Lecture Notes from Manchester University Computer Science course 1970-71.

Scanned November 2011
by John Buchanan
who took the notes himself

END OF MARY ALMOND NOTES

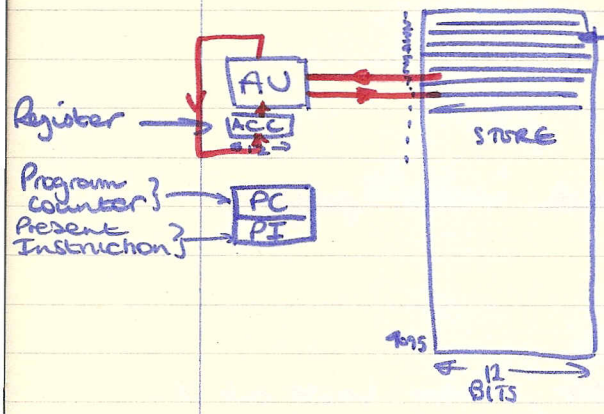
MACHINE CODING

I.E. WRITING PROGRAMS IN M/C CODE LANGUAGE.

WHY? 1. Test Programs.
2. Basic System Programs e.g. Compilers, and Operating Systems.
3. Better Efficiency (for programs which are to be used many times)
4. Small Machines may not be big enough to hold a compiler

PDP 8 M/C.

Programmers view of m/c

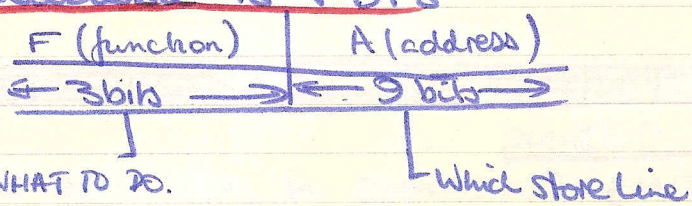


This takes the place of a variable in AA but the store must also contain the machine code program.
The Program Counter contains the address of the next instruction
let value of selected store line be [A]

Operation of Machine

Start on new instruction
- Takes instruction from store Line where address is in "P.C." and place in "P.I."
- add 1 to 'P.C'
- decode instruction in P.I
- carry out action specified by instruction

Actions available to PDP8



Function List: AND 000 Acc = Acc & [A]
- TAD: 001 Acc = Acc + [A]
- ISZ: 010
- DCA: 011 [A] = Acc, Acc = 0
- JMS: 100
- JMP: 101 PC = [A]
- FUNNY 110
- VERY FUNNY 111

Addressing System

ie meaning of A

'A' = 9 bits

Store size = 4K = 4096

A | Y | X | LINE IN PAGE |

Consider the store to be divided into 32 pages each of size 128 lines

If X is 0 then goto page 0
1 then goto page in 'PC'

If Y is 0 then address is direct, then look at X
1 the address is indirect

Indirect Addressing

ACC = ACC + (Line 2/P.0) ← 0010000000010

ACC = ACC + [[LINE 3. PAGE 0]] 0011000001 1

P.D.P. 8 instrumentation and controls.

On the front of the machine there are 26 switches the left hand 6 are not used. The next 12 are the switch register and denote '1' when in the up pos^{n} and '0' when down. The next 8 are :- START, LOAD ADDR, DEP, EXAM, CONT, STOP, SINGLE STEP, SINGLE INST.
Lamps in rows of 12 are, top → bot :- PROG COUNT, MEMORY ADDR, MEMORY BUFFER, ACCUMULATOR.

To load a program manually.

1. Set address of 1st instruction on switch register
   Operate 'LOAD ADDR'
   This sets P.C to switch register (S.R)
2. Set bit pattern of 1st Instruction on S.R.
   Operate 'DEP'
   This copys S.R into the store line given by P.C.
   and adds one to P.C.
3. Set next inst on SR
   Operate 'DEP'
4. etc etc.

To examine contents of store

1. Set address of 1st store line on SR
   Operate 'LOAD ADDR'
   This operates as 1 above

2. Operate Examine 'EXAM'
This does ACC = [PC]
PC = PC + 1

To obey Program.
CASE (A) 1 instruction at a time under operator control.
1. Set single step "UP"
2. Set single inst "DOWN"
3. Set entry address on SR
Operate "LOAD ADDR"
4. Operate "START"
This clears the accumulator and obeys one instruction and adds one to PC.
5. Operate "CONT"
this obeys one instruction and adds one to PC
Do this again etc.

CASE(B) Automatically at full machine speed
1. Set entry address on S.R
Operate "LOAD ADDR"
2. Set Single Step "UP"
Single Inst
3. Operate "START"

Simple Examples to be loaded into machine by hand
1. Sum the nos in storelines 120-127 of P1
Program in lines 0-7 of Page 1
Assume Acc = 0

CS104

		octal
LINE 0 / P1	0010111111000	ACC + [SL(120)] ≡ 1370
L1	0010111111001	ACC + [SL(121)] ≡ 1371
L2	0010111111010	ACC + [SL(122)] ≡ 1372
L3	0010111111011	ACC + [SL(123)] ≡ 1373
L4	0010111111100	ACC + [SL(124)] ≡ 1374
L5	0010111111101	ACC + [SL(125)] ≡ 1375
L6	0010111111110	ACC + [SL(126)] ≡ 1376
L7	0010111111111	ACC + [SL(127)] ≡ 1377

2. Write who each store line in its own address

Program in P31

Call some work space X

Set X = -1 initially

- ACC = X
- ACC = ACC + 1
- ACC ⇒ X
- ACC ⇒ [X]
- JUMP

	binary		octal
L0/P31	0010100001 10	≡	1206
L1	0010100001 11	≡	1207
L2	0110100001 10	≡	3206
L3	0010100001 10	≡	1206
L4	0111100001 10	≡	3606
L5	1010100000 00	≡	5200
L6	1111111111 11	≡	7777
L7	0000000000 01	≡	0001

Instruction ISZ ≡ 2.

Used to program 'cycles'

ISZ 'A' [A] = [A] + 1,

Skip next instruction if [A] = 0

EXAMPLE 1

	LINE	OCTAL
Clear ACC	0200	
Set count = -8	0201	
Set SL 'X' = ADDRESS OF LINE 120	0202	
→ ACC = ACC + [X]	0203	
X = X + 1	0204	
ISZ 'COUNT'	0205	
— JMP	0206	
STOP	0207	
	0210	
	0211	
	0212	
	0213	

0200	3215	
0201	1213	
0202	3216	
0203	1214	
0204	3215	
0205	1615	
0206	2215	
0207	2216	
0210	5205	
0211	5211	
0212	0000	
0213	7770	-8
0214	0370	ADDR OF LINE 2 - P1
0215	0000	X
0216	0000	COUNT

Summary of Work so Far

1) Instructions are 12 binary bits long
2) Basic instructions specify operations to be performed on 12 bits from the store (a store 'word') and in some cases a 12 bit accumulator.
3) Top 3 bits specify operation

Binary	octal	Operation
000	0	ACC = ACC & STORE
001	1	ACC = ACC + STORE
010	2	INC STORE, SKIP IF ZERO
011	3	ACC ⇒ STORE, ACC = 0
100	4	JUMP TO SUBROUTINE
101	5	JUMP
110	6	INPUT/OUTPUT INSTRUCTIONS
111	7	SPECIAL INSTRUCTIONS

4) Instruction splits in two parts
   a) 3 bit operation
   b) Word from store i.e. the remaining 9 bits specify which word.

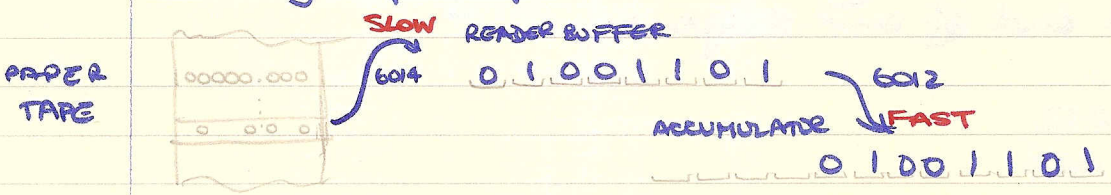
Operation Codes 6 and 7.

The Remaining 9 bits are not used to specify store word in these instructions.
These bits are used to specify one of many more instructions which do not require a word from store

EXAMPLES

6012	110,000,001,010	Acc = CHARACTER FROM READER BUFFER
6014	110,000,001,100	READER BUFFER = PATTERN OF HOLES ON PAPER TAPE
6011	110,000,001,001	SKIP IF READER BUFFER READY
7041	111,000100001	ACC = - ACC

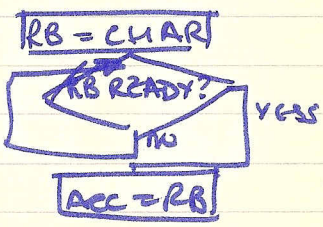
Reading Paper Tape



Example

1) Write a sequence of instrs at address 1 of some page to get a character into the accumulator.

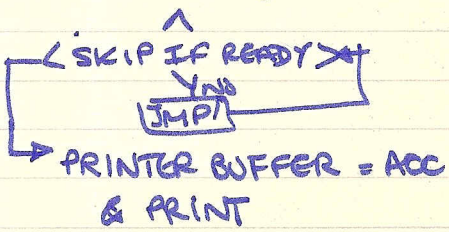
INS T. ADDR	
1	6014
2	6011
3	5202
4	6012



2) Same for printing (at address 7 of some page)

INST. ADDR.	
7	6041
8	5207
9	6046

PRINTER



We could now write such a sequence every time printing or reading was required, use of the JMS (code 4) instruction avoids this duplication

Example

INST. ADDR.		
11	4200	JMS specifying word of current page
12		

JMS is basic and specifies store word. The address of nesct instruction is stored here and we jump to the following address

∴ 11+1 = 12 stored in word 0 jump to word 1

We have made a subroutine jump to our read a character sequence.

We get back by an indirect jump via word 0, at the end of our sequence

ADDR.	
0	8 12
1	6014
2	6011
3	5202
4	6012
5	5600
6	
7	
8	
9	
10	
11	4200
12	4200

Operating instruction 7.

This is split into Group 1 7000
and Group 2 7400

1. 7000 No Operation
   7100 Clear link (link is the carry at top of acc)
   7200 Clear accumulator.
   7040 Complement acc :- change 0's to 1's and vice versa
   7010 Moves all digits in acc (link) one to the right
   7004 " " " " " " " " " left
   7012 " " " " " " two " " right
   7006 " " " " " " " " " left
   7001 Adds one to acc

2. See piece of paper.

Boot Strapping

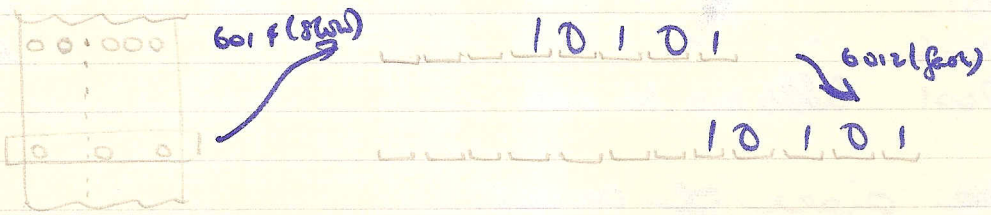
Write simple (short) program to read other programs.
(Not very readable programs)
Write less simple prog to read other programs (more readable)
etc etc.
We start our bootstrap with octal loader

Conventions of Octal Loader.

1) Ignore all characters up to and including the first octal digit
2) Digit "9" marks the end of the program.
3) Octal digits are seperated in groups of 4 by "CR" "LF"
4) Tape Code 5-hole

Revision

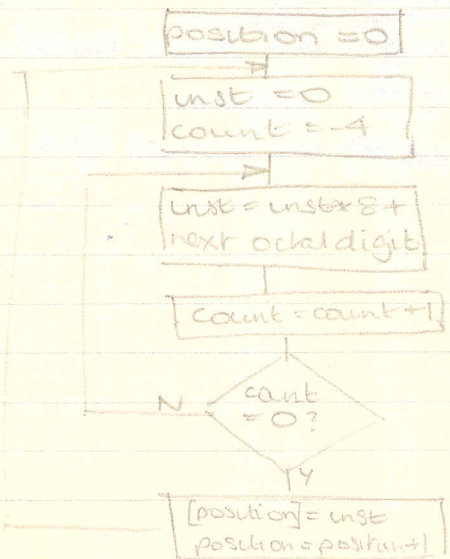
1) We have established a need for a program which will permit us to punch programs on paper tape rather than load them into the computer store on the handkeys.
2) This reading program (called an octal loader) must be short, as it will have to be loaded on the handkeys. We achieve this by having a simple form for our punched programs:- 4 octal digits per instruction.
3) Picture of paper tape on [illegible] is wrong, insert:-

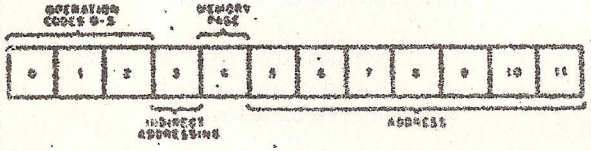


Octal loader

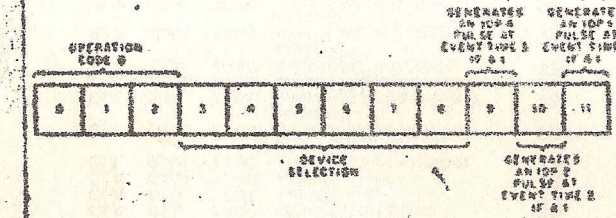
First let us form groups of 4 octal digits into an instruction. We shall assume:-

1) A store word [illegible] that we shall call 'position' holds the address of another store word in which the instruction is to be placed.
2) The notation [position] means the content of the store word whose address is given by 'position'





Memory Reference Instruction Bit Assignments



IOT Instruction Bit Assignments

ASCII CODE

Character	Code	Character	Code
A	301	!	241
B	302	"	242
C	303	#	243
D	304	$	244
E	305	%	245
F	306	&	246
G	307	'	247
H	310	(	250
I	311	)	251
J	312	*	252
K	313	+	253
L	314	,	254
M	315	-	255
N	316	.	256
O	317	/	257
P	320	:	272
Q	321	;	273
R	322	<	274
S	323	=	275
T	324	>	276
U	325	?	277
V	326	@	300
W	327	[	333
X	330	\	334
Y	331	]	335
Z	332	↑	336
0	260	←	337
1	261	EOT	204
2	262	W RU	205
3	263	RU	206
4	264	BELL	207
5	265	Line Feed	212
6	266	Return	215
7	267	Space	240
8	270	ALT MODE	375
9	271	Rub Out	377

EQUIPMENT CORPORATION
MAYNARD, MASS. 01754

PRINTED IN U.S.A. 25-1/67

F-96/5
1/67

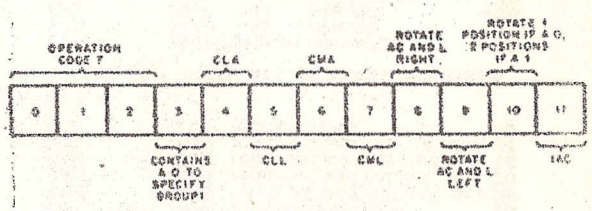
PDP-8/S

INSTRUCTION LIST

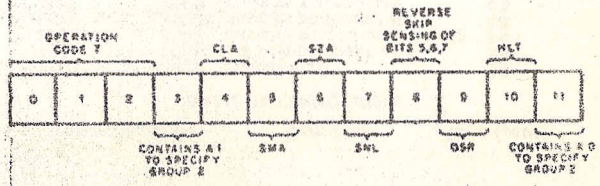
Mnemonic	Code	Operation	Time (μsec)
BASIC INSTRUCTIONS			
AND	0000	logical AND	36
TAD	1000	2's complement add	36
ISZ	2000	increment and skip if zero	54
DCA	3000	deposit and clear AC	46
JMS	4000	jump to subroutine	46
JMP	5000	jump	28
IOT	6000	in/out transfer	36*
OPR (I)	7000	operate GROUP 1	28
OPR (II)	7400	operate GROUP 2	38
GROUP 1 OPERATE MICROINSTRUCTIONS			
NOP	7000	no operation	28
CLA	7200	clear AC	28
CLL	7100	clear link	28
CMA	7040	complement AC	28
CML	7020	complement link	28
RAR	7010	rotate AC and link right one	28
RAL	7004	rotate AC and link left one	28
RTR	7012	rotate AC and link right two	28
RTL	7006	rotate AC and link left two	28
IAC	7001	increment AC	28
GROUP 2 OPERATE MICROINSTRUCTIONS			
SMA	7500	skip on minus AC	38
SZA	7440	skip on zero AC	38
SPA	7510	skip on plus AC	38
SNA	7450	skip on non-zero AC	38
SNL	7420	skip on non-zero link	38
SZL	7430	skip on zero link	38
SKP	7410	skip unconditionally	38
OSR	7404	inclusive OR, switch register with AC	38
HLT	7402	halts the program	38
CLA	7600	clear AC	38

Mneumonic		Code	Operation	Time (μsec)
COMBINED OPERATE MICROINSTRUCTIONS				
CIA		7041	complement and increment AC	2B
LAS		7604	load AC with switch register	3B
STL		7120	set link (to 1)	2B
GLK		7204	get link (put link in AC bit 11)	2B
CLA	CLL	7300	clear AC and link	2B
CLA	IAC	7201	set AC = 1	2B
CLA	CMA	7240	set AC = −1	2B
CLL	RAR	7110	shift positive number one right	2B
CLL	RAL	7104	shift positive number one left	2B
CLL	RTL	7106	clear link, rotate 2 left	2B
CLL	RTR	7112	clear link, rotate 2 right	2B
SZA	CLA	7640	skip if AC = 0, then clear AC	3B
SZA	SNL	7460	skip if AC = 0, or link is 1, or both	3B
SNA	CLA	7650	skip if AC ≠ 0, then clear AC	3B
SMA	CLA	7700	skip if AC < 0, then clear AC	3B
SMA	SZA	7540	skip if AC ≤ 0	3B
SMA	SNL	7520	skip if AC < 0 or link is 1, or both	3B
SPA	SNA	7550	skip if AC > 0	3B
SPA	SZL	7530	skip if AC ≥ 0, and if the link is 0	3B
SPA	CLA	7710	skip if AC ≥ 0, then clear AC	3B
SNA	SZL	7470	skip if AC ≠ 0 and link = 0	3B
IOT MICROINSTRUCTIONS				
PROGRAM INTERRUPT				
ION		6001	turn interrupt on	3B
IOF		6002	turn interrupt off	3B
TELETYPE KEYBOARD/READER				
KSF		6031	skip if keyboard/reader flag = 1	3B
KCC		6032	clear AC and keyboard/reader flag	3B
KRS		6034	read keyboard/reader buffer, static	3B
KRB		6036	Clear AC, read keyboard buffer, clear keyboard flag	3B

Mneumonic	Code	Operation	Time (μsec)
TELETYPE TELEPRINTER/PUNCH			
TSF	6041	skip if teleprinter/punch flag = 1	3B
TCF	6042	clear teleprinter/punch flag	3B
TPC	6044	load teleprinter/punch buffer, select and print	3B
TLS	6046	load teleprinter/punch buffer, select and print, and clear teleprinter/punch flag	3B
PARITY ERROR			
SMP	6101	skip on no memory parity error	3B
CMP	6104	clear memory parity error	3B
HIGH SPEED PERFORATED TAPE READER TYPE PC02			
RSF	6011	skip if reader flag = 1	3B
RRB	6012	read reader buffer, and clear flag	3B
RFC	6014	clear flag and buffer and fetch character	3B
HIGH SPEED PERFORATED TAPE PUNCH TYPE PC03			
PSF	6021	skip if punch flag = 1	3B
PCF	6022	clear flag and buffer	3B
PPC	6024	load buffer and punch character	3B
PLS	6026	clear flag and buffer; load and punch	3B
EXTENDED MEMORY CONTROL MC8S			
CDF	62n1	change to data field n	2B
CIF	62n2	change to instruction field n	2B
RDF	6214	read data field into AC 6-8	2B
RIF	6224	read instruction field into AC 6-8	2B
RMF	6244	restore memory field	2B
RIB	6234	read interrupt buffer	2B



Group 1 Operate Instruction Bit Assignments



Group 2 Operate Instruction Bit Assignments

Say ← 45 31

INST = 000 000 000 000
000 000 000 100 (+4)
000 000 100 000 (×8)
000 000 100 101 (+5)
000 100 101 000 (×8)
⋮
100 101 011 001

Disadvantages

1) instructions must go in store word 0 onwards
2) It doesn't ever seem to stop
3) It doesn't permit us to have any blank tape in our punched program.

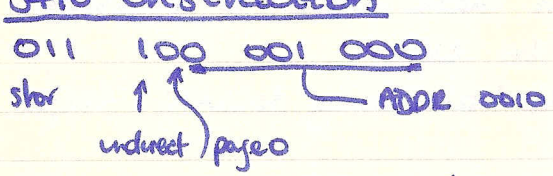
We may improve our program to cope with these points by:-

1) treating first group of four octal digits as a 12 bit address - the starting adress for our instructions
2) Ignoring blank tape etc. when reading our next octal digit
3) Recognising the character '9' specially when reading octal digits. If we have remembered the address of the first instruction we could enter the program at this point

---

We shall now write octal loader program; in order to simplify it, we shall use a subroutine to form a 12 bit instruction and another to read an octal digit.

ADDRESS octal	INSTR. octal	comment.
0030	4240	Jms to form instruction
0031	3010	ACC ⇒ position, ACC = 0 } position is at 0010
0032	1010	ACC = ACC + POSITION }
0033	3237	ACC ⇒ STARTING ADDRESS, ACC = 0
0034	4240	Jms to form instruction
0035	3410	STORE INSTR. or ACC ⇒ [POSITION]
0036	5234	JUMP TO FORM LOOP
0037	0000	STARTING ADDRESS
~~0040~~		cont'd below ⇓
0041		



Store words 0010 to 0017 have a special significance on the PDP8 computer.
If we use one of them in an indirect instruction the content has 1 added to it before the instruction is obeyed
Thus 3410 means:- 1) Add 1 to store word 0010
2) Store the acc in the store word whose address is now in the store word 0010.

0040	0000	
0041	1255	See
0042	3253	next
0043	7004	page
0044	7006	
0045	3259	
0046	4256	
0047	1254	
0050	2253	
0051	5243	
0052	5640	

```
0090    0000    Return addr. stored here by JMS
0091    1255    ACC = ACC + -4
0092    3253    ACC => COUNT , ACC = 0
0093    7004    } ACC = ACC * 8
0094    7006    }
0095    3254    ACC => INST , ACC = 0
0096    4256    JMS TO READ OCTAL DIGIT
0097    1254    ACC = ACC + INST
0050    2253    INC COUNT SKIP IF ZERO
0051    5243    JMP TO FORM LOOP
0052    5640    JMP TO ADDR STORED BY JMS
0053    0000    COUNT
0054    0000    INST
0055    7774    -4
```

Read Symbol

Read the next octal digit from the paper tape to the accumulator
Ignore blank or CR & LF
Enter program when 9 is read

An assembly language for the P.D.P. 8

This describes in convenient form a machine code program. From this description A program called an "assembler" will assemble the m/c instructions inside the machine.

Define assembly Language

Except for "control" statements, each statement is equivalent to a machine instruction

<program> ::= <sections><enter stat.>
<sections> ::= <section>|<sections><section>
<enter stat.> ::= E<octal word>
<section> ::= <starting addr.><word sequence>
<starting addr> ::= *<octal word><newline>
<word sequence> ::= <labelled word>|<word sequence><labelled word>
<labelled word> ::= <word><newline>|<label><word><newline>
<word> ::= <instruction>|<const>
<instruction> ::= <mnemonic code><addr. part>
<const> ::= <octal word>|+<integer>|-<integer>
<label> ::= (<integer>)
<octal word> ::= <octal digit><octal digit><octal digit><octal digit><octal digit><octal digit>
<integer> ::= <decimal digit>|<integer><decimal digit>
<newline> ::= *"CR"*"LF"*|<comment>*"CR"*"LF"*
<comment> ::= <text>

SAL = Simple ASSembly Language

To translate mnemonic codes we need table

AND = 0

TAD = 1

ISZ = 2

DCA = 3 etc.

In general to represent a 3 letter mnemonic code in the machine 15 bits are required

SAL imposes restrictions in order to manage with 12-bits.

∴ only 16 letters ~~many~~ of alphabet may be used in mnemonic codes:-

P, H, D, T, L, R, J, Z, N, A, M, C, S, K, I

∴ in SAL

<mnemonic code> ::= AND | CLA | CLL | CMA | CML | DCA | HLT |
ISZ | IAC | JMS | JMP | RAR | RAL | RTR |
RTL | SMA | SZA | SPA | SNA | SNL | SZL |
SKP | TAD | CIA |

<addr. part> ::= <label> | <label>* | <octal digit><octal digit>
<octaldigit> | <empty>

Example of the use of SAL

Multiply Subroutine :- see sheet.

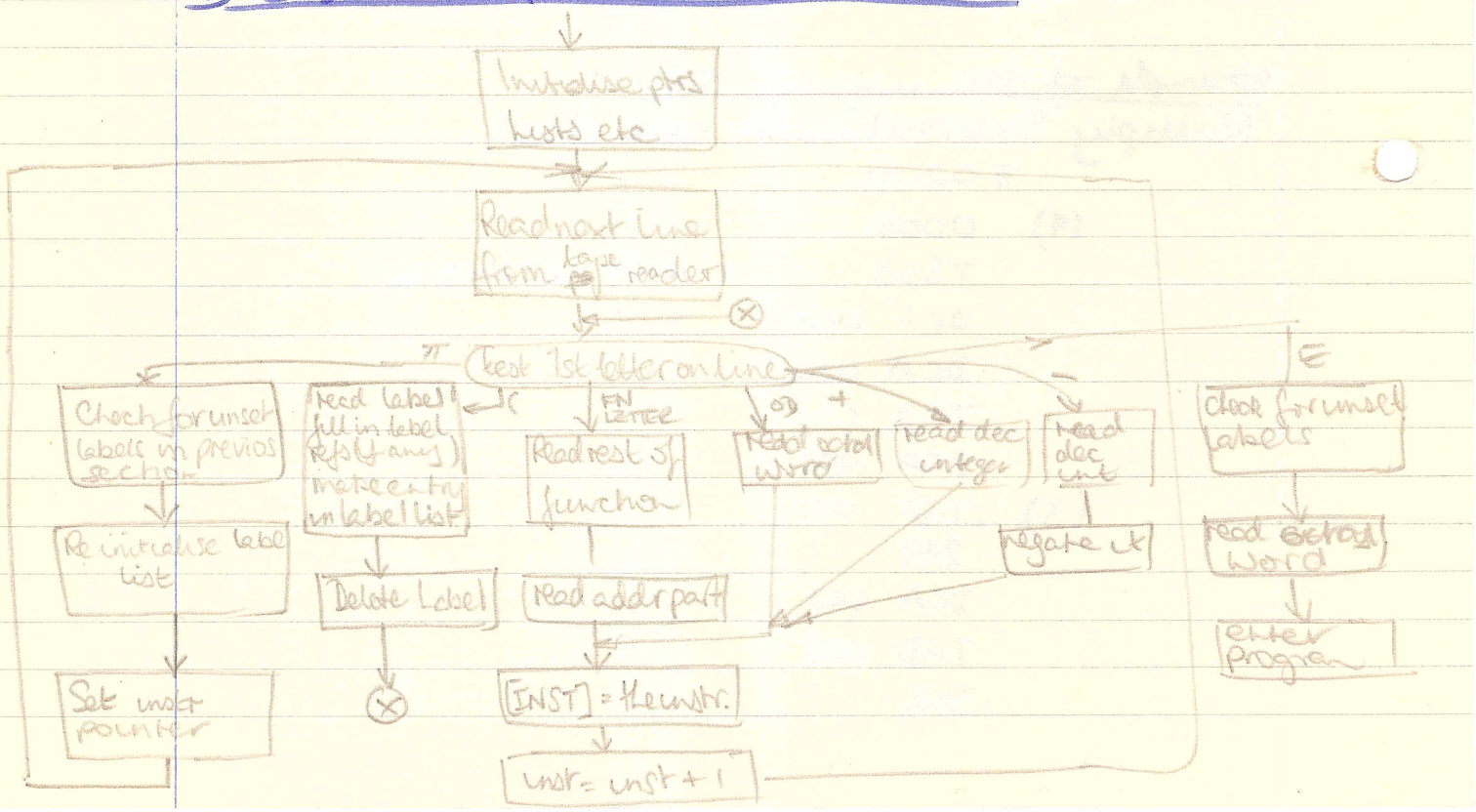
```
        π0200
(4)     0000
        7300        CLEAR ACC AND L
        DCA 003
        DCA 002
        TAD (1)
        DCA 004
(3)     TAD 001
        RAR
        DCA 001
        TAD 002
        SNL
```

```
        JMP  (2)
        CLL
        TAD  000
(2)     RAR
        DCA  002
        TAD  003
        RAR
        DCA  003
        ISZ  004
        JMP  (3)
        JMP  (4)*
(1)     -12
```

Restrictions on SAL

1) A section must not cross a page boundary
2) Each section has its own labels which cannot be accessed from another section.
3) No label may be greater than 127

DESIGN OF THE SAL ASSEMBLER



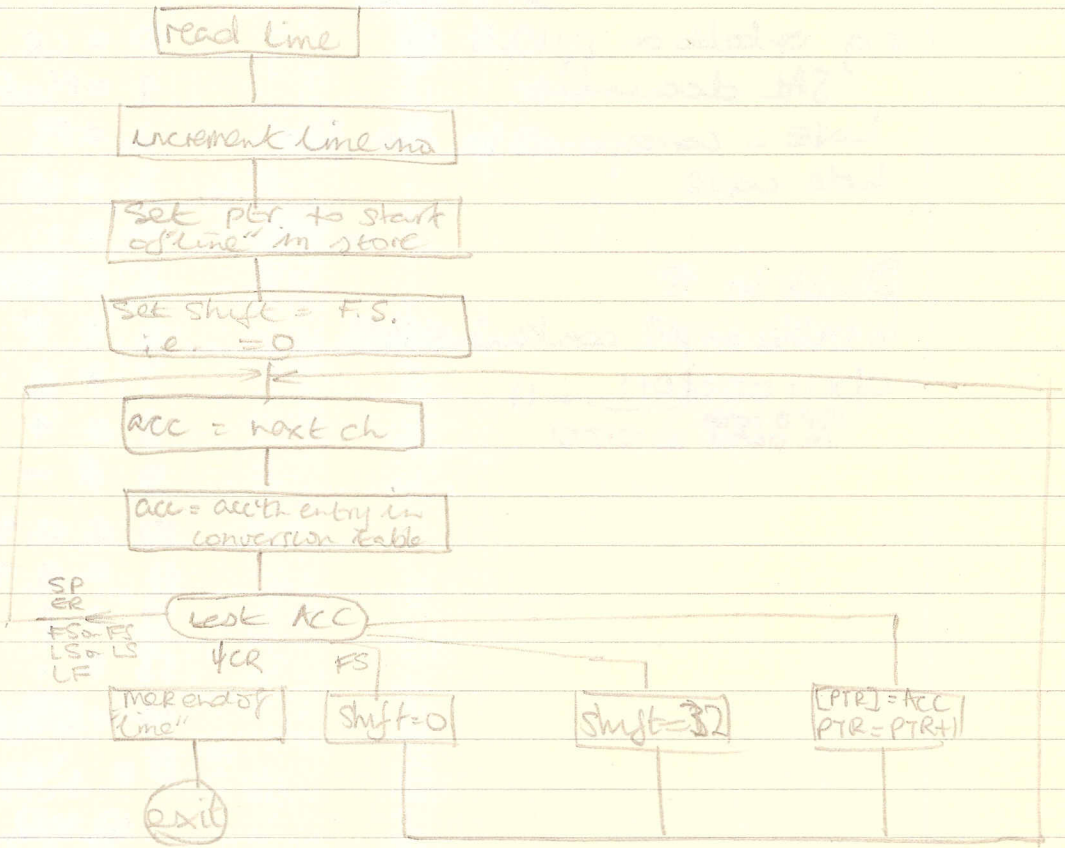
READLINE (AND EDIT)

Function

Read the next line of CH's from reader and ignore blanks.
Delete irrelevant CH's e.g. erase character (ER)

space
F.S. or F.S. → fig shift
L.S. or L.S. → letter shift.

Store the line 1 character/word.
Convert from 5-hole code to a more convenient form.



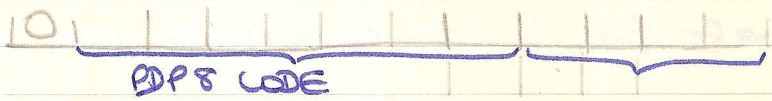
Conversion table
Each Entry = 12 bits
Two Cases : p.t.o

i) Function Letter.

PDP 8 CODE. distinguish each function letter. e.g.

a) 0000
b) 0001
d) 0010

ii) Not function letter.



e.g. on table on pp 10, 11 of SAL document

LINE 2 corresponds to the five hole code

Which is 8

In table on p10 contains 1613

= 0b1111000|011 = 11

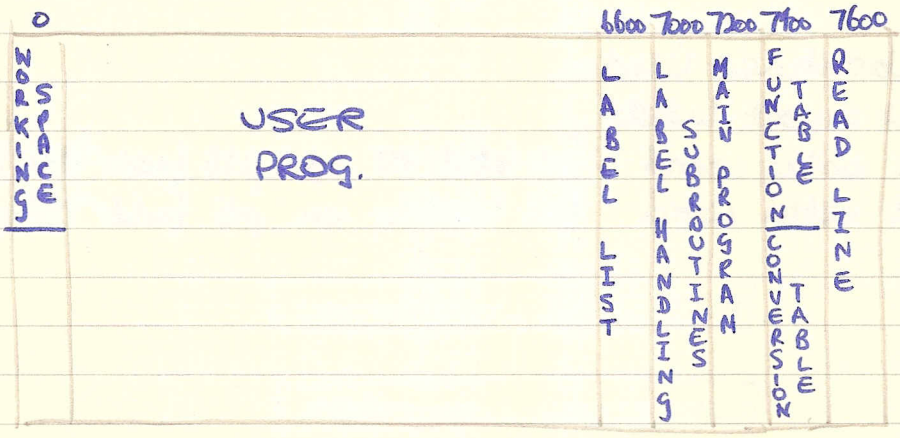
PDP 8 code in octal ≡ 070

0 ≡ CR
1 ≡ FS on FS, LS on LS, ER, SP, LF
2 ≡ FS
3 ≡ LS
4 ≡ :
5 ≡ SPARE
6 ≡ π
7 ≡ E
8 ≡ +
9 ≡ −
10 ≡ 0, 4, 2, 6, 1, 5, 3, 7
11 ≡ 8, 9
12 ≡ (
13 ≡ .
14 ≡ SPARE
15 ≡ ;:?<=)>|'*/, X, B, F, V Q, Y, U, _, G, W, O

To initialise the program (SAL)

Set fault count = 0
Set line count = 0
NI = 0177
clear label list

Use of Store by SAL

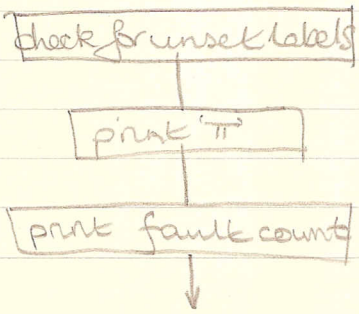


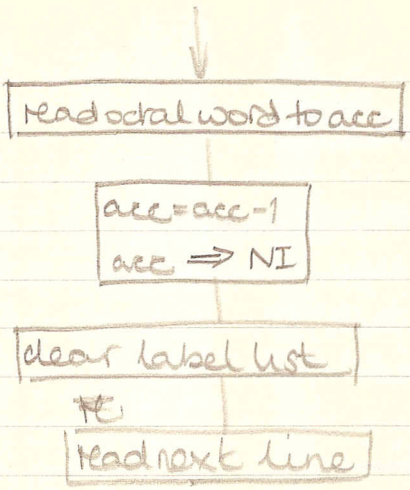
Types of Line acceptable to SAL.

1. π <octal Word>
2. e <octal word>
3. + <decimal integer>
4. - <decimal integer>
5. <octal word>
6. <mnemonic instr>
7. (<decimal integer>) <"any of the above">

Treatment of the above by SAL

1. π

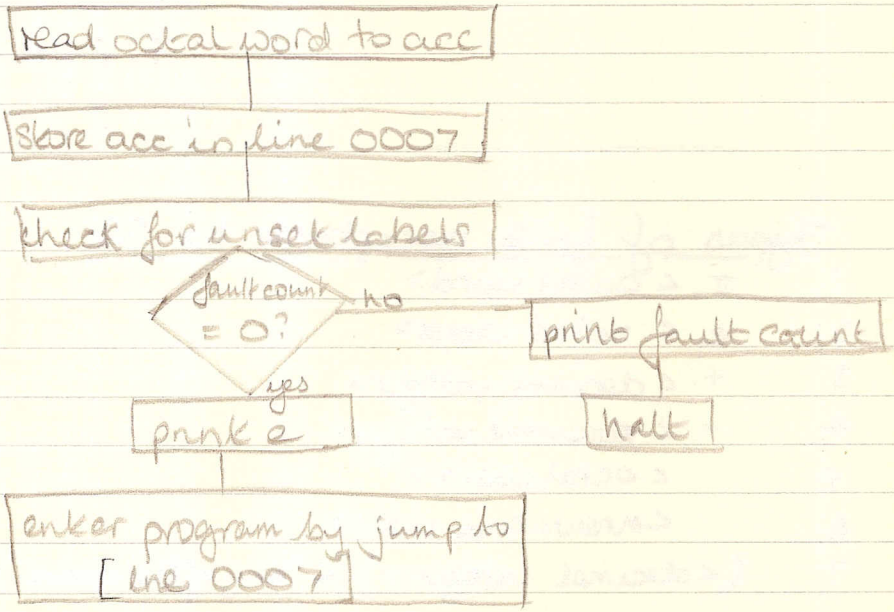




Main Subroutines Used:-

i) count unset labels
ii) print octal word (at label 80 on p18 (addr 7136))
iii) read octal word (at label 40 on p5 (addr 7665))

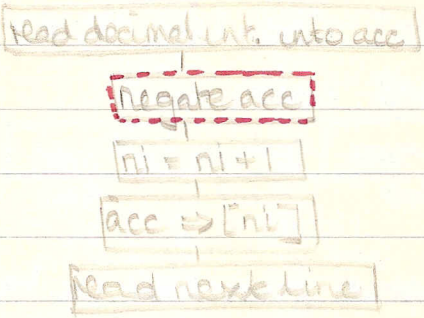
2 e.



Main Subroutines Used:-

i) print octal word

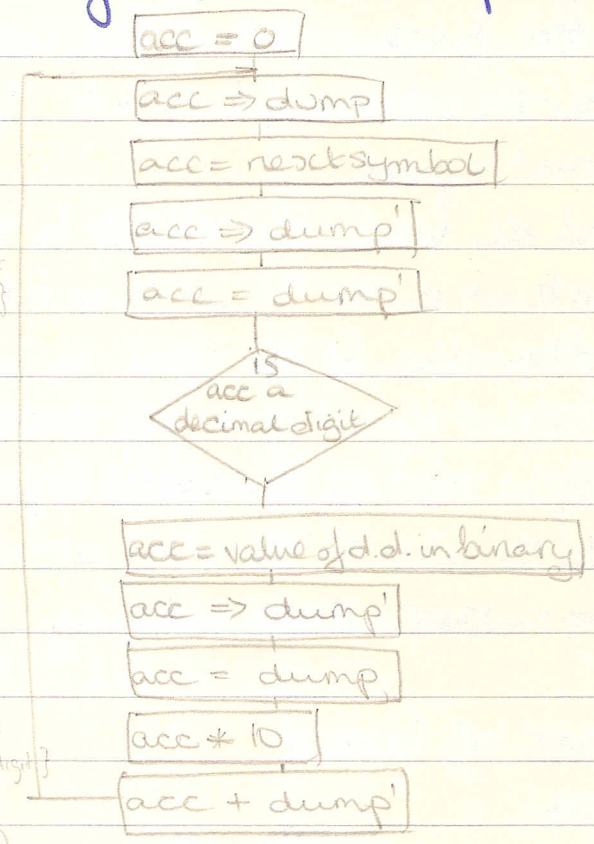
3.4.



Main Subroutines Used:-

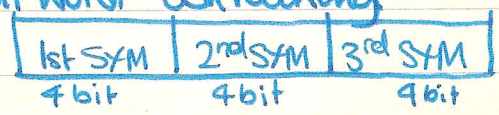
i) Read Decimal Integer (at label 60 on p.6 (addr 7717))

```
        7300
(59)    dca 001
        tad 416
        dca 002
        tad 002
        and (1)
        tad (61)
        sza
        jmp (62)    {jmp if not
(63)    tad 002       8,9 }
        rtr
        rtr
        and (1)
        dca 002
        cll
        tad 001
        rtL
        tad 001
        ral
        tad 002
        jmp (59)
(62)    iac
        sna
        jmp (63) {jmp if octal digit}
        7300
        tad 001
        jmp (60)*(exit)
```



6. Action taken by sal when line starts with function letter.

i) form a 12-bit word containing



ii) Look up mnemonic function in table on page 8

iii) Take the binary machine fn. from the corresponding pos" on page 9.

iv) Store the function part of the instruction

v) Remember the address of this instr (in line 0003)

vi) ~~af~~ Test next symbol

a) (: read the label and convert to binary (= L)
look at entry 'L' in label list
add the value of the label to instr
if the next symbol is * add 0400 to instr

b) octal digit: read all three octal digits and convert to binary
add the value of the octal address part into instr.

c) ~~nothing~~ neither: does nothing.

7 Treatment of labels.

The problem:

(3) TAD 001 — LABEL SET NOT PREVIOUSLY REFERENCED
JMP (3) — LABEL REFERENCED AND PREVIOUSLY SET
JMP (4) — LABEL REFERENCED NOT PREVIOUSLY SET
(4) — LABEL SET WHICH HAS PREVIOUSLY BEEN SET

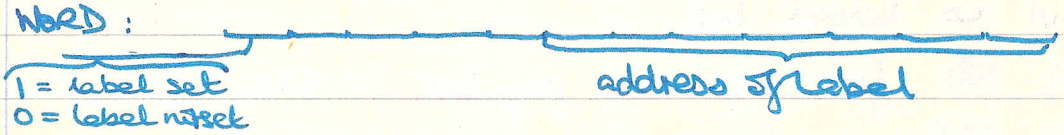
Solution

In the label list one word for each label which must be able to show: a) label set / not set

b) label referenced / not referenced

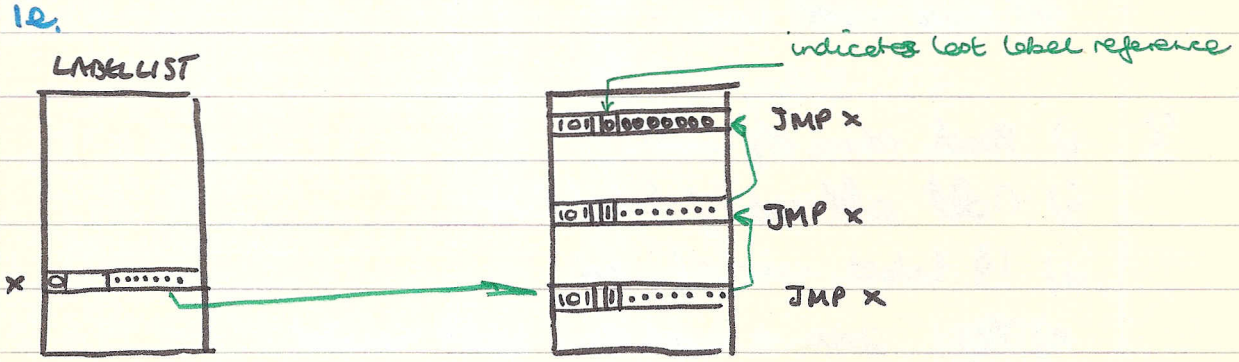
c) address associated with label

d) information about instructions which reference the the label before it is set.



In the case of the label not set the part of the label list word used for the address is given the address of one of the label references

ie.



Subroutines Used.

1. Check for unset labels (7000 label (90))
2. Translate reference to label (7023 .. (95))
3. Set Label (7066 .. (32))

1. If the label list word is +ve label is not set.
2. i) Read value of label
   ii) Add address of label list
   iii) Store acc in line 000
   iv) Add into address field of instr being translated the bottom 8 bits of [line 000]
   v) is label set
   a) NO: form reference pointer to the instruction
   b YES: continue
   vi) test for '*'
   a) '*' set indirect bit
   b) No '*' do nothing
   vii) Read next line.

3. i) Read value of label
   ii) Add address of label list
   iii) Is label undefined
   a) Yes: acc = addr of next instr.
      acc & 0177
      acc + 4200
      Store acc in the relevant pos' in label list
      read fn. part of instr.
   b) No. has label been set already?
      Y: increment fault count - exit
      N: acc & 0177

Dump acc in line 001
Acc = addr of next instr.
acc & 7600
acc = acc + [line 001]
dump acc in line 001
reset the link in the label list.
acc = addr of next instr (-1)
acc & 0177
acc = acc + 0201 (acc: line addr of next instr, + current page digit)
dump acc in 002
acc = 1st instr in reference chain
acc & 7400
acc = acc + 002
acc ⇒ 1st instr in reference chain
Return to instr (iii).

PROBLEMS IN USING A PDP/8

1. Too few functions
   e.g. −, *, / would be useful.
2. Too few address digits in word.
   e.g. be able to address the whole store directly
3. Acc has to be dumped very often.
4. Store too small.
5. No floating point facility