Unit-2

→ If literal is operand then fetch address
→ " " " opcode fetch value

→ In intermediate file

LTORG

LOCCTR	label	opcode
address	*	=C'EOF'

→ If LTORG is not there it is defined at end
→ LTORG has no machine code

Generate literal table:

Literal Name	Literal value	Literal length	Address
=C'EOF'	454F46	3	02D

get

→ In pass1 the literals defined (that are used in program) and addressed are assigned
→ In pass2 machine code is generated using symbol table and literal table

Pass I outputs:-

1) Intermediate file
2) Symbol table
3) Literal table
4) Program length

EQU - symbol defining statement

EQU is used to assign ~~add~~ or equate operand value to labels

EQU → constant (1000)
EQU → special symbol (*)
EQU → expression (BUFFEND - BUFFER)

→ * - assign locctr to label
→ Evaluate expression & assign it's output as current EQU statement address
→ if constant then constant value is address of instruction.

Program Block

→ Segments of a code that are rearranged within a single object program unit.

→ USE - define blocks

→ Include block number in intermediate file

USE → 0000

first state → 0000

Pass1

→

Block name	Block No	Address (A+length)	length (0-esht)
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

↓

1071 - program length

disp = ta - pc

(label address from intermediate file + Starting address of the block of label) - PC

Pass1 o/p:

1) Intermediate file (with block no)
2) Literal table
3) Block table
4) Program length
5) Symbol table (with block num)

To avoid format four instructions and to make it fall under relative addressing we use programs.

```
S1          1) Program blocks reduces the disp-
S2          lacement & fits within PC
S3                  relative
+JSUB   RDREC       /*
                    displacement cross base
        RESB  1     relative bound so is in
        RESB  10    format 4 can be avoided
        RESB  4096  using program blocks
          :         So larger memory occupying
          :         statements are put at end */
RDREC
```

2) Increases program readability

Control Sections:

1) each subroutine has independently assembled loaded & relocated & linked.

Assembler Directive : CSECT

secname CSECT

Separate loctr for each section

EXTDEF (Symbols defined in current CS and referred in other CS)

EXTREF (labels refurrend in current CS and defined in other CS)

Textbook :-

Program Blocks

→ feature of assembles that provides flexible handling of source and object programs
→ PB's are segments of code that are rearranged within a single program unit
→ This feature allows the generated machine instructions and data to appear in objed program in a different order from the corresponding source statements
→ It creates independent parts of objed program.
→ It contains several separate segments of source program
→ The assembles will rearrange the segments to gather together the pieces of each block
→ USE assembler directive (in objed program)
→ Blocks appear in same order in which they appear in soura program. -or- in some order in which they first begun in source program

Implementation:

→ a separate LOCCTR for each block
→ In pass1 each label in the program is assigned an address that is relative to the start of the block that contains it
→ In symbol tabel entry label + blockno + relative address

→ After pass1, latest value of LOCCTR for each block indicates length of that block

→ Assign starting address to each block in object prog (beginning with relative location 0)

* If the label is relative then no block numbe

Why relative?

1) Efficient machine sharing
2) for multi-programming.
3) User need not specify or know the location to load program.
4) It leads to efficient use of subroutine libraries
5) Efficient use of memory.
6) Many libraries have large no. of subroutines than those being used by program so we need to load exactly those routines that are needed. This can't be done effectively if all subroutines had preassigned addresses.

Relocation → machine dependent
Linking → " independent

→ Relocation
loader that allows program relocation is called relocation loader

→ Two methods to specify relocation as part of object program.
1) Modification record in SIC/XE (not immediate)
format 4 → M^address+1 ^05 + progname

Not suited for SIC as it uses direct addressing in both of it's format so all object code needs a corresponding M record which is not efficient

So all instructions except RSUB, WORD, BYTE

So for machines with fixed instruction format & direct addressing we go for bit-mask

No M records. T record is same but bit mask is added. A relocation bit is associated with each word of object code i.e a bit is "0" "1" instruction

Relocation bits are gathered together to form bit-mask

T^ SA ^ Length ^ Bitmask ^

3

Assemblee design options

→ One pass, Two pass, Mutlipass

↓ ↓ ↓

No forward reference is handled

handles forward reference

handles forward ref during symbol def's

→ One pass

Easy to remove f.r. from data items/ operands but not from label to instructions.

Types →

↙

Produce code directly in memory for immediate execution

Ex: Load-and-go

Suited for program development & testing.

→ The actual address to be loaded must be known

produce object program for later execution.

Suitable when external working-storage devices aren't available.

External storage is slow/inconvenient to use

Load and go - assembler

↳ overhead of one pass & writing object program and reading it everytime it is assembled.

Services of loader completes the forward references

DS for linking loader

→ In SIC/XE (∵ more are disp)

→ Why 2-pass?

EXTREF - (until address is assigned to ext symbol)

→ i/p - Object program set of diff CSECT's.

→ Pass1 → assign addresses to external symbol

→ Pass2 → loading, relocating, linking.

→ Pass1 implementation

→ DS used is ESTAB

↳ name address length of external symbol & CS's. → hashed table }

→ PROGADDR → starting memory address where linked program is to be loaded.

CSADDR - starting address assigned to CS currently being scanned by the loader.

Theory → SIC/XE - Relocation & modification
→ SIC ALP for bootstrap loader.
→ linking loader pass1/2 (block diagram)
→ One pass working
→ absolute loader

→ ESTAB add first CS name CSADDR and add all labels from D record with address as specified + CSADDR
→ when end record is read, CSLTH is update with program length that was retained from header record
→ Now CSLTH + CSADDR is added to give
→ starting address of next CS in input file
→ Pass2 implementation
→ CSADDR = PROGADDR // start address
→ EXECADDR = PROGADDR // start exec at start addr
→ Header → CSLTH = section length // from header
→ Read text record → do conversion (ASCI - hex digit)
load object code at specified + CSADDR current value
→ M records are read and values/addresses of external variables are found in ESTAB and added/subtracted from indicated location in memory + CSADDR
→ E record specifies address of first executable instruction so go there & begin execution (EXECADDR spe + CSADDR)

Theory

1) Algorithm for an absolute loader

→
```
begin
    read Header record
    verify program name and length
    read first text Record
    while record type ≠ 'E' do
    begin
        {if object code is in character form, convert
          to internal representation}
        move object code to specified location in memory
        read next object program record
    end
    jump to address specified in End record
end.
```

2) Bootstrap

```
begin
    X = 0X80
loop
    A ← GETC (read one ½ byte & convert to ASCII)
    Save value in S
    Shift S by 4 times
    A ← GETC
    add S and A to combine digits to form
    one byte    A ← A+S
    X ← X+1
end

GETC
    A ← read one character from F1 device
    if A = 0X04 then jump to 0X80
    if A < 48 then GETC
    A ← A-48 (0X30)
    if A < 10 then return
    A ← A-7
    return
```

ESA Paper

2a)

loc	label	opcode	operand	Objectcode
0500	SUM	START	500	-
0500	FIRST	LDX	#0	050000
0503		LDA	#0	010000
0506		LDS	#3	6D0003
0509		LDT	#1500	750 5DC
050C		LDA	=C'ABC'	032018
050F		+LDB	#TABLE2	69 100 1601
~~0513~~		BASE	TALBE2	-
0513	LOOP	ADD	TABLE, X	1BA017
0516		ADD	TABE2, X	1BB000
0519		ADDR	S, X	904 1
051B		COMPR	X, T	A015
051D		JLT	LOOP	3B2FF3
0520		+STA	TOTAL	0F10 169D
0524		RSUB		4C0000
0527		LTORG		-
052A	COUNT	=C'ABC'		-
052D	TABLE	RESW	1	-
1601	TABLE2	RESW	1500	-
169D	TOTAL	RESW	500	-
-		RESW	1	-
		END	FIRST	

SYMTAB.

Symbol	Value / Address.
FIRST	0500
LOOP	0513
COUNT	052A
TABLE	052D
TABLE2	16C1
TOTAL	169D

Program Length = 16A0.

LITTAB.

Name	=C'ABC'
Value	414243
Length	3
Address	0527

Object program.

H^ SUM ^000500^000536

T^ 0000500^ 13^ 050000^ 010000^ 6D0003^ 7505D

^ 032018 ^ 69100530

T^ 000513^ 1H^ 1BA017^ 1BA017^ 9041^ A015^

3B2FF3^ 0F100533^ 4C0000

E^ 000050

3a) Object code

Loc	Block	label	Opcode	operand	object code
0000		COPY	START	0000	-
0000	0	FIRST	STL	RETADDR	17 2 01F
0003	0	LOOP	JSUB	RDREC	4B 2 00D
0006	0		STCH	BUFFER,X	57A01D
0009	0		+COMP	#MAXLEN	291 01000
000D	0		JLT	LOOP	3B2FF3
0010	0		J	@RETADDR	3E200F
[redacted]	1		USE	CDATA	-
0000	1	RETADDR	RESW	1	-
	2		USE	CBLKS	-
0000	2	BUFFER	RESB	4096	-
1000	2	BUFFEND	EQU	*	-
1000	[redacted]	MAXLEN	EQU	BUFFEN-BUFFER	-
	0		USE		-
0013	0	RDREC	LDX	#0	050000
0016	0	RLOOP	TD	INPUT	E32006
0019	0		JEQ	RLOOP	332FFA
001C	0		RD	INPUT	DB2006
001F	0		RSUB		4C0000
[redacted]			USE	CDATA	-
0003	1	INPUT	BYTE	X'F1'	F1

Block Table

Name	Number	Address	Length
(default)	0	0000	0022
CDATA	1	0022	0004
CBLKS	2	0026	1000

→ Program length = 00267 1000 = 1026

SYMTAB

Label	value/address	Block NO
FIRST	0000	0
COOP	0003	0
RETADDR	0000	1
BUFFER	0000	2
BUFFEND	1000	2
MAXLEN	1000	-
RDREC	0013	0
RLOOP	0016	0
INPUT	0003	1

6(c) PROGADDR = 5000H.
CSADDR = 5000H.
ESTAB

Name	Address	Length
PROGA	5000H	0063H
LISTA	5040H.	
ENDA	5054H.	
PROGB.	5063H	007FH.
LISTB	50C3H	
ENDB	50D3H	

Before link

Memory structure

	Relocated			
5054	000014		50D3	000000
5057	FFFFF6		50D6	FFFFFC
505A	00003F		50D9	FFFFFF
505D	000014		50DC	FFFFF0
5060	FFFFC0		50DF	000060

PROGA

PROGB.

LOC	Label	opcode	operand	object code
0000	SWAP	START	0000	-
0000		LDX	ZERO	07#1000
0003		+LDB	#ZERO	69101F62
-		BASE	ZERO	-
0007		LDT	#4000	750FA0
000A	BACK	LDA	ALPHA, X	03A016
000D		STA	TEMP	0F4003
0010		LDA	BETA, X	03AFAF
0013		STA	ALPHA, X	0FA00C
0016		+LDA	TEMP	03101F65
001A		STA	BETA, X	0FA FA5
001D		TIXR	T	B850
001F		JLT	BACK	3B2FE8
0022	ALPHA	RESB	4000	-
0FC2	BETA	RESB	4000	-
1F62	ZERO	WORD	0	000000
1F65	TEMP	RESW	1	-
-		END.		

Program length = 1F65 + 3 = 1F68

Symbtab.

label	Address
SWAP	0000
BACK	000A.
ALPHA	0022
BETA	0FC2
ZERO	1FG2
TEMP	1FG5

Loc	Blo	Label	opcode	operand	OC
0000	0	COPY	START	0000	-
0000	0	FIRST	STL	RETADDR	172 01F
0003	0	LOOP	JSUB	RDREC	4B2 00D
0006	0		STCH	BUFFER,X	57A01D
0009	0		+COMP	#MAXLEN	29101000
000D	0		JLT	LOOP	3B2FF3
0010	0		J	@RETADR	3E200F
0013	0		USE	CDATA	-
0000	1	RETADDR	RESW	1	-
			USE	CBLKS	-
0000	2	BUFFER	RESB	4096	-
1000	2	BUFFEND	EQU	*	-
1000	0	MAXLEN	EQU	BUFFEND-BUFFER	-
			USE		-
0013	0	RDREC	LDX	#0	050000
0016	0	RLOOP	TD	INPUT	E32006
0019	0		JEQ	RLOOP	332FFA
001C	0		RD	INPUT	DB2006
001F	0		RSUB		4C0000
			USE	CDATA	-
0003	1	INPUT	BYTE	X'F1'	F1

0025 - 001F

Block Table

Name	No	SA	Length.
(Default)	0	0000	0022
CDATA	1	0022	0004
CBLKS	2	0026	1000

1026.

SS lab

* linking loader pass1

Datastructures:

1. ESTAB - symbol table for external references.

→ structure:

csname symbolName Address length

→ This tables contains labels and address of each external symbol in set of control sections being loaded.

→ Hashed Organization.

2. PROGADDR - beginning address in memory where the linked program is to be loaded.

→ This value is supplied by OS to loader

3. CSADDR - starting address assigned to CS currently being scanned by loader

→ CSADDR is added to all relative address in CS to convert to actual address

4 CSLTH → length of cs that is currently being scanned by loader.

→ CSLTH + CSADDR gives starting addrss of the next CS.

Algorithm: Pass1.

```
begin
    get PROGADDR
    set CSADDR to PROGADDR
    while (not end of file) input:
      do
        begin
          Read next input record {HR for CS}
          set CSLTH to CS length
          search ESTAB for CSNAME.
          if found then
              error
          else
            enter csname to estab with CSADDR.
          while recordtype ≠ 'E' do:
            begin
              read next input record.
                if recordtype == 'D' then
                  for each symbol in record do
                    begin
                      search ESTAB for symbolname
                      if found
                          error
```

else
enter the symbol name to ESTAB
with (CSADDR + specified address)
end {for}
end {while ≠ 'E'}
add CSLTH to CSADDR
end {while not EOF}
end {Pass 1}

* Pass 1

1) OPTAB
→ Used to lookup mnemonic operation codes and to translate them to their machine language equivalents. Hashed.

2) SYMTAB
→ stores labels & their corresponding addresses
Hashed (flags - indicate error cond'n)

3) LOCCTR
→ variable that helps in address assignment
→ initialized to address in START statement
→ increment based on length of instruction