

MODULE-2

ASSEMBLERS

9/8/2017



KTU Students



KTU Students
Dedication to Success

- * Assemblers convert assembly language into machine language

Fundamental functions of assembler

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

Assembler Directives/PSEUDO INSTRUCTIONS

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

e.g: Declarations, START, BYTE, . . .

START: Specify the name and starting address for the program

END: Indicate the end of the source pgm.

BYTE: Generate char/ 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: Unlike object pgm and assembly list code

Format of object code: 3 parts.

- Header - Pgm name
- Text
- End

Header \rightarrow Pgm name

Starting address : 8-13
length : 14-19

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

End \rightarrow address in pgm where execution is to begin 2-7

eg: obj code : HCOPY length :
A 100-1000, 00107A
T 001000, 1E, 141D33
E 1001000



eg: JLT MOVECH when reached at MOVECH assemble keeps it as blank. Then reached for instruction 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 fwd reference is the reference to the label ie defined later in the pgm. Hence we use a two pass assemble.

The first pass scans the source pgm for label definitions and assigns addresses. Second pass perform actual translation.

FC's of 2 pass assemble.

Pass 1 (define symbols)

- 1: Assign address to all symbols in program
- 2: Save the address assigned to all labels for use in pass two.
- 3: Perform some processing of assembly directives.

Pass 2 (assemble instructions & generate obj. programs)

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

e.g.: MOVECH LDA ALPHA Symb.name

ALPHA WORD 1 ass. directive

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

op code, ass. dire → ASSEMBLER → Mc lang equiv

Symbolic-label, name → ASSEMBLER → my address

Then creates object code:

LDA ALPHA obj. code → 00000303 / 00000301

Registers to destination and op

* Two types of assemblers

a) 1 pass assembler

b) MULTI-PASS → 2 pass assembler

main() { int a;

int b = c; // forward reference. c is declared later

int c = 100; }

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) Op code → OBTAB

2) Symbolic names → SYMTAB

3.

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

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 beginning address specified in the Start Stmt. After each source statement is processed the length of the asse. instruction or data area is added to the LOCCTR. So, Even if we reach a label in source pgm the current value of LOCCTR is the addr. of the label.

OPTAB

opcode	m/c lang
ass. direct	instr ⁿ format
	length

SYMTAB

Symb	m/c address
	mnemonic flag
	Type
	length

- Predefined
- Hash table.

- Blank table: first
- Program's initial data
- Hash table.

Pass 1:

In pass 1 check whether opcode is in OPTAB

Pass 2:

In pass 2 writes its equivalent machine language into Object code.



SYMBTAB:

In pass 3 name, and its name enter then in pass-2 it is written to Objectcode.

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

During pass-1 of assemble Labels are entered into SYMBTAB as they come in the source pgm along with their assigned address from LOCCTR also During Pass-2 Symbols used as operands are looked up in SYMBTAB 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 & certain operations that may be performed during Pass-1. As there need not be pc" & Pass-2 by pointers into OPTAB & SYMTAB may be retained for each opcode and symbol used.

Line Loc Source Stmt Obj. Code.

000 Loop CLEAR X R-Address of input operand

label opcode operand of address with

eg: LOC
000 START 0000
LDA #5
STA ALPHA
LDCH #90
STCH



- Location fills
- opcode check whether it includes it or not
- symtab fill Variable Enter.(ALPHA).

- Implement assembly 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

Machine Dep Assembly features.

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

→ Predefined numbers of registers.

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

0	1	0	0	1	0
1	0	0	0	1	0

SWI-9

Address Translation:

eg1
 0000 FIRST STL RETADR Program defined label
 0003 ^{Program} _{defn} OP VARIABLE
 :
 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 instruc}

$$TA = PC + displa$$

$$\text{displacement} = 0030 - 0003$$

$$= 002D. = 020 \text{ (hexadecimal)}$$

$$\text{opcode} = \begin{array}{lll} 0001 & 01 & 00 \\ \hline 0001 & 0111 & 0010 \end{array}$$

0030-

Base Register:

12 LDB #LENGTH
 13 BASE LENGTH
 :
 100 0033 LENGTH RESW 1
 105 0036 BUFFER RESB 1
 :
 160 104E STCH BUFFER, Y -54.0036
 1051

STCH | 54

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

$$TA = B + \text{disp}$$

$$\text{dis} = TA - B$$

$$= 036 - 033 = 003$$

$$\begin{array}{llll} 1 & 1 & 1 & 1 \\ 0101 & 01 & 00 & \end{array} \quad \begin{array}{llll} 00 & 003 \\ 0000 & 0000 & 0000 & 0000 \end{array}$$

$$\begin{array}{llll} 0101 & 0111 & 1100 & 003 \\ 5 & 7 & C & \end{array}$$

2A18/17 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
	0	1	1	1	0	0	-54

$$\text{displacement} = TA - PC$$

opcode is first 3 bits

= 006 - 001A \rightarrow complement

$$0000 \quad 0000 \quad 0110 -$$

2' complement of

$$\begin{array}{r} 0000 \quad 0001 \quad 1010 \\ - 1111 \quad 1110 \quad 0110 \\ \hline 000 \end{array}$$

disp = TA - PC

$$= 0006 - 01A$$

$$\begin{array}{r} 0000 \quad 0000 \quad 0110 \\ - 1111 \quad 1110 \quad 0110 \\ \hline 1111 \quad 1110 \quad 1100 \end{array}$$

F E C

Opcode	n	i	x	b	P	e	displacement
0011	1	0	0	1	0	FEC	
0011 1110 0010						FEC	

• Immediate

Q1 0020 LDA #3

Q2 1030 +LDT #4096.

LDA | 00

A1 Extended and immediate addressing mode

Opcode	n	i	x	b	P	e	displacement
0000	0	0	1	0	0	0	003

displacement b = 3

$$= 003$$

0

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

Q2 1030 +LDT #4096 LDT | 74

4 Extended and immediate addressing mode

Target address = value at location.

Opcode	n	i	x	b	P	e	displacement
0100	0	0	0	0	0	0	4096

displacement = 4096

0100 0000 1001 0110

convert 4096 to hexadecimal.

$$= (1000)_16$$

16	4096
16	256
16	16
16	0

$$2^{12} = 4096$$

$$(100000000000)_2$$

Since it is extended use format & therefore displacement have 6 bits.

opcode	n	i	n	b	P	e	display	10000000
0111 0100	0	1	0	0	0	1	←	00000000

0111 0101 0001 01000 → 0000 - 00000000

- Immediate + PC - Relative

0003 - LDB # LENGTH.
 0006 :
 0033 LENGTH RESW !
 LDA 00
 LDB 68
 LDT 674.

$$\text{dis} = TA - PC \\ = 033 - 006$$

- modification record
- 1) IF \$AM (Instruction format and Addressing)
 - 2) Program Relocation.



ALP using assembler

```

    PROG Start 00000000
    LDS #3 → 6D0003
    LDT #30 - 75001E11 0000
    LDX #000 - 05000000 0000
    LOOP LDA ARR1,X - 03A014
    ADDS ARR2,X
    STA ARR3,X
    ADDR S,X - 9041, T015
    CMPR X,T - A015
    JLT LOOP
    END PROG 0010-1110
    ARR1 RESW 10 1010 1110
    ARR2 RESW 10 1110
    ARR3 RESW 10 0000 0000
  
```

LDS	66	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	displacement
6C	0	1	0	0	0	0	003

0110 1101000000003

↓ ↓ ↓ ↓ ↓ ↓
G D X 000003 0011

LDT #30

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

0111 0100 10001100

↓ ↓ ↓ ↓ ↓ ↓
+ 5 01 6 1E 00000000

LDA ARRI,X.

opcode	n	i	x	b	P	e	displacement
111000001014	0	1	0	0	0	0	TA - PC

[0000 0000 1100]
1111 1111 00

opcode: 0000 00

0016 JLT LOOP

$$\text{displacement} = 009 - 010 \\ = \underline{\underline{FF0}}$$

$$\text{opcode} = 38 \rightarrow 0011$$

13/14

FIRST LDS #3

opcode	n	i	x	b	P	e	displacement
01000000	0	1	0	0	0	0	3

D110 1100

0110 1101 0000 0000 0000 0011

↓ ↓ ↓ ↓ ↓ ↓
6 0 0 0 0 0

→ LDT #60 1100 1100 0000 0000

opcode	n	i	x	b	P	e	displacement
74	0	1	0	0	0	0	30

0111 0100 0000 0000

↓ ↓ ↓ ↓ ↓ ↓
+ 5 40 0 3 0 0

→ LDX #0

Opcode	n	i	x	b	P	e	displacement
04	0	1	0	0	0	0	0

0000 0100 0000 0000 0000 0000 0000 0000

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

→ STX SUM

displacement = TA - PC

$$= 0033 - 000C$$

0 0 3 3
0000 0000 0011 0011

2's complement of 0000C:

0000 0000 0000 1100
↓ ↓ ↓ ↓
0 0 0 C
0000 0000 0011 0011
0000 0000 0000 1100

$$TA - PC = 033 - 00C$$

$$= 027$$

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

0001 0000



12-27-13-2027

→ LOOP LDA ARR1,X - 03AFF1
ADD SUM - 1B 2021
STA SUM OF 201E

14/6/14 LDT-74, LDA-00, ADDR=90, SUB-7C

STA-0C,

→ LDT NUM

Pc-addressing mode

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

displacement = TA - PC = 007D - 005B

STA SUMI
 opcode = 0C
 = 0000 1100 1000 0000
 displacement = TA - PC
 = 0074 - 0066
 Displacement = 0000 0110 0110 0110
 1111 1001 1010 2's complement
 0000 0111 0000 1110
 1111 1001 1010
 0000 0000 1110
 0 0 E
 1000 COPY START' 1000
 1000 EOF BYTE C'EOF'
 1008 THREE WORD 3
 1006 ZERO WORD 0
 1009 RETADR RESW 1
 100C LENGTH RESW 1
 100F BUFFER RESB 4096
 200F FIRST STL RETADR
 JSOB RDREC
 LENGTH LDA LENGTH
 COMP ZERO
 2012
 2015

2018 JEO ENDFIL
 201E JSUB WRREC
 2021 J CLOOP
 2024 ENDFIL LDA EDF
 2027 STA BUFFER
 202A EDA THREE
 202D STA LENGTH
 2030 JSUB WRREC
 2033 LDL RETADR
 2036 RSUB ADDR-90, STA-OC.
 JSUB-48, COMP-28, JEQ=30, J=36
 LDL-08, RSUB-4C STL-14, SUB-1C
 LDT-14, LDA-00 ADDR-90, STA-OC.
 0010 0000 0001 0010
 1101 1111 1110 1100

STL READRR

$$\text{Displacement} = \text{TA} - \text{PC}$$

2012
 0010 0000 0001 0010
 1101 1111 1110 1100

11111111 101
 00000000 1001
 11111110 1110
 E F F 8

$$\begin{aligned} \text{Displacement} &= TA \\ &= 1009 \end{aligned}$$

opcode	n	i	x	b	P	e	displacement
0001	0	0	0	1	0	1009	

0001 0100

0001 0100 = 0010 + 1009
 ↓ ↓
 1 40 DATA 34 module

16/9/17 Machine Independent features

- 1) Program blocks
- 2) Control sections.

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

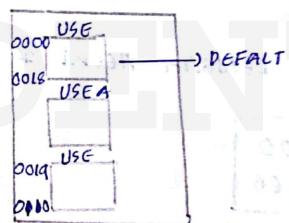
Program blocks

Segments of code which are rearranged 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 arrangement & rearrange the program.
- A block without assemble directive (USE) then that block is default.



There is a LOCCTR. for each block.

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

Address	Block No	Op	Op	Op	Op	Op
0000	0	STL	RETADR			
0003	0					
:						
0000	1	RETADR	RESW	1		

Block No	Starting add.
0	0000
1	0066

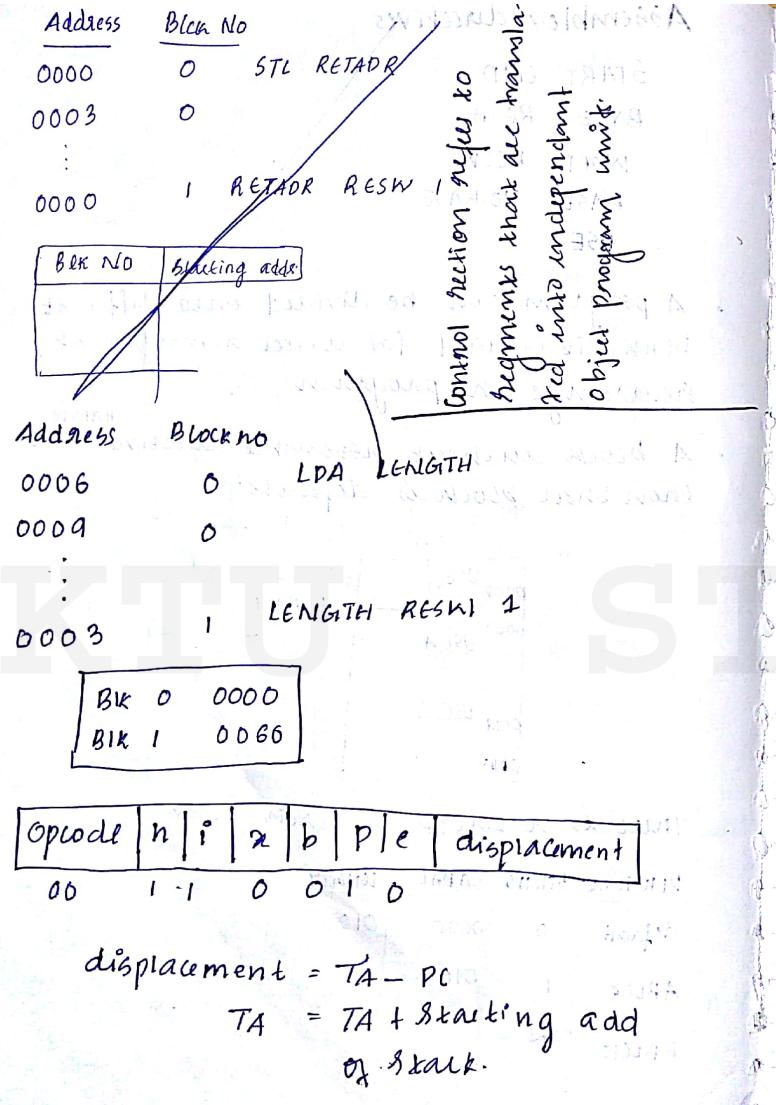
Address	Block no	Op	Op	Op	Op	Op
0006	0	LDA	LENGTH			
0009	0					
:						
0003	1	LENGTH	RESW	1		

Blk 0	0000
Blk 1	0066

Opcode	n	i	x	b	P	e	displacement
00	1	1	0	0	1	0	

$$\text{displacement} = TA - PC$$

$$TA = TA + \text{starting add of stack}$$



$$= 0003 + 0066$$

$$= 0069$$

$$\text{displacement} = 0069 - 0009$$

$$= 0060$$

$$0011 0010 060$$

$$032060$$

STL14

0000 0 STL RETADR
0003 6

0000 RETADR RESW 1

Blk 0 0000
Blk 1 0066

0001 0100

opcode | n | i | x | b | P | e | dis
1 | 1 | 0 | 0 | 1 | 0 | 0 | 066

$$TA = TA + \text{starting of stack}$$

$$= 0000 + 0066$$

$$= 0066$$

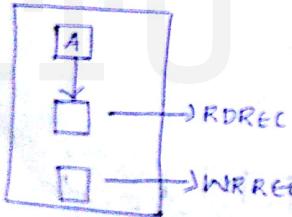
$$\text{disp} = TA - PC = 0066 - 0003 = 0063$$

opcode	n	i	x	b	P	e	disp	0000 0000 0011	1111 1111 1101	1111 1111 1101
00	11	00	10	063						
0000 00 00										
0000 00 11	0010	063	0000	0110	0110					
(0 3, 2 063)				1111	1111	1101				
				0000	0110	0011				

CONTROL SECTIONS

Three data structures are used they are

1. Control Section [SECT]
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

14-15 Name and addrs of other external
Symbol.

eg: Dn BUFFERn00033nBUFFEN Dn 001033

REFER RECORD

1 R

2-7 Name of internal symbol referred in this

9-13 Name of other internal set symbol

eg RnRDREC, 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 modified

eg: Mn 000004 n05 n+RDREC.



- Difference b/w Program blocks and control sections
- Program blocks have a continuity.
- A control section is a logical unit of its own it is not continuous.

Assembly directive of program block - USE

1. "Control section - CSECT"

Object record

H₁

T₁

⋮

En 0000



→ Assembly directive of Control section.

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

EXTDEF	BUFFER	LENGTH
EXTREF	RDREC	WRREC
⋮		
+JSUB	RDREC	

RDREC (CSECT)
 - EXTDEF BUFFER, LENGTH
 :
 +STX LENGTH

In I section we define buffer and length.

e.g: 0003 CLOOP +JSUB RDREC

EXTDEF BUFFER, LENGTH
 EXTREF RDREC, WRREC

⋮
 : BUFFER LENGTH RESW 1
 : RESB 1
 0003 CLOOP +JSUB RDREC

Displacement = 0

Opcode | n | ix | b | P | c | displacement

0 0 0 0 1 00000

0100 1000

0100 1000 0001 0000
 1 000.00

4/2 0017 STCH BUFFERIX
displacement = 0.

opcode	n	i	r	b	p	e	displacement
STCH	1	1	0	0	1		00000
0101	0100						

0101	DIII	1001	00000
5	7	9	00000

4/3 ASSEMBLER DESIGN

OPTIONS BUILT 4 01010 8000

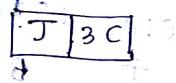
1) ONE PASS ASSEMBLER:

a) Load and Go.

Greater Opcode

0003	CLOOP	+ JSUB	RDRCL
:			
0014	J000	CLOOP	5
0017	ENDFL		

Displacement = TA - PC



opcode | n | i | r | b | p | e | displacement

1 1 0 0 1 0
0011 1111 0010 FCC

Object code:

COPY START 1000
1000 — 4B160000
— 3202016
— 945000
— 587 FCC

H COPY 1000

T A 4B10000 A 3202016 A 945000 A 587 FCC

E 1000.

4/10/17 ASSEMBLER DESIGN OPTIONS

Single Pass

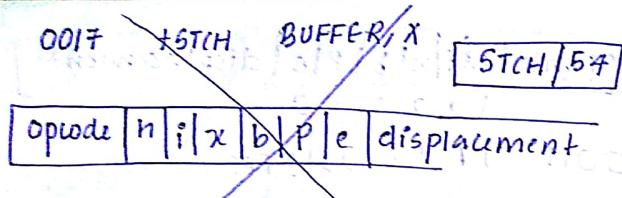
load & Go

Assembly

Ass. that produces link

Code.

e.g.2



→ LOAD AND GO ASSEMBLER :-

Forward referencing has to be handled well

mem addrs	contents
2013	-----
201A	100948
201D	00100C
201F	-----

100C LENGTH REW 1
2013 +JSUB RDREC

201F +JSUB RDREC

FFF EXTDEF RDREC.

Symb Value

LENSE	100C
RDREC	*!

- RDREC is an external symbol
its 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 * we fill the blank spaces with FFF.

PN 92, 93, 95

001F LDA = 'EOF'
0030 * = 'EOF'



Literal - without using variable directly equating

LDA BUFFER.

BUFFER RESB 'EOF'

- To create object code for Literal.

Op	n	i	z	b	p	e	displacement
----	---	---	---	---	---	---	--------------

00 10 00 10 016
disp = 030 - 01A
= 016

LDA 00

0030 * = 'EOF' (maximum value of DATA)
Here write the ASCII value of E, O, F

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

0033 BUFFER RESB 4096

1033 BUFFERID EQU *

1000 MAXLEN EQU BUFFEREND - BUFFER.

ASSEMBLER

Single pass
Multipass - Indirect
LEGI Int code

Multipass Assemblers

ALPHA EQU BETA

BETA RES1

BETA EQU DELTA

DELTA RES1

To fill symbol table we need to pass many times

Multipass assembler Some statements can't be resolved by 2-pass assembler. 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 an undefined symbol.

HALFS2 EQU MAXLEN/2

MAXLEN EQU BUFFEREND - BUFFER

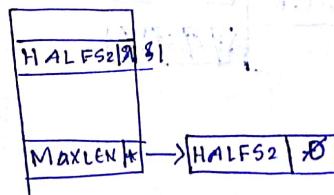
PREVBT EQU BUFFER-1

:

1034 BUFFER RESB 4096

BUFFEND EQU * (literal)

After 1st pass 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 passes. Every pass it will update link list.



In first length we get.

In source program it is literal but in symbol table it's undefined

→ After 1st & 2nd Statement

HALFS2	\$1	HALFS2	\$1
BUFFER	*1034	MAXLEN	
BUFFEND	*1034	MAXLEN	
MAXLEN	\$2	BUFFER = BUFFER	HALFS2 \$0

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

→ After 3rd statement

HALFS2	\$1	HALFS2	\$1
BUFFER	*1034	MAXLEN	
BUFFEND	*1034	MAXLEN	
MAXLEN	\$1	BUFFER = BUFFER	MAXLEN
PREVBT	\$1	1033	HALFS2 \$0

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



Microprocessor ~~A~~ developed MASM

- 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 stmt.

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

68011 13 18 A3 39

