0	R1	R2	R3	R4	R5	SP	PC
R	3315	509	587	878	1414	54	54	0
(R)	88	12	76	135	271	0	1739	0
(R++)	0	30	35	52	56	0	907	3794
((R++))	0	0	0	0	0	0	5	219
(--R)	0	0	6	5	10	0	1114	0
((--R))	0	0	0	0	0	0	0	0
X(R)	355	52	160	155	621	2224	0	1827
(X(R))	3	0	3	0	12	141	1	80

TABLE 2
Distribution of Literal Values in C programs

In the following table *P(n)* is the number of occurrences of the literal value *n* in the object program. *M(n)* is the number of occurrences of the value *−n*. *S(n)* is the number of occurrences of literals in the range *−n* to *n* inclusive. *F(n)* is the fraction of the total number of literals that are in that range. In compiling this table, the literals in index words X(R) were divided by 2 to compensate for word rather than byte addressing.

n	*P(n)*	*M(n)*	*S(n)*	*F(n)*
0	86		86	1.3%
1	555	101	742	11.3%
2	1025	40	1807	27.5%
3	572	20	2399	36.4%
4	266	417	3082	46.8%
5	142	272	3496	53.1%
6	198	187	3881	59.0%
7	122	107	4110	62.4%
8	162	81	4353	66.1%
9	72	37	4462	67.8%
10	141	41	4644	70.6%
11	16	23	4683	71.1%
12	52	25	4760	72.3%
13	57	23	4840	73.5%
14	27	20	4887	74.2%
15	16	7	4910	74.6%
>15	1108	564	6582	100.0%

TABLE 3
Comparison of PDP11/40 and C Machine Times

In the following table *X* denotes a 16-bit literal while *x* denotes a literal in the range −7 to +7. A '$' prefix indicates that the operand is a literal constant rather than the content of a memory location. 'Mov' is a full-word data move; 'movb' is a one-byte transfer. The pairs of times for the byte moves are for the right- and left-byte cases respectively.

operation	operation times (microseconds) 11/40	21MX
mov R to R	0.90	1.95
mov $x to R	2.45	2.60
mov $X to R	2.45	3.25
mov x(R) to R	3.22	3.90
mov X(R) to R	3.22	4.55
mov R to X(R)	3.29	3.90
mov x(R) to X(R)	5.39	5.53
mov X(R) to X(R)	5.39	6.18
movb R to R	1.80, 2.14	3.58, 4.23
movb x(R) to R	3.56, 3.90	5.53, 6.18
movb X(R) to R	3.56, 3.90	6.18, 6.83
movb R to x(R)	3.43, 3.77	7.15, 7.48
movb R to X(R)	3.43, 3.77	7.80, 8.13
add R to R	1.14	1.95
add x(R) to R	3.51	3.90
add X(R) to R	3.51	4.55
add R to X(R)	4.10	4.88
add x(R) to X(R)	5.86	6.50
add X(R) to X(R)	5.86	7.15
function call	22.38	6.83
function return	18.76	6.50

APPENDIX A
Description of C Machine Language

Conventions

Eight of the HP registers are available to the user and are referred to herein as R0 through R7. They are used as follows.

R7 The program counter PC. R7 reads as zero when used in an instruction address field. R7 cannot be used as a destination.

R6 The stack pointer SP. SP points to the last entry put on the stack. It decreases as items are pushed on the stack.

R5 The stack frame pointer FP. It is set equal to the stack pointer immediately after a procedure call.

R4-R2 General purpose registers.

R01 General-purpose register used implicitly in double-length instructions is combination of R0 and R1.

Parentheses denote indexing or indirection. Thus Y(R) is Y indexed by R.

'++' denotes increment and is used to describe a side-effect after computing an address. Thus (PC++) refers to the word pointed to by the program counter and the program counter is then incremented.

'--' denotes decrement and takes place before the address is computed. Thus (--SP) is the word pointed to by the stack pointer after the stack pointer has been decremented.

The field structure of addresses and instructions is described by giving the name of each field followed by its size in brackets, for example R[3]. The field 0[9] is nine zeros.

Architecture

Address mapping and protection

The memory mapping unit can be made to give two full spaces, one for the user and the other for the system. The only special case is that locations 0 and 1 cannot be write protected.

Byte addressing

Two 8-bit bytes are stored in one 16-bit word. The most significant byte is in the least significant half of the word. A pointer is the address of a byte in main memory. It has the word address in its rightmost 15 bits; the top bit is set to address the leftmost byte in a word. Word operations ignore the leftmost bit of an address. When a byte is transferred to a register it occupies the right half of the register and the left half is cleared. It is not possible to address the leftmost byte of a register directly.

Input/Output

The HP Input/Output operations are preserved. The single C machine I/O instruction has an HP instruction as its operand. Since HP offers no byte-organized device that uses DMA there is no problem arising out of the different ways in which the HP 2100 machine and the C machine locate bytes in a word.

Addresses

address format (a)

field R[3] is a register address

address format (b)

fields: I[1],R[3]

	r-value	*l-value*
I=1	(R)	R
I=0	((PC++)+(R))	(PC++)+(R)

address format (c)

fields: I[1],R[3],X[4]
let Y = (if X = 15 then (PC++) else X−7)

	r-value	*l-value*
I=1 & R<7	(R)	R
I=1 & R=7	Y	null
I=0 & R<7	(Y+(R))	Y+(R)
I=0 & R=7	(Y)	Y

address format (d)

fields: I[2],R[3],X[4] or I[2],J[7]
let Y = (if X = 15 then (PC++) else X−7)

	jump addr
I=1	(pc)+J
I=2	Y(R)
I=3	(pc)−J

Instructions

double operand instructions

fields: F[4],D[4],S[8]
D is format (b) address; S is format (c) address
$4 \leqslant F < 12$.

D = S, S = D	word assignment
D = D+S, D = D−S	integer arithmetic
D = D | S, D = D&S	logical operations
test (S&D), test (S−D)	word comparison

register-operand instructions

fields: F[5], D[3], S[8]
D is format (a), S is format (c), $24 \leqslant F < 32$

D = D+S, D = D−S	pointer arithmetic
test (S−D)	pointer comparison
D = S	load byte
S = D	store byte from ls of D
test (S−D)	byte comparison
shift D by S places	logical shift
rotate D by S places	

single operand instructions

fields: F[8], S[8]
S is format (c), $1 \leqslant F < 32$

(--SP)=S, S=(SP++), S=0	word transfer
(--SP)=S, S=(SP++), S=0	S is byte
(--SP)=S, S=(SP++)	word S in user space
(--SP)=S, S=(SP++)	byte S in user space
S = S+1, S = S−1, S = −S	integer arithmetic
S = S+1, S = S−1	pointer arithmetic
test S	compare word with zero
test S	compare byte with zero
rotate S by 8 bits	
R01 = R0 * S	multiply
R0 = R01 / S	with remainder in R1
R01 = S, S = R01	double word assignment
R01 = R01+S, R01 = R01−S,	floating point
R01 = R01*S, R01 = R01/S	floating point
test (R01−S)	floating point
shift R01 by S places	arithmetical shift
shift R01 by S places	logical shift
rotate R01 by S places	
execute I/O instruction S	

jump instructions

fields: F[7], J[9]
J is format (d), $16 \leqslant F < 32$.
Jump conditions are determined by previous arithmetic, test or logical operation.

jump on zero, on non-zero
jump on negative, on non-negative
jump if carry set, if not set
jump if overflow set, if not set
jump if less than zero (neg. or ovr. not both),
jump if not less than zero
jump if less than or equal to zero
jump if greater than zero
jump unconditionally
function call; saves PC, FP, R4, R3, R2
interrupt call J (I=1 only); saves all regs and flags
system call J (I=3 only)

miscellaneous instructions

fields: 0[9], F[7]; $0 \leqslant F < 128$

return from function
return from interrupt
return from system
set priority level
halt
reset peripheral devices

APPENDIX B
Outline of Micro-Code

Main loop

```
        read   ftch  inc   pnm p     /fetch next instr
        jmp    stfl  1f              /jump to avoid location 4.
                                     /interrupts transfer to 4.
1:
               ion   pass  ir   t
        jsb    j74   d.Rs            /switch on Is, copy Rs into L
                                     /switch on F if Is=1
        jsb    j30   d.Xs            /decode Xs into s1
               clfl  add   s1   s1    /l-value to s1
        read   ftch  pass  m    s1    /get operand
        nop
               jtab  pass  l    t     /S to l, switch on F
```

jump tables - address decoding

```
d.Rs:                                /switch on IRs
               jtab  pass  l    Rs    /entry for Is=1 & Rs<7
        jsb    j30   d.Xs            /entry for Is=1 & Rs=7
               jtab  pass  l    s1
               rtn   pass  l    Rs    /entry for Is=0 & Rs<7
               rtn   pass  l    s1    /entry for Is=0 & Rs=7

d.Xs:                                /switch on Xs
        imm    rtn   cmlo  s1        '-(X-7)' /entry for all X < 15
        read   ftch  inc   pnm p     /entry for X=15
               nop
               rtn   pass  s1    t   /get X from next instrn word
```

main jump table - JTAB

```
d.f:                                 /switch jump table for F
                                     /S is in l, addr of S in s1
                                     /entry for add to reg
        enve   rnt   add   Rd   Rd
                                     /entry for add to mem
        jmp    uncd  addM
                                     /entry for load byte
        jmp    uncd  load
                                     /entry for store byte
        jmp    uncd  stor
                                     /entry for function call
        jmp    uncd  fun0
        jmp    uncd  fun1
                                     /function return
        jmp    uncd  frtn
```

add to memory

```
addM:                                /add S to D
        read   ftch  inc   pnm p     /read Xd
```

```
                pasl  s3    l     /save S
                pass  l     t
                add   s2    Rd    /addr of D to s2
        read    mpckpass    m     s2    /fetch D and check for writing
                ftch  pass  l     s3
        enve    add   s3    t     /result to s3
        write   rtn   pass  t     s3    /store away
```

byte operations

```
load:                               /load D from memory
        jmp   cndx flag  2f         /operand in register
              pass       s1
        jmp   cndx al15  1f         /j if in ms 8 bits
2:
        imm        cmlo  s1    0
              rtn  and   Rd    s1
1:
        imm        cmhi  s1    0
              l4   and   s1    s1
              l4   pass  Rd    s1
              rtn
stor:                               /store D into memory
              mpckpass   m     s1
        jmp   cndx flag  2f
        jmp   cndx al15  1f         /j if in ms 8 bits
2:
        imm        cmhi  s1    0
                   and   s3    s1
                   pass  l     s1
                   op5   s1    Rd    /and with ~l
                   pass  l     s1
        jmp   uncd j74   2f
1:
        imm        cmlo  s1    0
                   and   s3    s1
                   pass  l     s1
              l4   and   s1    Rd
              l4   pass  s1    s1
                   pass  l     s1
        jmp   uncd j74   2f
2:                                  /switch jump table on IRs
              rtn  xor   Rs    s3    /entry for Is=1 & Rs<7
              rtn                    /entry for Is=1 & Rs=7
        write rtn  xor   t     s3    /entry for Is=0
```

function call and return

```
fun0:                               /function call (I=1)
        jsb   uncd fun
        jmp   uncd jmp0
fun1:                               /function call (I=3)
        jsb   uncd fun
        jmp   uncd jmp1
fun:
```

```
                pass  s2    PC   /save PC
          mpckdec  PC    SP
          mpckdec  pnm   PC
     write     pass  t     FP   /stack frame pointer
          mpckdec  pnm   PC
     write     pass  t     s2   /prog counter
          mpckdec  pnm   PC
     write     pass  t     R4   /reg 4
          mpckdec  pnm   PC
     write     pass  t     R3   /reg 3
          mpckdec  pnm   PC
     write     pass  t     R2   /reg 2
               inc   SP    PC
          rtn  pass  FP    SP

frtn:                             /return from function
     read  ftch inc   pnm   PC
               nop
               pass  r2    t
     read  ftch inc   pnm   PC
               nop
               pass  r3    t
     read  ftch inc   pnm   PC
               nop
               pass  R4    t
     read  ftch inc   pnm   PC
               nop
               pass  FP    t
     read  ftch inc   pnm   PC
               pass  SP    PC
          rtn  pass  PC    t
```

all jumps end here

```
jmp0:                             /effect a jump with I=1
     imm       low   1     0100
               op5   s1    adr   /pick off J
               pass  1     s1
          rtn  add   PC    PC

jmp1:                             /effect jump with I=2 or 3
     jmp  uncd j74   j.I          /decode I

j.I:                              /switch to decode I
          rtn  pass  PC    s1    /entry for I=2
     jmp  uncd 1f                 /entry for I=3
1:
     imm       low   1     0100
               op5   s1    adr   /pick off J
               pass  1     s1
          rtn  sub   PC    PC
```