digital INTEROFFICE MEMORANDUM

DATE: March 4, 1969

SUBJECT: PDP-11 Architecture

TO: Nick Mazzarese

FROM: Rick Merrill

cc: Distribution List

I. PRO

The instruction set of the new PDP-11 is the most core conservative in the 8/16 bit field of small computers. And it is easy to learn, to program, and to debug.

For purchasers of quantities of computers there is a savings in the cost of memory. For those concerned about programming costs and lead times, there are substantial cost reductions. For anyone who realizes that it costs five to ten times as much to program a computer than to buy it, here is an opportunity to stretch your programming expertise. The PDP-11 offers the most cost-effective solution to machine control and data processing problems.

The PDP-11 architecture is unsurpassed in I/O efficiency and completeness. All memory reference instructions are also general purpose I/O instructions.

II. Codes

Attached is a PDP-11 Assembler Mnemonic Matrix. This form of architecture description is a common denominator for assembler creation, hardware specifications, and user programming. For use in on-line debugging the octal values must be added.

I recommend that a pocket-size Instruction List like the 8's be created and made available post-haste to one and all. This will serve to unify our own internal thinking and allow customers to evaluate our claims themselves.

The assembler Pseudo-Ops should also be listed along with the usual ASCII character set.

III. PDP-11 Assembler

A name needs to be chosen (perhaps with the help of advertising) and a design review scheduled as soon as feasible. Three letter symbolic mnemonics are not adequate for painless programming. Most assembler mnemonics are four! We should try for six letters packed. Commas should be used as in the coding examples. This facilitates computation of program bytes actually used and makes the assembler syntax decoder much simpler to build.

IV. FOCAL - 11

Enclosed are three coding examples from the central operations of the FOCAL interpreter. These are atypical examples, but the 11 code is ?~~10%~~ more bit efficient. Overall the 11 could save 600 bytes over the 8. This estimate is based on the following rough instruction distribution in FOCAL-8: (octal)

AND -	500	JMP -	600
TAD -	1000	IOT -	20
DCA -	200	OPR1 -	300
JMS -	500	OPR2 -	400

I recommend that FOCAL be translated at once for the PDP-11. This will produce several other programs and subroutines as by-products: a PDP-8 like editor, a well-debugged floating point package, and a real-time character-oriented I/O package.

V. CON

We need a "SWAP halves of the AC" instruction. "TTA" can be used for this purpose.

We need a "HALT" instruction (1byte). "TFA" can be used for this one.

The "Add to Register" instructions need to set a testable overflow flag without affecting the contents of the AC. "INC" should have a similar test.

We need some possibility of multi-level indirect.

There needs to be a way of reading the switch registers!

The index registers are nearly real ones but they still cannot truly be used to index, only to point. A true index register can transform any subroutine, like a double precision add, into a table add. i.e. add elements of two tables to produce elements of a third table. True index registers also facilitate matrix manipulations.

A solution to the compromise between powerful index registers and bit efficiency would be an addressing mode bit in the I/O area to determine whether the Q of an indexed instruction shall be one or two bytes. Thus

LDW X1, A (two bytes requires X1 (16 bits) to point to a table and A (8 bits) to be the increment. If A uses 16 bits then X1 could be the increment and all table references would be indexed. X1 would then also be used as the count to detect the end of a table operation. For table operations, this is the optimum bit-efficient approach.

```
2 byte way               (24 bit add)
LDW, (-100)
STW, COUNT

LDW, (AA) ; AA is top of list
TTX1
LDW, (BB)
TTX2 LDW, (BC)

DOUBL: CCC               TTX3
       .
       .
       .
```

```
        3 byte way
        LDW, (-100); table length
        TTX1
DOUBL:  CCC
        LDW X4,,A;faster table
        ADW X4,,B;B is end of list
        STW X4,,C
        CLA,RAL

        ADB X4,,A+3
        ADB X4,,B+3
        STB X4,,C+3
        ATX1,3
        JCFL,DOUBL
```

28 bytes

```
LDW  I       X1 , Ø
ADW  I       X2 , Ø
STW  I       X3 , Ø
CLA,RAL
ADB  I       X1 , 2
ADB  I       X2 , 2
STB  I       X3,  2
ATX1 , 3
ATX2 , 3
ATX3 , 3
INC  , COUNT
JCFL , DOUBL
```

38 bytes /28 bytes

Notice also that the present method uses three index registers while the other uses only one. I recommend that XR#4 be called XR and when it is used the address field is two bytes. This solution lets us advertise five index registers, bit efficiency, true indexing, and stack control!

The new design negates all arguments for hexadecimal representation. I recommend that octal be used.

TRAP must effectively execute a JSR I, [3] so that there may be several simultaneous breakpoints. This for both logical complexity and so that both halves of a word may be trapped.

PDP-11 Assembler Mnemonic Matrix

MEM REF

LD	-B	I	X1	,Q	relative or page zero
ST	-W	△	X2	,(Q)	immediate
AD			X3	,"Q"	immediate literal ascii
CP			XR	,(#)	immediate number
AND			XS		
INC			△		
JMP					
JSR					

OPR

	NOP	CLA
*	TRAP	CML
	IAC	CCC
	CMA	RAR
	NEG	RAL

REGISTER

AT	-X1	,$Q
TT	-X2	
TF	-X3	
PU	-X4 XR	
PO	-XS	
	-PC	
	-CC	

(index stack)
(PC and jump)
(Condition codes

MISC

*	SWAP
*	CLL
	ATA,Q
	PUA
	POA
	XTR,Q

Jump on Conditions True or False

JCT	-Z	-S	-L	,Q relative or zero
JCF	△	△	△	

Condition Codes

I/O flag, Z,S,L!priorities

All mnemonics are recognizable by only first three letters plus register designator or conditon codes.

Machine Organization

Eight hardware registers:

Accumulator	AC
Program Counter	PC
Priority and condition codes	CC
Index Registers	X1
	X2
	X3
Stack Index	XS
True Index	XR

Addressing modes (automatic)

	bytes	not deferred	deferred
Immediate	3	EFA=next location	EFA=(next location)
Relative to P	2	EFA=(PC)+OFFSET	EFA=((PC)+OFFSET)
Page Zero	2	EFA=ØØOFFSET	EFA=(ØØOFFSET)
Indexed X1,X2,X3,XS	2	EFA=(IR)+OFFSET	EFA=((IR)+OFFSET)
Indexed XR	3	EFA=(XR)+POINTER	EFA=((XR)+POINTER)

Notes:

OFFSET is an 8 bit quantity (7 bits magnitude, 1 bit sign) and is the second byte of the two byte instruction.

For Page Zero references, OFFSET is considered an 8 bit quantity which forms the least significant byte of an address of which the most significant byte is all zeroes. Page zero is thus 256 bytes long.

IR refers to the index register which is desired to be used in the address pomputation. It may be X1, X2, X3, or XS.

POINTER is a 16 bit quantify for absolute address of tables indexed by XR.

The internal registers of the processor may be explicitly addressed by external devices, but may not be explicitly addressed in a program execution of a memory reference instruction. This is to simplify hardware.

Transfer to/from register (using accumulator)   (1 byte)

Typical Execution:

TTX1 = A $\longrightarrow$ X1

TFX1 = X1 $\longrightarrow$ A

Push/Pop Group   (1 byte)

Typical Execution:

PUXL = (X1) $\longrightarrow$ ((S))   (S) + 2 $\longrightarrow$ S

POX1 = (S)-2 $\longrightarrow$ S ,   ((S)) $\longrightarrow$ (X1)

Condition Jump   (2 byte)   (Second byte is signed byte which is added to P if test is true)

JCT	Z,N,L	(logical or)	may be micro-programmed
JCF	Z,N,L	(logical and)	may be micro-programmed
JFS	I/O Flag set		
JFR	I/O Flag reset		

Interrupt Process

PC $\longrightarrow$ (S)
C $\longrightarrow$ (S+2)
80 $\longrightarrow$ (S+3)
(S)+4 $\longrightarrow$ S

Double precision add

```
CCC                                 ;clear link
LDW      , A1                       ;add var
ADW      , A2                       ;use X3,X Y as AC
STW      , A1
CLA,RAL                             ;bug in carry bit
ADW      , B1
ADW      , B
STW      , B1                       ;may be indexed to a table
```

To store on another stack

```
  PUSHF:          PUX4
                  PUXS
                  POX4
                  POXS
                  LDB, (-4)
                  TTX1
                  LDW I X1, FLAC+4
                  PUA
                  ATX1, 1
                  JCFZ .-5
                  PUX4
                  PUXS
                  POX4
                  POXS
                  POPJ
```

```
;table sort and branch routine - FOCAL
; calling sequence:
;JSR I, [SORTB]
;(LISTA)              ; parens indicate two bytes
;(LISTB-LISTA)
;                     ; return if not in list

SORTB:    JCTZ , 0+3    ;use AC or CHAR
          LDB ,  CHAR   ;page zero reference
          NEG
          STB,   T2     ;save search character
          POX2
SORT:     LDW I X2, 1   ;pick up list address
          TTX3
          LDB    X3,Ø
          JCTS , SORTX  ;neg quantity signifier
          ADB ,  T2     ;end of list.
          ATX3 , 1      ;update pointer
          JCFZ , SORT
          TFX3          ;final branch address
          ADW I X2, 2
          TTX3
          JMP I  X3,Ø   ;transfer to that location
;default exit
SORTX:    ATX2,4        ;not in list
          JMP I X2,Ø
;100*(31*8-26*12)/31*8=-25%
```

To simulate a "JMS"

```
        .
        .
        .
     PUSHJ  SUB
        .
        .
        .
SUB:        POPA
            STW,  (Ø)
            .
            .
            .
            LDW,  SUB+2
            TTPC
```

Distribution List

Roger Cady
Alan Kotok
Bruce Delagi
Allan Kent
Dave Gross
Tom Eggers
George Thissel
Larry McGowan
John Cohen
Harold McFarland
Jim O'Loughlin
Don Langbein
Tom Stockebrand
Ken Larson