Lecture Notes Contd. (Dated:- 24-2-22)

Program Relocation (supported by SIC/XE)

- SIC/XE has more memory ⇒ so now we can have multiple programs with time-shared execution

- In the SIC pgm, starting address = 1000

1000	
101B	001020
1020	000023

⇒ will work provided program loads from 1000.

- Suppose loader decides to load it from 3000.

3000	
3006	1

⇒ WILL BE CHAOS

- For SIC/XE

They're all OFFSETS & must be calculated from starting address.

0000		1000		3000		SAME AS ITS MENTIONING THE OFFSET
000A	032026	100A	032026	300A	032026	
0033		1033		3033		
LENGTH		LENGTH		LENGTH		

⇒ Problems

(1) Format- 4 instructions ⇒ you're hardcoding the address.

- Solution ⇒ Some kind of modification for the address field of f4

- Communication betn assembler & loader ⇒ the object code ⇒ the header record, text record etc

- Now, modification record will be introduced

⇒ Modification record

Col 1 : M

Col 2-7 : starting location of the address field to be modified relative to the beginning of the program

Col 8-9 : length of the address field to be modified in half bytes

⇓

20 bits address field

↳ can't be expressed in bytes

↳ so 4-bits

⇒ 5 4-bit fields

5 half bits.

eg :- M00000705

↳ for :- 0006 CLOOP +JSUB RDREC 4B101036

8 bits ⇒ 0006 0007 ⇒ Address FIELD ⇒ 4bit of address STARTS FROM HERE!!

OPCODE	n	i	x	b	p	e	Address
6							20

0014

M^000014^05

M^000027^05

as its not a symbol.

⇒ +LDT #4096 ⇒ modification record is not required.

⇒ Lecture Notes Contd. (Dated :- 25-2-22)

⇒ NOBASE ⇒ directive ⇒ don't use base register from programming after this.

So if PC is not possible, after this ⇒ error

⇒ Machine-Independent Assembler Features

① Literal

- Constant usage
- fixed value

ENDFIL	LDA	EOF
⋮		
EOF	BYTE	C'EOF'
⇓		
ENDFIL	LDA	=C'EOF'

tells that its a symbol

fixed value is directly given in instruction itself

- How to process the literals?
- as soon as assembler encounters this literal = C'EOF' it will put it in that literal pool.

⇒ All the literals gathered by assembler = literal pool
↓
using a LITTAB

⇒ LITTAB = Hash table
→ Literal Name, Operand value, length

⇒ Literal pool ⇒ at the end of the program usually
⇒ LTORG ⇒ you can keep the literal operand close to the instruction
⇒ #3 & LTORG = =C 'EOF' ⇒ not same as mem. is not assigned for #3

⇒ Need for LTORG ⇒ if placed at the end ⇒ a large buffer in between ⇒ high disp
↳ PC relative might fail

⇒ Pass ①
- =C 'EOF'
  ↳ pushed onto LITTAB

- LTORG, end of pgm
  ↳ Scans the LITTAB
    ↳ for each entry in LITTAB, assign the address + increment the LOCCTR

⇒ Pass 2

- Object code for an instruction

```
  1 1
  002D        FFFF
+ FFF3       - 001D
-------      ------
  0020        FFE2
```

literals ⇒ treated as symbols itself ⇒ have their own addresses.

⇒ ② Symbol defining statement

- Symbol (label) given to an instruction.
  eg :- CLOOP
- Data

⇒ Creating symbols using assembler directive = EQU
equate

Syntax :- Symbol EQU value

⇒ * ⇒ present value of LOCCTR

MAXLEN EQU BUFEND - BUFFER.
⇓
So length is not hardcoded now

- also used for :- A EQU 0
  X EQU 1

⇒ ③ Expressions

- literals, operands
  −, *, +, /
  ↓
  present value of LOCCTR

Date
Page

⇒ Lecture Notes Contd (Dated :- 2-3-22)

⇒ Machine independent assembler features

① Literals
② Symbol defining statements
③ Expression

④ Program blocks

⇒ So far, entire program was treated as a single entity.

⇒ Instructions & data can be made to appear in different order.

- Rearranged ⇒ object program & memory loaded version

⇒ With program blocks, we can change this order
↓
Using an assembler directive : USE

⇒ Assembler directive : USE

- 3 program blocks

① Unnamed block ⇒ all executable instructions
eg :- USE

② CDATA ⇒ all data that are a few words long or less in length

③ CBLKS ⇒ all data areas consisting of larger blocks of memory.

Using USE again ⇒ changing program block to default one.

- Why do we do this?

→ It makes addressing simple

→ It reduces the no. of F4 instructions ⇒ since the buffers was pushed to one end.

↳ everything can be done using only PC relative F4

- Drawback ⇒ additional info about the blocks should be ~~mentioned~~ maintained

⇓

block table

⇒ Block Table

Block name	Block number	Start Address	length
Default	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

⇓ 4096 bytes

⇒ Pass 1

⇒ Pass 2

- Symbol → Address from SYMTAB

+

Starting address from block

⇒ STL RETADR

[TA] = 0000 + 0066 = 0066

[PC] = 0003

= 0066

− 0003

0063 → disp.

Date
Page

(2) LDA LENGTH
[TA] = 0003 + 0066
= 0069
[PC] = 0009
$\boxed{disp = 0060}$

(3) JSUB WRREC
[TA] = 004D
[PC] = 0012

(4) JEQ RLOOP
[TA] = 0031
[PC] = 0037
⇒ disp = −006 = FFA

⇒ Lecture Notes Contd. (Dated :- 4-3-22)

(5) Control Sections & program linking

⇒ Control sections ⇒ part of the program that maintains its identity after assembly.

⇒ Each control section can be loaded and relocated independently

(1) Define record
Col 1 : D
Col 2-7 : Name of the external symbol defined in this control section.
Col 8-13 : Relative address of symbol within this control section
Col 14-73 : Repeats information in Col 2-13 for other external symbols

② Refer record

Col 1 : R

Col 2 – 7 → Name of the external symbol referred to in this control section

Col 8 – 73 ⇒ names of other external symbols

③ Modification record (modified)

Col 1 : M

Col 2 – 7 : starting address of the address field to be modified relative to the beginning of the control section (in hexadecimal)

Col 8 – 9 : length of the address field to be modified in half bytes

Col 10 : modification flag (+ or –)

Col 11 – 16 : External symbol whose value is to be added or subtracted from the indicated field