KTU STUDY MATERIALS

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

DOWNLOAD KTU ASSIST APP FROM PLAYSTORE

MODULE 2
ASSEMBLERS

* Assembleas convert assembly language into machine language

Fundamental functions of assembler

* Translating mnemonic opcode to m/c lang equivalances.
* Assign m/c addresses to symbolic labels use by the programmer

Assembler Directives / PSEDO INSTRUCTIONS

The assembli must also processes statemts called statemt directives These stmts are not translated into m/c instructions instead they provide instructions to the assembler itself.

eg: Declarations, START, BYTE, . . .

START: Specify the name and starting address for the progm

END: Indicate the end of the source pgm.

BYTE: Generate chara / hexadecimal constants

WORD Generate one word integer constant

RESERVE BYTE : Reserve the indicated no: of bytes for a data area

RESERVE WORD. Reserve the indicated no: of words for a data area.

SIC ASSEMBLER

1: Convert mnemonic opcode to their m/c lang equivalent

2: Convert Symbolic operands to their equivalent m/c address

3: Build m/c instructions in proper format.

4: Convert data constants specified in the source pgm into their internal m/c representations.

5: Write object pgm and assembly listing

format of object code : 3 parts.

- Header - ~~Pgm name~~
- Text
- End.

column.

H

Header → pgm name 1
Starting address 2-7
length 8-13
14-19

T

Text → Starting address 1
length of object code. 2-7
obj. code. 8-9
14-19

E

End → address in pgm where execution is to begin 1
2-7

eg: object code.

- H^COPY^001000^00107A
- T^001000^1E^141033
- E^001000

eg: JLT MOVECH when reached at MOVECH assemble keeps it as blank. Then reached at the MOVECH it will be filled

MOVECH

The problem with conversion of Symbolic operands to m/c address comes when there is a forward reference. A frwd reference is the reference to the label ie defined later in the pgm. Hence we use a two pass assembler. The first pass scans the source pgm for label definitions and assigns addresses. Second pass perform actual translation.

F()s of 2 pass assemble..

↳ Pass 1 (define symbols)

1: Assign address to all stmts in prgm
2: Save the address assigned to all label for use in pass two.
3: Perform some processing of assemble directives

↳ Pass 2 (assemble instructions & generate obj. programs

1: assemble instructions
2: Generate data values define by BYTE, WORD, ...
3: Perform processing of assemble directives not done in pass one
4: Write object program and assembly listing

Symb. label opcode
eg: MOVECH LDA ALPHA Symbol name

ALPHA WORD 1
ass. directive.

op code, ass. dire - predefined
Symb. name, label - defined by programmer.

opcode, ass. dire → ASSEMBLER → m/c lang equi
Predefined code will generat..

Symbolic-label, name → ASSEMBLER → m/y address

Then creates object code:

eg: LDA ALPHA → (obj code) 00 0303.

* Two types of assemblers
a) 1 pass assembler.
b) MULTI-PASS → 2 pass assembler.

main() what is the need of multi-pass?
{ int a;
int b = c; assigned first then declared. ∴ the pb
int c = 100 is forward reference.
a = b + c; }

- In 1-pass assembler forward referencing is not possible. So we use multi-pass assembler

Assembler data structures.

Two types of symbols for Assembler.

1) opcode — OPTAB
2) Symbolic names — SYMTAB.

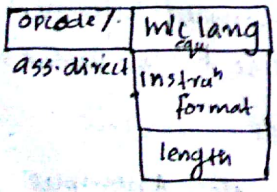
3 cl.

OPTAB: is used to lookup mnemonic opcodes and translates them to their m/c language equivalance.

SYMTAB: is used to store addresses assigned to label

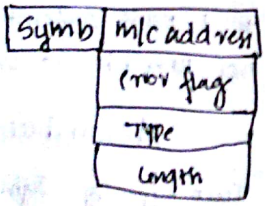
LOCCTR: (Location counter): is a variable is used to help the assignment of addresses.
LOCCTR is initialized to the begining address specified in the START STMT. After each source statement is processed the length of the assgd. instruct^{n} or data area is added to the LOCCTR. So, Everytime we reach a label in source pgm the current value of LOCCTR is the addr. of the label.

OPTAB



- Predefined
- Hash table.

SYMTAB



- Blank table. first
- Programmer defined data
- Hash table.

Pass 1:

In pass 1 checks whether opcode is in OPTAB

Pass 2:

In pass2 writes its equivalent machine language into object code.

SYMBTAB:

In pass Sym. name, and its names enter them in pass-2 it is used to Object code. machine address

TV

During pass-1 OBTAB is used to look up and validate operat^{n} code in the source pgm. In pass-2 it is used to translate the opcode to machine language

During pass-1 of assembler labels are entered into Symbtab as they come in the source pgm along with their assigned address from LOCCTR ~~Dur~~ During pass-2 Symbols used as operands are looked up in Symtab to obtain the address to be inserted in the assemble instructions.

Pass-1 writes an intermediate file. that contains each source stmt together with its assign indicators, error indicators etc.. This file is used to input to pass-2. This working copy of the source pgm can also be used to retain the results of certain operations that may be performed during pass-1 but these need not be pe" in Pass-2 If by pointers into OPTAB & SYMTAB may be retained for each opcode and symbol used.

Line	Loc	Source stmt	obj code
	000	LOOP CLEAR X	

label opcode operand

eg:

LOC			
000	START	0000	
	LDA	#5	
	STA	ALPHA	
	LDCH	#'90'	
	STCH	C2	

At First Pass 1..

- Location fills.
- opcode check whether it includes it or not
- Symtab ~~fill~~ Variable enters. (ALPHA).

- Implement assembler directives.
- After pass 1 there will be a intermediate file.

At Pass-2

- OPTAB - Translates to machine language
- SYMTAB - machine address

Fig 2.4 (a) pass-1 algorithm.
Fig 2.4 (b) pass-2

22/9/17

Machine Dep Assembler features.

op	m	Base/pc addressing mode. b/p=1
op	#c	Immediate. i=1 n=0
op	@m	indirect i=0 n=1
op	m,x	index addressing mode. x=1
+op	m	Extended " " e=1

→ Predefined members of Registers.

A	0
X	1
L	2
B	4
S	4
T	5
F	6
PC	8

n i x b p e
0 1 0 0 1 0
1 0 0 0 1 0

SM-9

Address Translation.

eg1

```
0000  FIRST  STL  RETADR   Program defined label
0003          OP   YARIABLE
 :
 :
 :
0030  RETADR  RESW 1
```

Opcode	n	i	x	b	p	e	displacement
	1	1	0	0	1	0	

displa = Target address − Pc. (address of next instruction)

TA = Pc + displa

0030 −

displacement = 0030 − 0003

= 002D. = 020 (hexadecimal)

0 2 1

opcode = 0001 01 00 — 0000 0010 1101

0001 0111 0010

eg2 Base Register

```
12    LDB  #LENGTH   (immediate)
13    BASE LENGTH
 :
100   0033 LENGTH RESW 1   (variable)
105   0036 BUFFER RES B 1
 :
160   104E STCH BUFFER,X   − 54.0036
      1051
```

STCH 54

line number

Opcode	n	i	x	b	p	e	displacement
	1	1	1	1	0	0	003

TA = B + disp

dis = TA − B

= 036 − 033. = 003

1 1 1 1 00 003

0101 01 0 0

0101 0111 1100 003

5 7 C

24/6/17

```
0006  CLOOP  +JSUB  RDREC
 :
0017  J      CLOOP   — Pc relative addressing mode.
001A  ENDFIL LDA  EOF
```

opcode − JSUB, J, LDA

(format: address Label opcode)

J	3C

opcode	n	i	x	b	p	e	disp

displacement = TA − PC

= 006 − 001A (2's complement)

0000 0006 0110

2' complement of

0000 0001 1010
1

0000
1111 1110 0110

disp = TA − PC
= 0006 − 01A
= 0000 0000 0110
1111 1110 0110 +
1111 1110 1100
F E C

Opcode	n	i	x	b	P	e	displacement
	1	1	0	0	1	0	FEC

0011 11 0 0
0011 1110 0010 FEC

- **Immediate**

Q1 0020 LDA #3
Q2 1030 +LDT #4096.

LDA 00

A:1 Extended and immediate addressing mode

Opcode	n	i	x	b	P	e	displ
0000 00	0	1	0	0	0	0	003

displacement = 3
= 003

0

Opcode = 0000 0000 0000 0000 0000 0011
0 1 0 0 0 3

Q:2 1030 +LDT #4096 LDT 74

A Extended and immediate addressing mode.

Target address = value at location.

opcode	n	i	x	b	P	e	displacement

displacement = 4096

0100 0000 1001 0110

Convert 4096 to hexadecimal.

16 | 4096	
16 | 256	0
16 | 16	0
1	0

$= (1000)_{16}$

$2^{12} = 4096$

$(100000000000)_2$

Since it is extended use format 4 therefore displacement have 5 bits.

opcode	n	i	x	b	P	e	displa

0111 0100 0 1 0 0 0 1 ←

0111 0101 0001 01000

- Immediate + Pc-relative

```
0003  LDB   #LENGTH.
0006
  :
0033  LENGTH  RESW 1
```

LDA	00
LDB	~~78~~ 68
LDT	6 74.

dis = TA − PC

= 033 − 006

=

mac dept

1) IF $AM (Instruct^{n} format and Addressing
2) Program Relocation.
   modification record

ALP using assembler.

```
0000   PROG  start 0      no opcode.
0000          LDS #3 -> 6D0003
0006          LDT # 30 -750018
0003          LDX # 0 - 050000
0006   LOOP LDA  ARR1,X-03A014
0009          ADD  ARR2,X
000C          STA  ARR3,X
000F          ADDR S,X - 9041. Registers.
0012          COMPR X,T - A015
0014          JLT  LOOP
0016          END  PROG
0019   ARR1   RESW 10
0020   ARR2   RESW 10
0040   ARR3   RESW 10.
0060
```

LDS	6C	JLT	38
LDA	00	ADD	18
LDT	74	ADDR	90
LDX	04	CMPR	A0
STA	0C		

- LDS #3

opcode	n	i	x	b	P	e	disp
6C		0	1	0	0	0	003

0110 11

0110 1101 0000 003

↓ ↓ ↓ ↓

6 D 0 003 0011

- LDT #30

opcode	n	i	x	b	P	e	displacement
74		0	1	0	0	0	1E

16 | 30 E

1

0111 0100

0111 0101 0000 1E

↓ ↓ ↓

7 5 6 1E

- LDA ARR1,X

opcode	n	i	x	b	P	e	disp
	1	1	1	0	0	0	014

disp = TA − PC

000

00C

0000 0000 1100

1111 1111 00

opcode: 0000 00

0016 JLT LOOP

displacement = 009 − 019

= FF0

opcode = 38 → 0011

13/9/17

FIRST LDS #3

opcode	n	i	x	b	P	e	displacement
		0	1	0	0	0	3

0110 1100

0110 1101 0000 0000 0000 0011

↓ ↓ ↓ ↓ ↓ ↓

6 C 0 0 0 3

8421

→ LDT #60

opcode	n	i	x	b	P	e	displacement
74		0	1	0	0	0	3C

0111 0100

0111 0101 0000 0000 0011 1101

↓ ↓ ↓ ↓ ↓ ↓

7 5 0 0 3 C

→ LDX #0

opcode	n	i	x	b	P	e	displace
04	0	1	0	0	0	0	0

0000 0100

0000 0101 0000 0000 0000 0000

0 5 0 0 0 0

→ STX SUM

displacement = TA − PC

= 0033 − 000C

0 0 3 3

0000 0000 0011 0011

2's complement of 000C:

0000 0000 0000 1100

0 0 0 C

0000 0000 0000 0011 + 1

0000 0000 0000 1101

1100

0011

0011 +

0

TA − PC = 033 − 00C

= 027

Opcode	n	i	x	b	P	e	displacement
10	0	0	0	0	0	0	027

0001 0000

~~12-37~~ · 13 2027

→ LOOP LDA ARR1,X − 03AFF1

ADD SUM − 1B 2021

STA SUM

14/9/16

LDT − 74 , LDA − 00 , ADDR − 90 , SUB − 1C

STA − 0C ,

→ LDT NUM

PC-addressing mode

opcode	n	i	x	b	P	e	displacement
74	1	1	0	0	1	0	

displacement − TA − PC = 0070 − 005B

0 0 5 B (2's complement)

0101 1011

1010 0101 2's — PC

1111 1111

TD → 0111 1101 = TA

1010 0101 +

0000 0010

0 2 0 2

Displacement = 022.

1 1 0 0 1 0

1) 1 1 0 2 2

2) LDA ALPHA

opcode for LDA = 00

displacement = TA - PC

= 0071 - 005E

0 0 5 E

0000 0101 1110

1111 1010 0010 2's complement

1111 1110 0000

F E 0

0000 0111 0001

1111 1010 0010

0000 0001 0011

0 1 3

opcode	n	i	x	b	P	e	displacement
00	1	1	0	0	1	0	0 1 3

1) 0 3 2 0 1 3

3) ADDR TA - 9050.

4) SUB #8

immediate addressing mode.

displacement = 8

1000

SUB = 1 C => 0001 1100

opcode	n	i	x	b	P	e	displacement
0 0	0	0	0	0	0		1000

0001 1100

1 D 0 0 8

5) STA SUM1

opcode = 0C

= 0000 1100

displacement = TA − PC

= 0074 − 0066

0000	0110	0110
1111	1001	1010

2' complement

0000	0111	0000
1111	1001	1010
0000	0000	1110
↓	↓	↓
0	0	E

```
1000    COPY    START   1000
1000    EOF     BYTE    C'EOF'
1003    THREE   WORD    3
1006    ZERO    WORD    0
1009    RETADR  RESW    1
100C    LENGTH  RESW    1
100F    BUFFER  RESB    4096
2005    FIRST   STL     RETADR
2012    CLOOP   JSUB    RDREC
201B            LDA     LENGTH
                COMP    ZERO
```

```
201B            JEQ     ENDFIL
201E            JSUB    WRREC
2021            J       CLOOP
2024    ENDFIL  LDA     EOF
2027            STA     BUFFER
202A            LDA     THREE
202D            STA     LENGTH
2030            JSUB    WRREC
2033            LDL     RETADR
2036            RSUB
```

JSUB − 48 , COMP − 28, JEQ = 30, J = 36

LDL − 08 RSUB − 4C STL − 14 , SUB − 1C

LDT − 74 LDA − 00 ADDR − 90 STA − 0C.

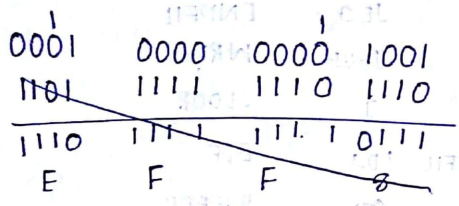
STL RETADR

Displacement = TA − PC

= 1009 − 2012

2012 = 0010 0000 0001 0010

1101 1111 1110 1110 2's



Displacement = TA
= 1009

opcode	n	i	x	b	P	e	displacement
	0	0	0	0	1	0	1009

0001 0100

0001 0100 0010 1009
↓ ↓
1 4

3rd module

16/9/17 Machine Independent features

1) Program blocks
2) Control sections.

- Any machine can use program block as well as control sections.

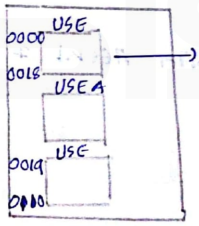
Program blocks

Segments of code that are re-arranged within a single object program blocks.

Assembler directives

START END
BYTE RESB
WORD RESW
BASE NO BASE
USE

- A program can be divided into different blocks, it is used for correct arrangment / rearrange the program.
- A block without assemble directive (USE name) then that block is default.



There is a LOCCTR for each block:

BLK name	BLK NO	SADAL	LENGTH
default	0	0000	0100
ABLCK	1	0103	
BBLCK	2		

Address	Block No	
0000	0	STL RETADR
0003	0	
⋮		
0000	1	RETADR RESW 1

Blk No	Starting addr

Control section refers to segments that are translated into independent object program units.

Address	Block no	
0006	0	LDA LENGTH
0009	0	
⋮		
0003	1	LENGTH RESW 1

Blk 0	0000
Blk 1	0066

opcode	n	i	x	b	p	e	displacement
00	1	1	0	0	1	0	

displacement = TA − PC

TA = TA + Starting add of Stack.

= 0003 + 0066

= 0069

displacement = 0069 − 0009

= 0060

0110 0110

0011

0110 1001

0011 0010 060

032060

STL	14

0000 0 STL RETADR

0003 6

⋮

0000 1 RETADR RESW 1

Blk 0 0000

Blk 1 0066

0001 0100

opcode	n	i	x	b	p	e	disp
	1	1	0	0	1	0	063

TA = TA + Starting of Stack

= 0000 + 0066

= 0066

disp = TA − PC = 0066 − 0003

= 0063

opcode	n	i	x	b	p	e	disp
00	1	1	0	0	1	0	063

0000 00 0 0

0000 00 1 1 00 10 063

(0 3 2 063)

0000 0000 0011
1111 1111 1100
1111 1111 1101

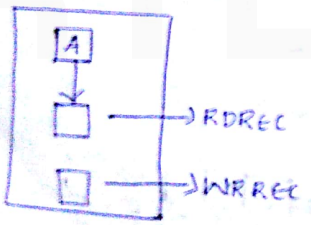
0000 0110 0110
1111 1111 1101

0000 0110 0011

CONTROL SECTIONS

Three data structures are used they are

1. Control section [CSECT]
2. External definition [EXTDEF]
3. External reference [EXTREF]



In control section records are

1) Header
2) Text
3) End

4) Modification record
5) Define record
6) Refer record

DEFINE RECORD

1 D
2-7 Name of External symbol defined in
8-13 Relative addr within this CS
14-73 Name and addr of other external symbol

eg. D^BUFFER^00033^BUFFEN D^001033

REFER RECORD

1 R
2-7 Name of external symbol referred in this CS
8-73 Name of other external ref symbol

eg R^RDREC, WRREC

MODIFICATION RECORD.

1 M
2-7 Starting address
8-9 Length of field to be modified
10 +/-
11-16 External symbol whose value is to be added/Substract

eg: M^000004^05^+RDREC.

- Difference blw Program blocks and control Sections
  - program blocks have a continuity
  - A control section is a logical unit of its own ie not continous.

Assembler directive of prgm block – USE

" " control section – CSECT

Object record

H_A

T_A

⋮

E_A 0000

→ Assembler directive of control section.

1. CSECT
2. (External definition) EXTDEF
3. (External Reference) EXTREF

Ex

```
EXTDEF  BUFFER  LENGTH
EXTREF  RDREC   WRREC
  ⋮
+JSUB   RDREC
```

Ex

```
RDREC   CSECT
-EXTREF BUFFER, LENGTH
  ⋮
+STX    LENGTH
```

In I section we define buffer and length.

eg: 0003 CLOOP +JSUB RDREC

```
EXTDEF  BUFFER, LENGTH
EXTREF  RDREC, WRREC
  ⋮
BUFFER  RESW 1
LENGTH  RES B 1
```

0003 CLOOP +JSUB RDREC

Displacement = 0

JSUB	4 B

Ex

Opcode	n	i	x	b	p	e	displacement
	1	1	0	0	0	1	00000

0100 1000

0100 1011 0001 0000

4 B 1 0000

eg: 0017 STCH BUFFER,X

STCH	54

displacement = 0.

opcode	n	i	x	b	p	e	displacement
0101	1	1	1	0	0	1	00000

0101 0100

0101 0111 1001 00000

5 7 9 00000

ASSEMBLER DESIGN OPTIONS

1) ONE PASS ASSEMBLER.

a) Load and Go.

Create opcode

```
0003   CLOOP   +JSUB   RDREC
  :
0014     J      CLOOP  ←
0017   ENDFIL
```

J	3C

Displacement = TA − PC

opcode	n	i	x	b	p	e	displacement
	1	1	0	0	1	0	

0011 11 11 0010 FEC

Object record.

```
COPY  START  1000
1000   ——    4B100000
       ——    3202016
       ——    945000
       ——    587FCC
```

H^ COPY^ 1000

T^ 4B10000^3202016^945000^587FCC

E^ 1000.

4/10/17 ASSEMBLER DESIGN OPTIONS

- Single Pass
  - Load & Go Assemblee
  - Ass. that produces int code.

eg:2 0017 +STCH BUFFER,X

STCH	54

opcode	n	i	x	b	p	e	displacement

→ LOAD AND GO ASSEMBLER :-

Forward referencing has to be handled well

mem addr	contents
2013	------
201A	100948
201D	001000
201F	------

Symb value

LENGTH	100C
RDREC	*1 → 2013 → 201F

100C LENGTH RESW 1
2013 +JSUB RDREC
⋮
201F +JSUB RDREC
⋮
⋮
9FF EXTDEF RDREC.

- RDREC is an external symbol it's address is unknown so we put address as 4B100000
  CONTENT
  4BL ----
- In the value column we create a linked list and save it where ever we use the RDREC
- When we reached at 9FF we fill the blank spaces with 9FF.

PN 92, 93, 95

5/10/17

0017 LDA = C'EOF'

0030 * = C'EOF'

Literal - without using variable directly equating

LDA BUFFER.

BUFFER RESB 'EOF'

- To create object code for literal.

LDA	00

op	n	i	x	b	p	e	displacement
00	1	1	0	0	1	0	016

disp = 030 - 01A
= 016

0030 * = C'EOF'

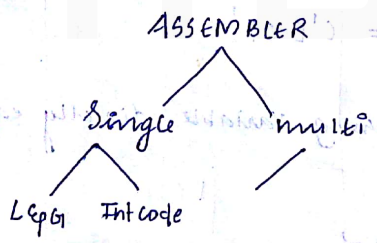
Here write the ascii value of E, O, F

Alphabet	ASCII
A	41
E	45
F	46
O	4F
Z	5A

16 | 4096 | 0
16 | 256 | 0
16 | 16 | 0
1

1000

0033 BUFFER RESB 4096
1033 BUFFEND EQU *
1000 MAXLEN EQU BUFFEND- BUFFER.



Multipass assembler

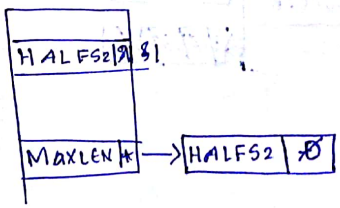
ALPHA EQU BETA
~~BETA RESW 1~~
BETA EQU DELTA
DELTA RESW 1

To fill symbol table we need to pass many time

Multipass assembler some statements can't be resolved by 2-pass assemblers. In multipass assembler we resolve forward reference with as many passes as needed. Multipass assembler use link list to keep track of whose value depends on on an undefined symbol

HALFS2 EQU MAXLEN/2
MAXLEN EQU BUFFEND- BUFFER
1033 PREVBT EQU BUFFER-1
⋮
1034 BUFFER RESB 4096
BUFFEND EQU * (literal)

~~At~~ In pass 1 we get value of buffer and Buffend. In next pass we get the value of maxlen and Prevbt. In 3rd pass we get HALFS2
We need 3 pass. Three.



In first length we get.

In source program * is literal but in symbol table * undefined

→ After second statement

HALFS2	&1	
BUFFER	*	→ MAXLEN
BUFFEND	*	→ MAXLEN
MAXLEN	$2	BUFFER-BUFFEND → HALFS2 | Ø

$1 and $2 are no: of variable depend on it.

→ After 3rd statement

HALFS2	$1	
BUFFER	*1034	
BUFFEND	*	→ MAXLEN
MAXLEN	$1	BUFFER-BUFFER → MAXLEN
PREVBT	$1	1033 → HALFS2 | Ø

HALFSZ	800
BUFFER	1034
BUFFEND	2034
MAXLEN	1000
PREVBT	$1 | 1033

Microprocessor A developed MASM

Macro assembler

- Software tool
- to assemble assembly language

Four segments

Code → CS

Data → DS, ES, FS, GS by default DS using ASSUME others can be used

Constant →

Stack → SS

Jump is used for skipping a set of stmts.

Two types of jump

1) Near
2) far

Near

JMP SHORT TARGET

- within same control section
- assembled using CS
- It occupies 2-3 Bytes
- with 128 Bytes jump is called near jump.

~~Far~~ long.

JMP FAR PTR TARGET.

- ~~within diff~~ different control sections
- assembled using any other segments
- It occupies 5 Bytes.
- More than 128 bytes

END