PDP-K Technical Memorandum # 4

Title: An Instruction Set for the 18-bit PDP-K

Author(s): Ad van de Goor

Index Keys: Instruction Set
Op Code
Data Types

Distribution Keys:

Revision: None

Obsolete: None

Date: February 20, 1970

1.0 Introduction

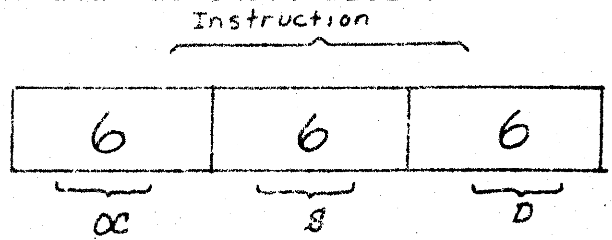
An instruction set for an 18-bit computer[1] is proposed. It combines the best features of the PDP-11's architecture and the PDP-10's instruction set

For several reasons, an 18-bit computer was considered superior; it solves both the op code and address space problems of a 16-bit computer. In addition, it is a better data base in two important area's. Pulse Height Analysis (PHA) programs have proven the need for 18 bits. Also, the 36-bit floating-point representation has much wider acceptance, due to its superiority of 32-bit formats.

[1] I.e., a computer with a word length of 18 bits.

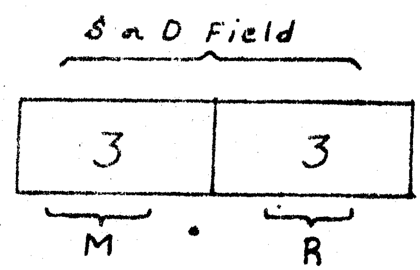
2.0 Instruction Format and Terminology

The instruction format of most binary (two address) instructions is shown below. It resembles that of the PDP-11 and has three fields



Field	Description
OC:	Operation Code Specifies the binary instruction.
S:	Source Specifies the Effective Address (EA) of the source.
D:	Destination Specifies the Effective Address (EA) of the destination.

The formats of the S and D fields are identical and shown below.



Field	Description
R:	Register Denotes 1 out of 8 general registers.
M:	Mode Specifies the addressing mode in a similar way to those for the PDP-11.[1]

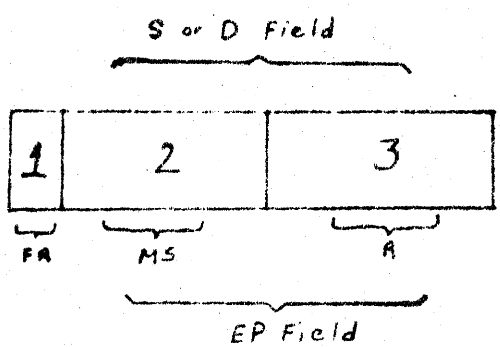
---

[1]See PDP-11 Handbook.

0:	R	; R contains data
1:	@R	; R contains address of data[1]
2:	@(R)+	; Autoincrement, defer
3:	@A(R)	; Index, defer
4:	(R)+	; Autoincrement
5:	-(R)	; Autodecrement
6:	@-(R)	; Autodecrement, defer
7:	A(R)	;Index

The address as computed from the R and M fields is called the Effective Address "EA". When M=0, this is from 0 to 7. The location of the memory cell[2] actually addressed is called the Effective Location "EL". For most binary instructions, EL=EA, i.e., the effective location = the effective address.

In some instructions, the S or D field denotes an integer number; for example, to specify the number of shifts in a shift instruction. The format is as follows:



[1]"@" is used as the "indirect" symbol.

[2]A register is also considered "memory".

Field	Description
R:	Register Denotes 1 out of 8 general registers.
MS:	Modes, Short Specifies the addressing mode. They are identical to the first 4 modes of the M field:

```
0:   R          ; EP[1] is R
1:   @R         ; EP is (R)[2]
2:   @(R)+      ; EP is @(R), autoincrement
3:   @A(R)      ; EP is (R)+A
```

FR:	Free Bit Bit not used to determine EP.

The Effective Position "EP" can be interpreted as the number representing the EA when M would be restricted to the first 4 combinations. The table below shows the values EP can have:

Values of EP

MS	Signed Integer	Unsigned Integer
0: R	-4 to 0; 0 to +3	0 to +7
1: @R	-2^{17} to 0; 0 to $2^{17}-1$	0 to 2^{18}
2: @(R)+	-2^{17} to 0; 0 to $2^{17}-1$	0 to 2^{18}
3: @A(R)	-2^{17} to 0; 0 to $2^{17}-1$	0 to 2^{18}

[1]"EP" = Effective Position

[2]"(R)" = Contents of R.

3.0 Compatibility

Introducing a different word length will cause some compatibility problems.

3.1 Peripheral Compatibility

A separate memorandum will be devoted to this problem. The incompatibility can be reduced by having the same bus structure for the PDP-K as the PDP-11. This is being considered.

3.2 Program Compatibility

Two aspects have to be considered.

3.2.1 Word Length Compatibility

This can be done by hardware by having a 16- and an 18-bit mode; by software through a conversion program similar to that for converting PDP-8 to PDP-9/15 programs leaving certain portions to be recoded "by hand" (e.g., shift and rotate instructions).

3.2.2 Instruction Set Compatibility

This can be accomplished through microprogramming. Because of the PDP-K's 18-bit word length, microprogramming becomes very attractive because the PDP-10 can be emulated.

4.0 Proposed PDP-K Instruction Set

The proposed instruction set is shown in Appendix A. Only the major instructions are shown. These are the essential ones or those requiring lots of op code space. It is assured that the reader has some knowledge of the PDP-11 instruction set.

The instructions operate on 5 data types.

4.1 Bit, "B"[1]

A bit is a Boolean quantity which is true "T" or false 'F".

4.2 Byte, "Y"

A byte is a character

4.3 Word, "W"

A word is:

1. A Boolean Array with 18 elements
2. A signed integer (2's complement)
3. An unsigned integer

4.4 Double Word, "D"

A double word is a single precision, floating-point number.

4.5 Quadruple Word, "Q"

A quadruple word is a double-precision, floating-point number.

---

[1]Denotes abbreviation for the particular data type.

Bytes are handled in a way similar to the PDP-10, as described in Appendix B. Few instructions operate on byte because bytes are considered a data format for characters only.

Most instructions operate on words as the word is considered the data format for program control and integer numbers. It is felt that higher level languages (FORTRAN, ALGOL, etc.) use integers mostly for subscripting and program control and, therefore, a single 18-bit integer is considered sufficient.

The condition code "CC" is handled in a way as described in Appendix C.

5.0 Description of Instructions

Appendix D describes the instruction formats and the interpretation of the fields of the format.

The data type of the instruction will be indicated by a letter following the mnemonic of the instruction. The letters are, as defined before: B = bit, Y = byte, W = word or no letter (default), D = double word and Q = quadruple word. Hence, MOV can be designated by MOVY, MOVW or MOV, MOVD and MOVQ.

The operation of the individual instructions is given below.

MNEM	Operation	Name	Format
MOV	(S)→D ((S)<0)[2]→C1 ((S)=0)→C2 ((S)>0)→C3	Move	#SD[1]
COM	(D)-(S) (r<0)[3]→C1 (r=0)→C2 (r>0)→C3 (Carry=0)→C (Overflow=1)→V	Compare	#SD
COML	(D)-(S),(S+n)[4]-(D) ((D)<(S))→C1 (((D)≥(S))& ((S+n)≥(D)))→C2 ((D)>(S+n))→C3	Compare with Limits	#SD
ADD	(D)+(S)→D (r<0)→C1 (r=0)→C2 (r>0)→C3 (Carry=1)→C (Overflow=1)→V	Add	#SD

---

[1]For instruction format see Appendix D.

[2]"((S)<0)→C1" means: if (S)<0 then 1→C1, else 0→C1.

[3]"r"= result of operation.

[4]"S+n"= next data location from the source.

MNEM	Operation	Name	Format
SUB	(D)-(S)→D For CC[1] see COM	Subtract	#SD
MUL	(D)*(S)→D,D+1 (r<0)→C1 (r=0)→C2 (r>0)→C3 ($\|r\|>2^{17}$)[2]→V	Multiply	#SD
DIV	(D),(D+1)/(S)→D,D+1 (q<0)[3]→C1 (q=0)→C2 (q>0)→C3 ($\|q\|\geq 2^{17}$)→V	Divide	#SD
IMUL	(D)*(S)→D For CC see MUL	Integer Multiply	#SD
IDIV	(D)/(S)→D, D+1 For CC see DIV	Integer Divide	#SD
EXCH	(S)→temp.,(D)→S,(temp.)→D ((S)<0)→C1 ((S)=0)→C2 ((S)>0)→C3	Exchange	#SD
COML	(D)&(S),((D)&(S))⊕(S)→CC	Compare Logical	#SD
	((D)&(S)=0) →CC	All 1's in (S) are 0 in (D)	
	(((D)&(S))⊕(S)≠0) →CC	Some 1's in (S) are 0 in (D)	
	((D)&(S)≠0) →CC	Some 1's in (S) are 1 in (D)	
	(((D)&(S))⊕(S)=0) →CC	All 1's in (S) are 1 in (D)	
AND	(D)&(S)→D (r<0)→C1 (r=0)→C2 (r>0)→C3	Logical AND	#SD

---

[1] CC= condition code.

[2] $|r|$ = absolute value of r.

[3] q = quotient of division.

MNEM	Operation	Name	Format
ANDCS	(D)&(S)'→D For CC see AND.	Logical AND with Comple-mented Source	#SD
IOR	(D)!(S)→D For CC see AND	Logical Inclusive OR	#SD
IORCS	(D)!(S)'→D For CC see AND	Logical Inclusive OR with Comple-mented Source	#SD
XOR	(D)⊕(S)→D For CC see AND	Logical Exclusive OR	#SD
XORCS	(D)!(S)'→D	Logical Exclusive OR with Comple-mented Source	#SD
FADD	(D)+(S)→D (r<0)→C1 (r=0)→C2 (r>0)→C3 (Overflow=1)→V and trap (Underflow=1)→U	Floating ADD	#SD
FSUB	(D)-(S)→D For CC see FADD	Floating Subtract	#SD
FMUL	(D)*(S)→D For CC see FADD	Floating Multiply	#SD
FDIV	(D)/(S)→D For CC see FADD	Floating Divide	#SD

MNEM	Operation	Name	Format
AOS	(D)+1→D	Add One and Skip	#CSKP
	if (CC=T), then (PC)+R→PC When skip condition is satisfied, the PC is incremented with the value in the R field (0 to 7) of the instruction.		
SOS	(D)-1→D	Subtract One and Skip	#CSKP
	if (CC=T), then (PC)+R→PC		
TSTS	(D)→D	Test and Skip	#CSKP
	if (CC=T), then (PC)+R→PC		
AOJ	(R)+1→R	Add One and Jump	#CJMP
	if (CC=T), then (D)→PC		
SOJ	(R)-1→R	Subtract One and Jump	#CJMP
	if (CC=T), then (D)→PC		
TSTJ	(R)→R	Test and Jump	#CJMP
	if (CC=T), then (D)→PC		
LSH	Shift (D)→D	Logical Shift	#EPD, #LSH
	The shift direction and the number of shifts depend on the sign and absolute value of the number determined by the EP in the S field of the instruction. (r<0)→C1 (r=0)→C2 (r>0)→C3 (last bit shifted out)→C (Overflow=1)[1]→V		
LSHC	Shift Combined (D),(D+1) →D,D+1 For explanation and CC see LSH	Logical Shift Combined	#EPD, #LSHC

[1]Overflow occurs (on left shifts and rotates only) whenever the value of the two most significant bits of (D) become unequal. Once V is set, it stays set. On a right shift or rotate, V is cleared.

MNEM	Operation	Name	Format
ROT	Rotate (D)→D	Rotate	#EPD, #ROT

The rotate direction and the number of bit positions rotated depend on the sign and absolute value of the number determined by the EP in the S field of the instruction.

(r<0)→C1
(r=0)→C2
(r>0)→C3
(last bit rotated out)→C
(Overflow=1)[1]→V

MNEM	Operation	Name	Format
ROTC	Rotate Combined (D),(D+1) →D,D+1 For explanation and CC, see ROT.	Rotate Combined	#EPD, #ROTC
ASH	Shift Arithmetically (D)→D For explanation and CC, see SH.	Arithmetic Shift	#EPD, #ASH
ASHC	Shift Arithmetically Combined (D),(D+1)→D,D+1 For explanation and CC, see LSH.	Arithmetic Shift Combined	#EPD, #ASHC
BIS	1→EBL[2]	Bit Set	#EPD

The EBL is determined as follows: the EA of the D field of the instruction is taken, starting from the beginning of the word denoted by EA, EP bit locations are counted. Note: EP is allowed to be bigger than 18.

(EBL)[3] & (C)→C1
(EBL)[3] ! (C)→C2
(EBL)[3] 6 (C)→C3
(EBL)[3] →C

MNEM	Operation	Name	Format
BICL	0→EBL For explanation and CC, see BIS.	Bit Clear	#EPD

---

[1] Overflow occurs (on left shifts and rotates only) whenever the value of the two most significant bits of (D) become unequal. Once V is set it stays set. On a right shift or rotate, V is cleared.

[2] "EBL" = effective bit location.

[3] In here it is meant the (EBL) prior to change.

MNEM	Operation	Name	Format
BICM	(EBL)'→EBL For explanation and CC, see BIS.	Bit Complement	#EPD
BICP	(C)→EBL (EBL)[1]&(C)→C1 (EBL)[1]!(C)→C2 (EBL)[1]⊕(C)→C3 (C)→EBL	Bit Copy	#EPD
BIT	(EBL)→EBL For explanation and CC, see BIS.	Bit Test	#EPD
BITC	(EBL)→EBL (EBL)'&(C)→C1 (EBL)'!(C)→C2 (EBL)'⊕(C)→C3 (EBL)'→C	Bit Test Complement	#EPD
BIMS	(EBL)→(-SP)	Bit Move to Stack	#EPD
	The (EBL) is pushed on the stack as if it were an 18-bit word.		
BIMM	(SP)+→EBL	Bit Move to Memory	#EPD
	If (SP)=0, then 0→EBL else 1→EBL.		
SMOV	(D)→-(SP)	Stack Move	#EPD
	This is a move from memory to the stack (R6 is implied stack pointer). EP is interpreted as a post index and the FR field is interpreted as a post indirect bit. EA[2]= if FR=0 then EA+EP else (EA+EP) For CC, see MOV.		
MMOV	(SP)+→D	Memory Move	#EPD
	This is a move from stack to memory. For further explanation and CC, see SMOV.		
BR	if (CC=T) then (PC)+(OFFS)→PC	Branch	#BR
	When the branch condition is satisfied, the offset (a 9-bit signed quantity) is added to the PC.		

---

1 In here it is meant the (EBL) prior to change.

MNEM Operation — Name — Format

JSR, JSP — Jump to Subroutine — #EPD

Special subroutine call, passes parameters to the stack automatically. See Appendix E.

ANAL — Analyse — #SD

To be defined later.

REPS — Repeat Single — #REP

The EP is interpreted as an unsigned integer representing the repeat count "RC". The repeat action is stopped when (RC=0)!(CC=T). When REPS stops and (CC=T)&(RC≠0), then the remainder of the repeat count is pushed on the stack, i.e., RC_{rem} →-(SP).

REPD — Repeat Double — #REP

Repeat next two instructions. For explanation, see REPS.

JMP if (CC=T) the (D)→PC — Jump — #JMP

Jump takes place when jump condition is satisfied.

XCT if (CC=T) then Execute — Execute — #JMP

When condition satisfied, the instruction denoted by (D) is executed.

XCTU if (CC=T) the Execute Undisturbed — Execute Undisturbed — #JMP

When condition satisfied, the instruction denoted by (D) is executed undisturbed, i.e., the result of the operation is not stored only the CC is set.

6.0 Register Seven

General register "R7" is used in the PDP-11 as the PC (program counter). Because of this, certain addressing modes are not advisable or lead to "self-destruction" of the program. The table below shows this.

ADDRESSING MODES FOR R7

Source		Destination	
R7	OK	R7	OK
@R7	OK	@R7	Error
@(R7)+	OK	@(R7)+	OK
@A(R7)	OK	@A(R7)	OK
(R7)+	OK	(R7)+	NR[1]
-(R7)	Error	-(R7)	Error
@-(R7)	Error	@-(R7)	Error
A(R7)	OK	A(R7)	OK

It is suggested not only to prevent the programmer from making these errors, but also to turn these faulty combinations into something useful.

6.1 Use the destination mode (R7)+ the normal way except do not store the result of the operation. This way all binary instructions become "test immediate" instructions.

[1]"NR" = produces non-reentrant code.

6.2 Use the destination modes -(R7) and @-(R7) as flags indicating the following.

6.2.1 -(R7) Case

Consider the instruction a stack operation with the stack (i.e., there where R6 points to) as the destination and as source the contents of "(R5)+EN". The Effective Number "EN" is the contents of the S field of the instruction interpreted as an unsigned integer (i.e., from 0 to 63). The binary instructions look like:

(SP) Operation ((R5)+EN)→(SP)

6.2.2 @-(R7) Case

Operation similar to the -(R7) case except as source the contents of ((R5)+EN) is taken. Binary instructions look like:

(SP) Operation @((R5)+EN)→(SP)

APPENDIX A

PROPOSED PDP-K INSTRUCTION SET

Count	Instruction	Description		Bit	Byte	Word	DW[1]	QW[2]
4	MOV	(S)→D	Move		✓	✓	✓	✓
4	COM	(D)-(S)	Compare		✓	✓	✓	✓
4	COML	(D)-(S),(S+n)[3]-(D)	Compare with Limits		✓	✓	✓	✓
1	ADD	(D)+(S)→D	Add			✓		
1	SUB	(D)-(S)→D	Subtract			✓		
1	MUL	(D)*(S)→D,D+1	Multiply			✓		
1	DIV	(D),(D+1)/(S)→D,D+1	Divide			✓		
1	IMUL	(D)*(S)→D	Integer Multiply			✓		
1	IDIV	(D)/(S)→D,D+1	Integer Divide			✓		
1	EXCH	(D)⇄(S)	Exchange			✓		
1	COML	(D)&(S)→CC ((D)&(S))⊕(S)→CC	Compare Logical			✓		
1	AND	(D)&(S)→D	And			✓		
1	ANDCS	(D)&(S)'→D				✓		
1	IOR	(D)!(S)→D	Inclusive Or			✓		
1	IORCS	(D)!(S)'→D				✓		
1	XOR	(D)⊕(S)→D	Exclusive Or			✓		
1	XORCS	(D)⊕(S)'→D				✓		
2	FADD	(D)+(S)→D	Floating Add				✓	✓
2	FSUB	(D)-(S)→D	Floating Subtract				✓	✓
2	FMUL	(D)*(S)→D	Floating Multiply				✓	✓
2	FDIV	(D)/(S)→D	Floating Divide				✓	✓
							✓	✓
1	AOS	(D)+1→D, skip?	Add One and Skip			✓		
1	SOS	(D)-1→D, skip?	Subtract One and Skip			✓		
1	TSTS	(D)→D, skip?	Test and Skip			✓		

[1] DW = double [2] QW = quadruple word [3] S+n = next data word

Count	Instruction	Description		Bit	Byte	Word		
1	AOJ	(R)+1→R, jump?	Add One and Jump			✓		
1	SOJ	(R)-1→R, jump?	Subtract One and Jump			✓		
1	TSTJ	(R)→R, jump?	Test and Jump			✓		
1/2	LSH		Logical Shift			✓		
1/2	LSHC		Logical Shift Combined			✓		
1/2	ROT		Rotate			✓		
1/2	ROTC		Rotate Combined			✓		
1/2	ASH		Arithmetic Shift			✓		
1/2	ASHC		Arithmetic Shift Combined			✓		
1/2	BIMS	(EBL)→-(SP)[4]	Bit Move to Stack	✓				
1/2	BIMM	(SP)+→EBL	Bit Move to Memory	✓				
1/2	BIS	1→EBL[1]	Bit Set	✓				
1/2	BICL	0→EBL	Bit Clear	✓				
1/2	BICM	(EBL)'→EBL	Bit Complement	✓				
1/2	BICP	(C)[2]→EBL	Bit Copy	✓				
1/2	BIT	(EBL)→CC[3]	Bit Test	✓				
1/2	BITC	(EBL)'→CC	Bit Test Complement	✓				
3	SMOV	(D)→-(SP)[4]	Stack Move, Multiple Indexed			✓	✓	✓
3	MMOV	(SP)+→D	Memory Move, Multiple Indexed			✓	✓	✓
2	BR		Branch					
2	JSR, JSP		Subroutine Call					
1	ANAL		Analyze					
1/8	REPS	Repeat Single	Cond, N					
1/8	REPD	Repeat Double	Cond, N					
1/4	JMP	Jump	Cond, D					
1/4	XCT	Execute	Cond, D					
1/4	XCTU	Execute Undisturbed	Cond, D					
4/64	TST	(D)→CC	Test		✓	✓	✓	

[1]EBL = effective bit location

[2](C) = contents of carry, status bit

[3]CC = condition code

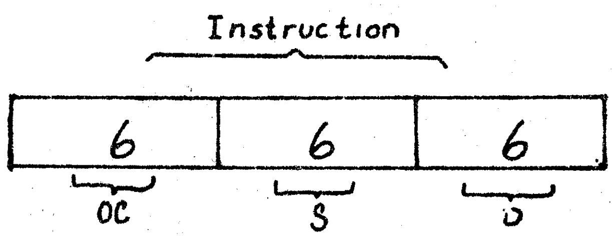
[4]SP = stack pointer

Count	Instruction	Description		Bit	Byte	Word	D	QW
4/64	SETZ	0→D	Set to all Zeros		✓	✓	✓	✓
1/64	SETPO	1→D	Set to Plus One			✓		
1/64	SETMO	-1→D	Set to Minus One					
1/64	ADDC	(D)+(C)→D	Add Carry			✓		
1/64	SUBC	(D)-(C)→D	Subtract Carry			✓		
1/64	TOC	(D)'→D	Take One's Complement			✓		
3/64	TTC	(D)'+1→D	Take Two's Complement			✓	✓	✓
1/64	CIFS	(D)→-(SP)	Convert Integer to Float. Single			✓		
1/64	CIFD	(D)→-(SP)	Convert Integer to Float. Double			✓		
1/64	DFSI	(D)→-(SP)	Convert Float. Single to Integer				✓	
1/64	CFSD	(D)→-(SP)	Convert Float. Single to Float. D.				✓	
1/64	CFDI	(D)→-(SP)	Convert Float. D. to Integer				✓	
1/64	CFDS	(D)→-(SP)	Convert Float. D. to Float. S.				✓	
1/64	INCBP		Increment Byte Pointer		✓			
1/64	DECBP		Decrement Byte Pointer		✓			
1/256	MCCS	(CC)→-(SP)	Move CC to Stack					
1/256	MCCC	(CC)→C	Move CC to C Bit					
1/4096	MSCC	(SP)+→C	Move Stack to C Bit					
3/64	NECH	(D)⇄(D+n)	Next Exchange			✓	✓	✓
1/64	LOCK	((D)=0)⊃(SP)+→D	Lock			✓		

APPENDIX B

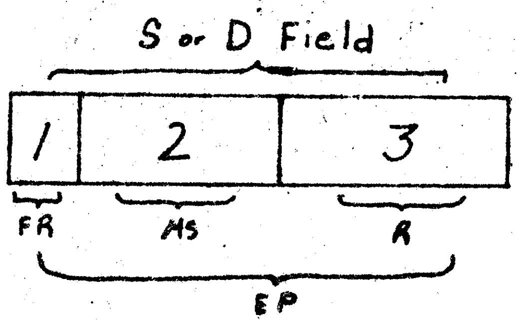
PDP-K Byte Handling

The PDP-K will handle bytes in the same manner as the PDP-10. The format of the byte instructions will be similar to all other instructions.



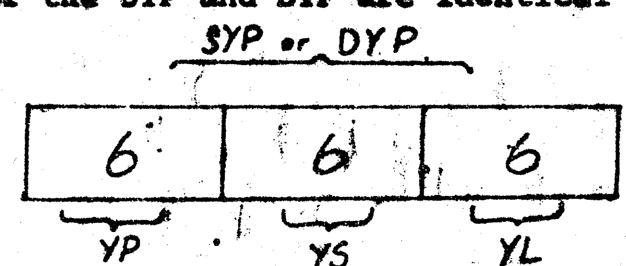
The possible OC's are MOVY, COMY, and COMLY.

The S and D fields are identical in format and define the locations of the source and destination byte pointers "SYP and DYP". The S and D fields are interpreted the same way as the EP field, described in section 2.0 and as shown below.



The locations of the SYP (source byte pointer) and the DYP (destination byte pointer) are determined by the contents of the EP's of the S and D fields of the instructions. The free bits "FR" are used to allow for incrementing the byte pointer.

The formats of the SYP and DYP are identical and shown below.

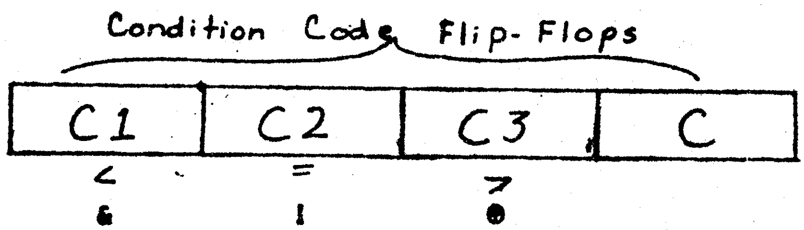


Field	Description
YP	The position of the first bit of the byte in the double word addressed by YL.
YS	The length of the byte in bits.
YL	YL is interpreted as a regular destination and denotes the location of the double word containing the byte.

APPENDIX C

Condition Codes

The PDP-K condition code differs from the PDP-11 because of the special requirements imposed by the single bit diddling instructions[1] of PDP-K. The instructions making use of the condition code have 4 bits to specify the condition. The function of 4 of the condition code flip-flops will be discussed below.



C1: indicates "<" in arithmetic operations
indicates "&" in single bit operations

C2: indicates "=" in arithmetic operations
indicates "!" in single bit operations

C3: indicates ">" in arithmetic operations
indicates "●" in single bit operations

C: carry bit also used as test bit in single bit operations

In arithmetic operations the flip-flops C1, C2, C3 and C are used as listed in the table below and interpreted as follows. C1=1 when the result is <0; C2=I when result =0; C3=1 when result >0, and C=1 when there is a carry or when there is no borrow.

In the case of bit diddling, the flip-flops are used as follows:

(EBL)[2]&(C)[3]→C1
(EBL)!(C)→C2
(EBL)●(C)→C3
(EBL)→C

---

[1] See Appendix A instructions BIS, BICL, BICD, BIT, BITC and BICP.

[2] EBL = contents of Effective Bit Location, complemented when the BTC (bit test complement) instruction is used.

[3] (C) = contents of the carry flip-flop.

The operation above allows all 16 boolean operators between 2 variables directly and allows complex boolean equations to be evaluated easily.

The interpretation of the contents of the flip-flops C1, C2 and C3 for signed arithmetic and bit didling is shown below and required 8 "condition code combinations" out of the 16 total.

TABLE C1

	& < C1	! = C2	● > C3	Signed Arithmetic Interpretation		Bit Diddling Interpretation	
0	0	0	0	False	BNOT		
1	0	0	1	>	BGT	●	BXOR
2	0	1	0	=	BEQ	!	BIOR
3	0	1	1	≥	BGE	&'	BNAND
4	1	0	0	<	BLT	&	BAND
5	1	0	1	≠	BNE	!'	BNIOR
6	1	1	0	≤	BLE	●'	BNXOR
7	1	1	1	True	BRA	True	BRA

The remaining 8 combinations are used as shown in the table below. Together with the BEQ and BNE conditions from above they contain all conditions for unsigned arithmetic.

TABLE C2

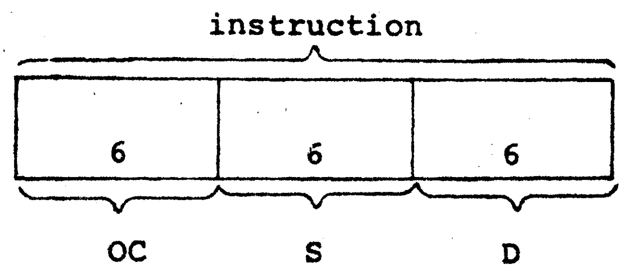
	Special Condition Interpretation		Unsigned Arithmetic Interpretation	
0	Repeat count = 0	BZR		
1			>	BHI
2	Overflow	BOV		
3	No Carry	BNCA	≥	BHIE
4	Carry	BCA	<	BLO
5	No Overflow	BNOV		
6			≤	BLOE
7	Repeat count ≠ 0	BNZR		

APPENDIX D

Instruction Formats

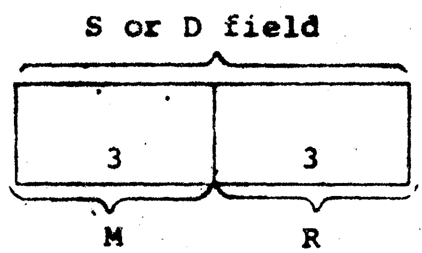
D.1 Format #SD, Source Destination

Instruction has 3 fields of 6 bits



OC = operation code
S = source
D = destination

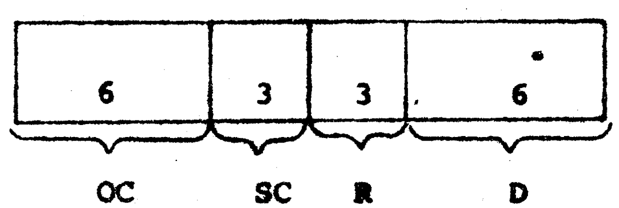
The S and D fields have the same format as shown below.



R = register field
M = mode field

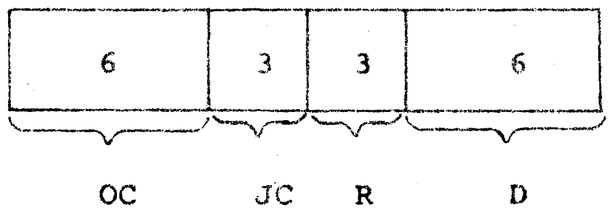
D.2 **Format #CSKP, Conditional Skip**

Instruction has 4 fields. The SC field (skip condition) is interpreted as in Table C1 of Appendix C. The R field contains the number of words to be skipped (from 0 to 7).



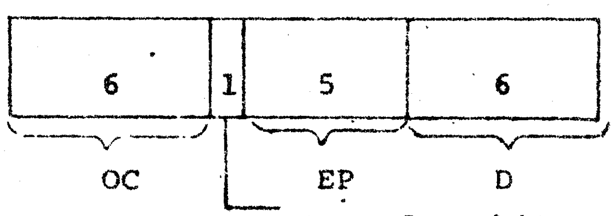
D.3 Format #CJMP, Conditional Jump

This instruction has 4 fields. The JC field contains the jump condition, interpreted as shown in Table Cl of Appendix C. The R field denotes the register to be tested after an increment (decrement or test).



JC = jump condition

D.4 **Format #EPD, Effective Position-Destination**

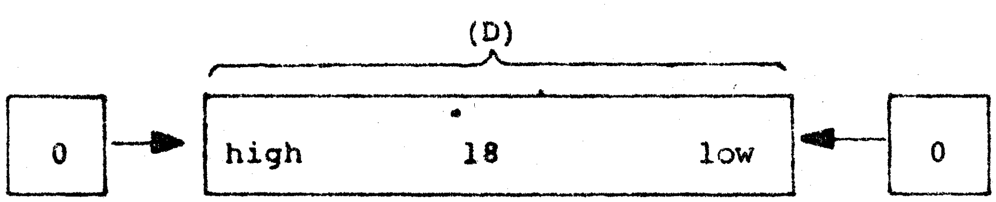


EP = effective position field

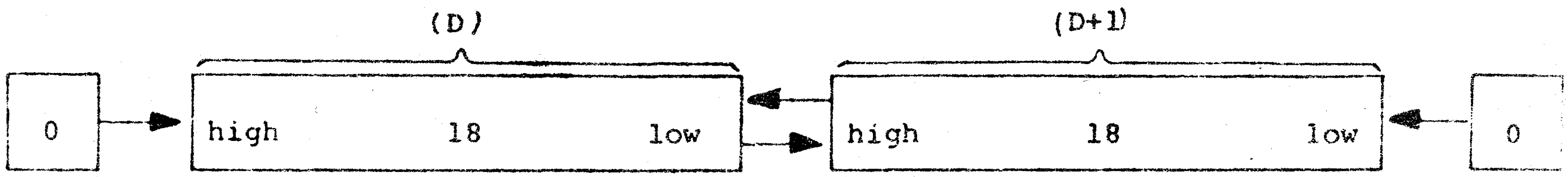
FR = free bit used to extend the OC field

D is a regular destination field, EP is a regular effective position field.

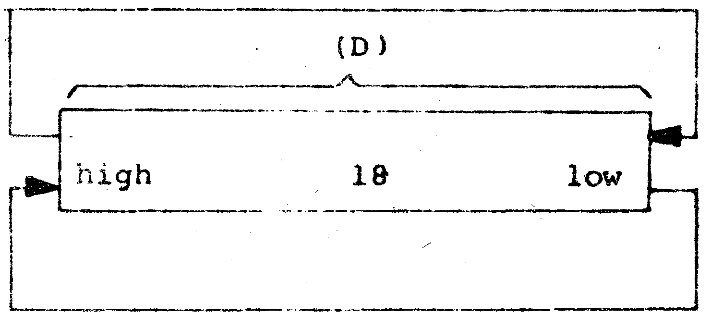
D.5 **Format #LSH, Logical Shift**



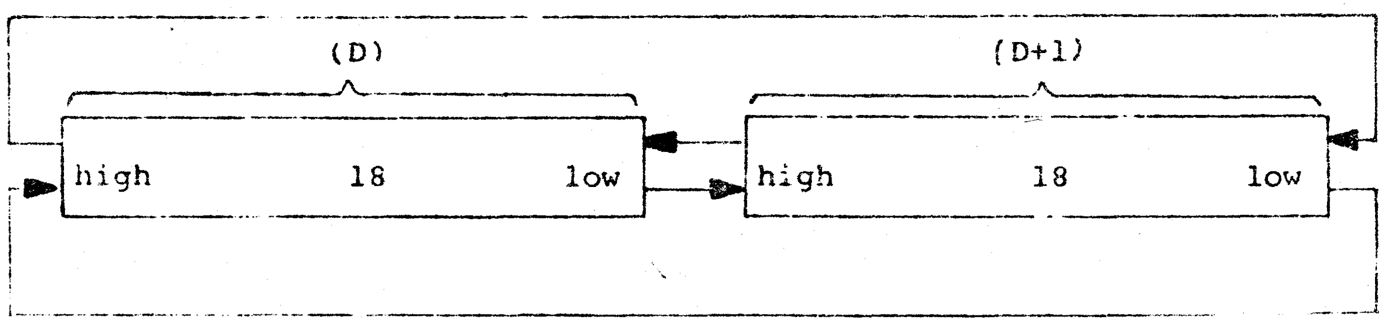
D.6 **Format #LSHC, Logical Shift Combined**



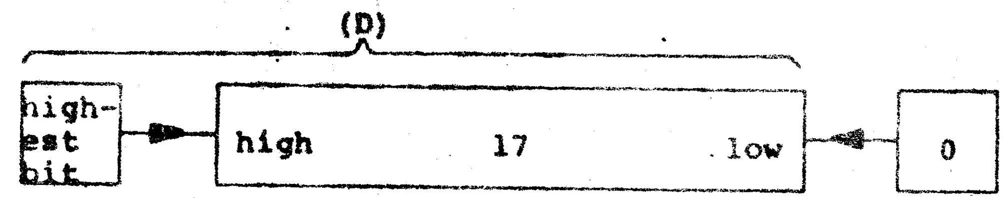
D.7 **Format #ROT, Rotate**



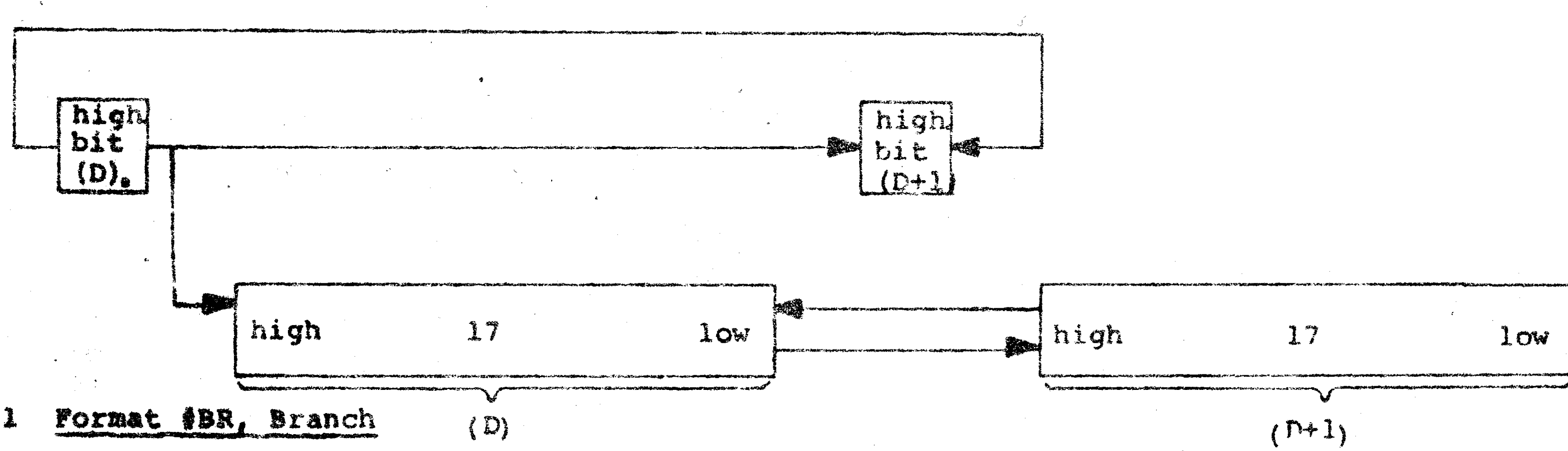
D.8 **Format #ROTC, Rotate Combined**



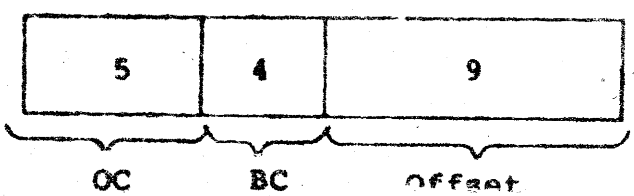
D.9 Format #ASH, Arithmetic Shift



D.10 Format #ASHC, Arithmetic Shift Combined

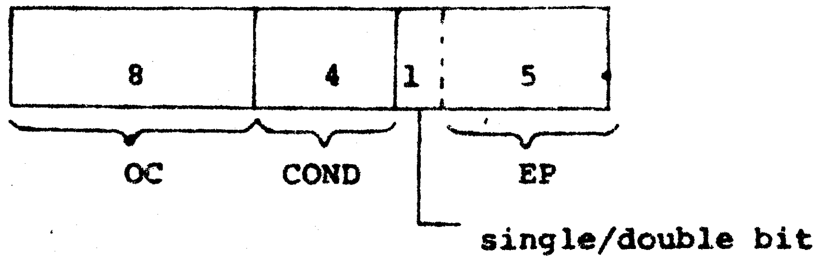


D.11 Format #BR, Branch

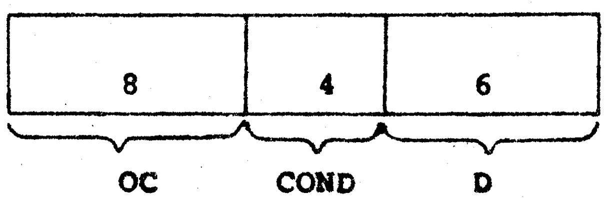


BC = branch condition
Offset = 9-bit signed integer

D.12 Format #REP, Repeat



D.13 Format #JMP, Jump



APPENDIX E

Subroutine Calls

Besides the standard PDP-11 JSR, the PDP-K will have a more powerful subroutine call. This new call "JSP" (Jump to Subroutine with Parameters) automatically passes parameters to the stack and does "stack house-keeping" in such a way that subroutine returns can be done in a trivial way while the stack is "cleaned up" automatically.

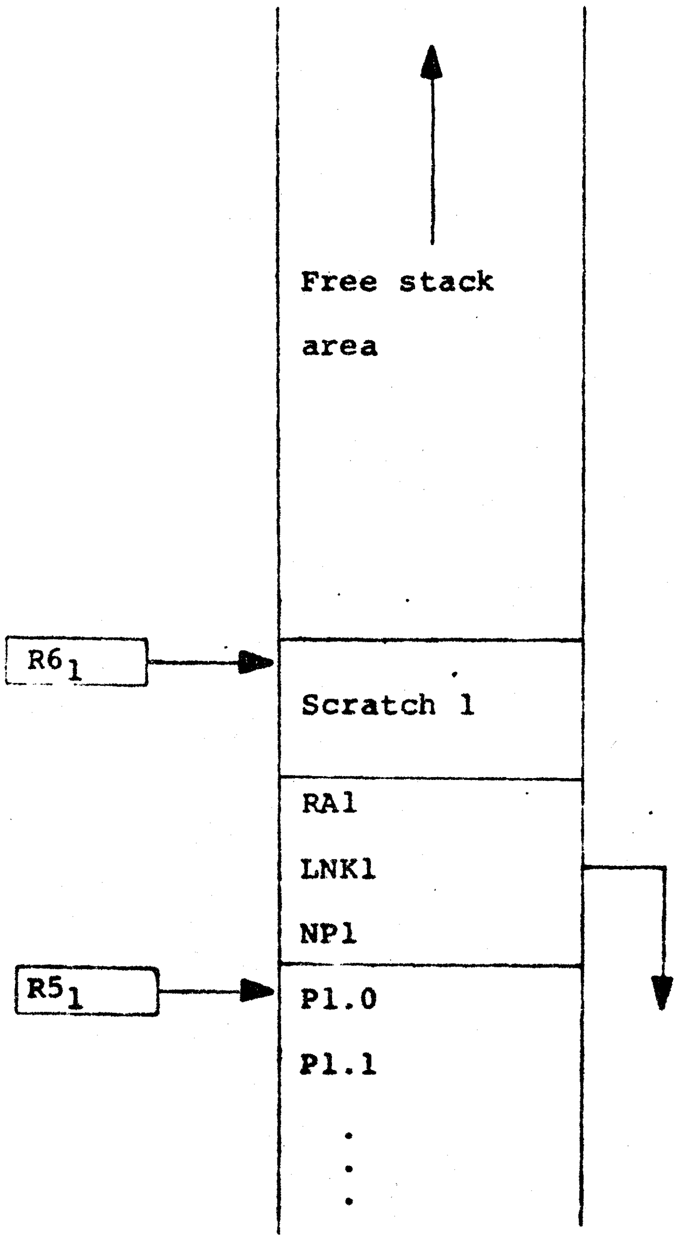
The format of the call is #EPD where the EP field is interpreted as the number of parameters to be pushed on the stack. Register R5 is used to point to the first passed parameter after it has been pushed on the stack. The example below shows how the JSP could be implemented. Note that in addition to the parameters themselves, three other quantities have to be pushed on the stack to allow for automatic updating upon return from the subroutine.

1. The number of parameters "NP"

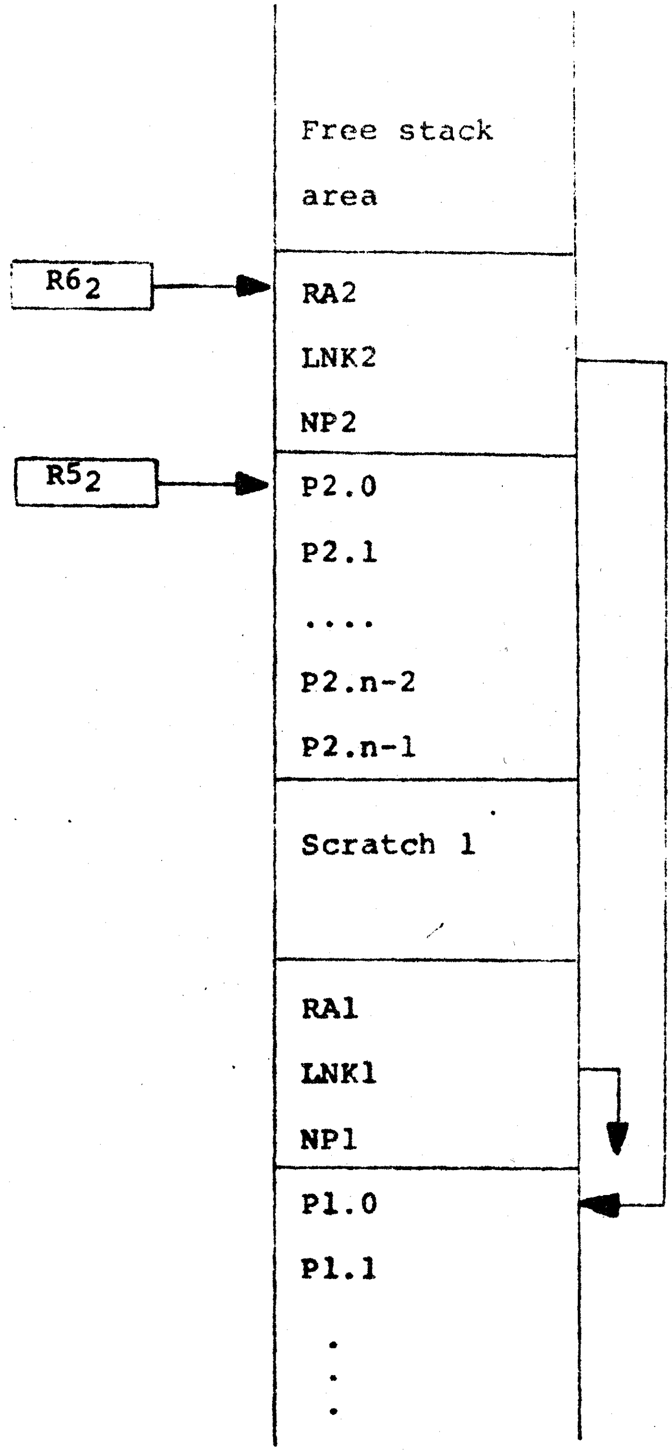
2. A link to the previous call "LNK"

3. The return address "RA"

Below is shown how the JSP actually operates. The left stack shows the situation just prior to the call of subroutine 2, the right stack shows the situation just after the call.



Stack just prior
to the call "JSP
n, SUB2"



Stack just after
the call "JSP
n, SUB2"

The passing on of parameters which are passed as parameters is taken care of by giving the to-be-passed-on parameter an address relative to the parameter pointer, i.e., (R5). A parameter following a subroutine call is considered a "new" parameter when its value is ≥64 and a passed parameter otherwise. See example below:

```
        JSP      n,SUB1     /call SUB1 with n parameters
        P1.0
        P1.1

        P1.n-1

        SUB1,    -----
                 -----
                 JSP m,SUB2            /call SUB2 with m
                 P2.0                  /parameters
                 P2.1
                 P2.2
                    1                  /parameter ≤ 63 so it
                 P2.4                   is interpreted as a
                                        passed parameter, not
                 P2.m-1                 "1" but ((R5)+1) will
                                        be pushed on the stack.
                                        This is just parameter
                                        P1.1 of the previous
                                        call.
```