ST. JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CST 305 – System Software

Prof. Sarju S

26 November 2023

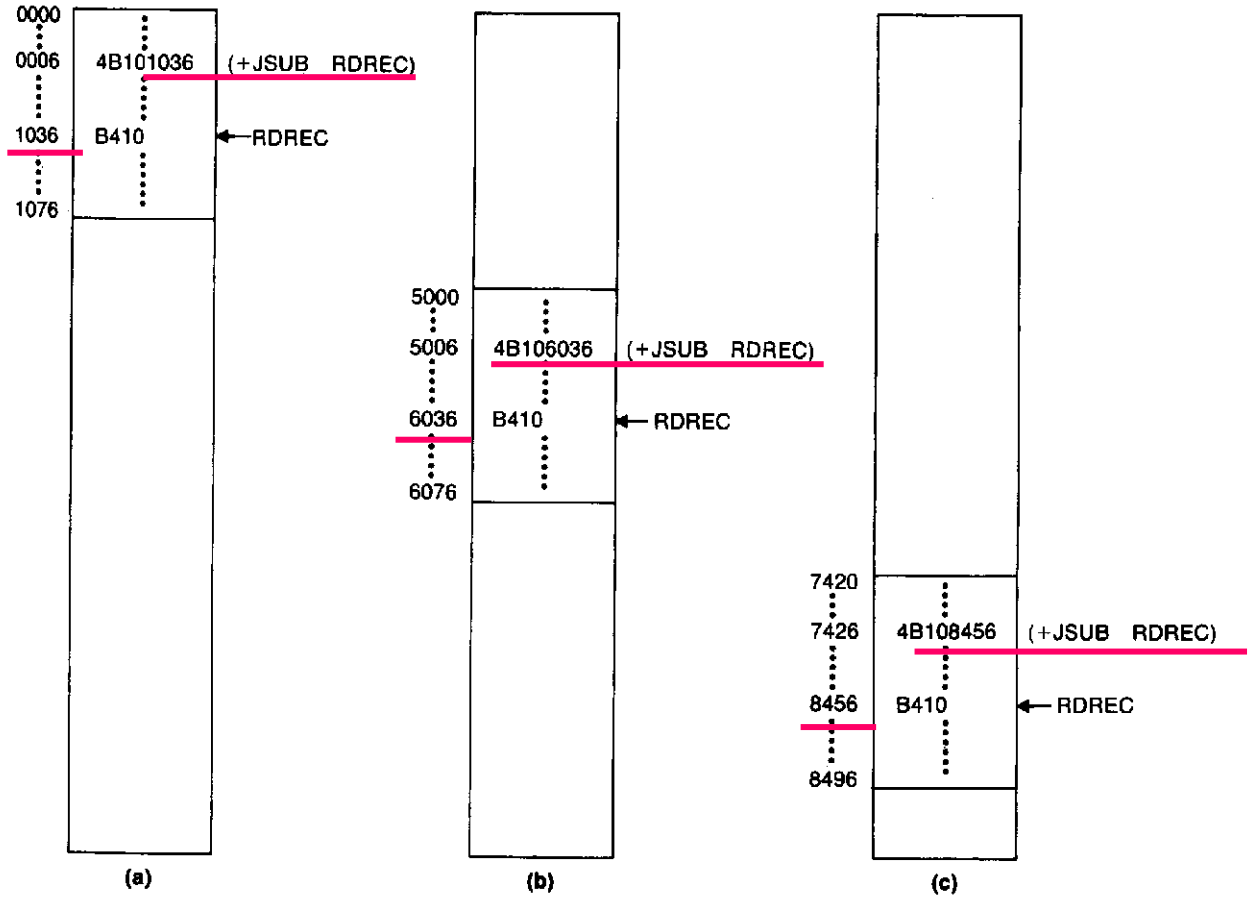
Module 3

- Machine Dependent Assembler Features-Instruction Format and Addressing Modes, Program Relocation. Machine Independent Assembler Features –Literals, Symbol Defining Statements, Expressions, Program Blocks, Control Sections and Program Linking. Assembler Design Options- One Pass Assembler, Multi Pass Assembler. Implementation Example-MASM Assembler.

Program Relocation

Program Relocation

- Absolute program, relocatable program



Program Relocation

Note that no matter where the program is loaded, RDREC is always 1036 bytes past the starting address of the program. This means that we can solve the relocation problem in the following way:

1. When the assembler generates the object code for the JSUB instruction we are considering, it will insert the address of RDREC *relative to the start of the program.* (This is the reason we initialized the location counter to 0 for the assembly.)

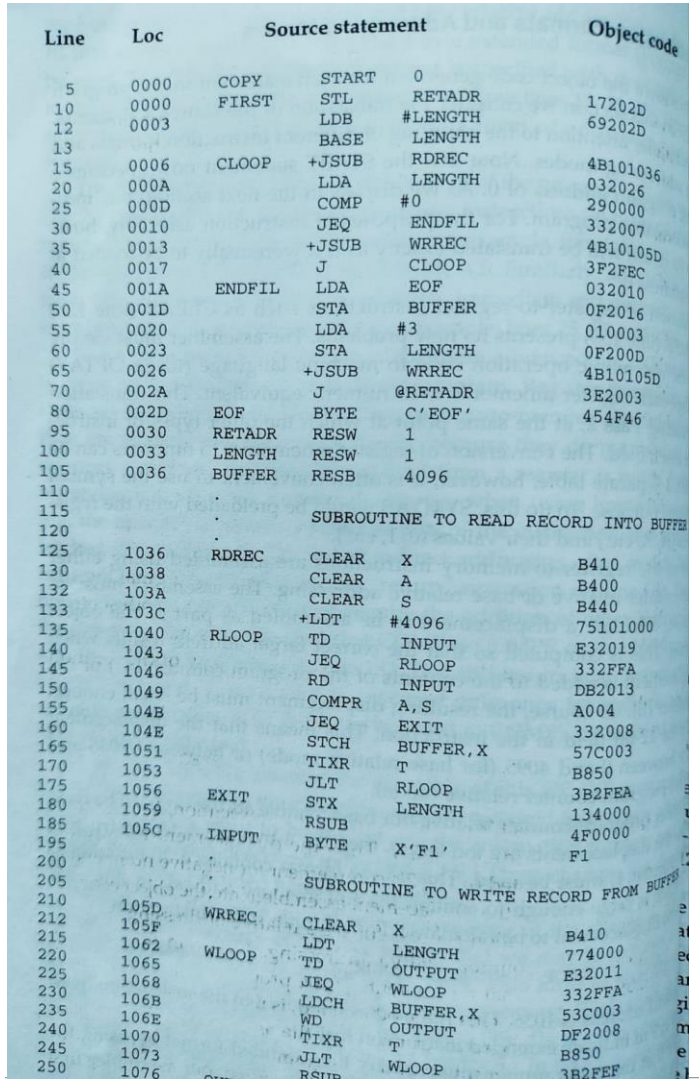
2. The assembler will also produce a command for the loader, instructing it to *add* the beginning address of the program to the address field in the JSUB instruction at load time.

Program Relocation

Modification record:

Col. 1	M
Col. 2–7	Starting location of the address field to be modified, relative to the beginning of the program (hexadecimal)
Col. 8–9	Length of the address field to be modified, in half-bytes (hexadecimal)

Program Relocation



Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER,X	57C003
165	1051		TIXR	T	B850
170	1053		JLT	RLOOP	3B2FEA
175	1056	EXIT	STX	LENGTH	134000
180	1059		RSUB		4F0000
185	105C	INPUT	BYTE	X'F1'	F1
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076				

0006 CLOOP +JSUB RDREC 4B101036

000A LDA LENGTH 032026

.

.

.

1036 RDREC CLEAR X B410

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T001036 1DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T001053 1D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107007 3B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Machine-Independent Assembler Features

Machine-Independent Assembler Features

- Literals
- Symbol-Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

Machine-Independent Assembler Features - Literals

- It is convenient if a programmer can write the value of a constant operand as a part of the instruction that uses it.

- This avoids having to define the constant elsewhere in the program
- and make up a label for it.

- Such an operand is called a **literal** because the value is stated literally in the instruction

Machine-Independent Assembler Features - Literals

- A literal is identified with the **prefix =**

```
45      001A    ENDFIL      LDA     =C'EOF'     032010
```

- Specifies a 3-byte operand whose value is the character string EOF.

```
215  1062       WLOOP       TD      =X'05'      E32011
```

- Specifies a 1-byte literal with the hexadecimal value 05

- To do:
  - Difference between Literal and immediate operand

Machine-Independent Assembler Features - Literals

- The difference between **literal** and **immediate**
- Immediate addressing**, the operand value is assembled as part of the machine instruction, no memory reference**.
- With a literal, the assembler generates the specified value as a **constant at some other memory location**.
- The *address* of this generated constant is used as the **TA** for the machine instruction, using PC-relative or base-relative addressing with memory reference.

Machine-Independent Assembler Features – Literals

- All of the literal operands used in the program are gathered together into one or more **literal pools**.
- Normally literals are placed into a pool at the **end of the program**.

```
                    .
200                 .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
210   105D    WRREC CLEAR    X                 B410
212   105F          LDT      LENGTH            774000
215   1062    WLOOP TD       =X'05'            E32011
220   1065          JEQ      WLOOP             332FFA
225   1068          LDCH     BUFFER,X          53C003
230   106B          WD       =X'05'            DF2008
235   106E          TIXR .   T                 B850
240   1070          JLT      WLOOP             3B2FEF
245   1073          RSUB                       4F0000
255                 END      FIRST
      1076    *     =X'05'                     05
```

Figure 2.10 Program from Fig. 2.9 with object code.

Machine-Independent Assembler Features – Literals

- **LTORG**
- When the assembler encounters a LTORG statement, it creates a literal pool that contains all of the literal operands used since the previous LTORG (or the beginning of the program).
- This literal pool is placed in the object program at the location where the LTORG directive was encountered.
- Note that, the literals placed in a pool by LTORG will not be repeated in the pool at the end of the program.

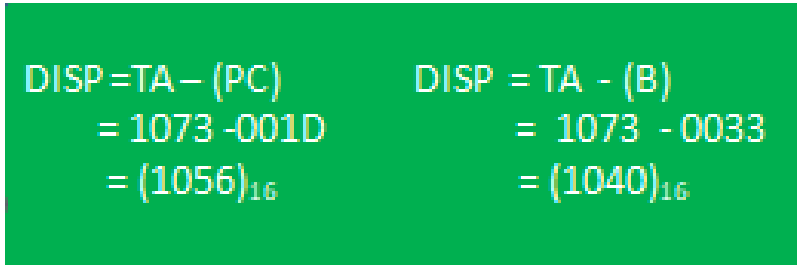
Machine-Independent Assembler Features – Literals

- **LTORG**
  - If we had not used the LTORG statement on line 93, the
  - would be placed in the pool at the end of the prog at address 1073.
  - This means that the literal operand would be place tion referencing it to allow program

```
13    0003        LDB     #LENGTH         69202D
14                BASE    LENGTH
15    0006  CLOOP +JSUB   RDREC           4B101036
20    000A        LDA     LENGTH          032026
25    000D        COMP    #0              290000
30    0010        JEQ     ENDFIL          332007
35    0013        +JSUB   WRREC           4B10105D
40    0017        J       CLOOP           3F2FEC
45    001A  ENDFIL LDA    =C'EOF'         032010
50    001D        STA     BUFFER          0F2016
55    0020        LDA     #3              010003
60    0023        STA     LENGTH          0F200D
65    0026        +JSUB   WRREC           4B10105D
70    002A        J       @RETADR         3E2003
93                LTORG
      002D  *     =C'EOF'                 454F46
95    0030  RETADR RESW   1
100   0033  LENGTH RESW   1
105   0036  BUFFER RESB   4096
106   1036  BUFEND EQU    *
107   1000  MAXLEN EQU    BUFEND-BUFFER
110               .
115               .       SUBROUTINE TO READ RECORD INTO BUFFER
120               .
125   1036  RDREC CLEAR   X               B410
130   1038        CLEAR   A               B400
132   103A        CLEAR   S               B440
133   103C        +LDT    #MAXLEN         75101000
135   1040  RLOOP TD      INPUT           E32019
140   1043        JEQ     RLOOP           332FFA
145   1046        RD      INPUT           DB2013
150   1049        COMPR   A,S             A004
155   104B        JEQ     EXIT            332008
160   104E        STCH    BUFFER,X        57C003
165   1051        TIXR    T               B850
170   1053        JLT     RLOOP           3B2FEA
175   1056  EXIT  STX     LENGTH          134000
180   1059        RSUB                    4F0000
185   105C  INPUT BYTE    X'F1'           F1
195               .
200               .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205               .
210   105D  WRREC CLEAR   X               B410
212   105F        LDT     LENGTH          774000
215   1062  WLOOP TD      =X'05'          E32011
220   1065        JEQ     WLOOP           332FFA
225   1068        LDCH    BUFFER,X        53C003
230   106B        WD      =X'05'          DF2008
235   106E        TIXR .  T               B850
240   1070        JLT     WLOOP           3B2FEF
245   1073        RSUB                    4F0000
255               END     FIRST
      1076  *     =X'05'                  05
```

Machine-Independent Assembler Features – Literals

- **LTORG**
  - The problem is the large amount of storage reserved for BUFFER.



  - By placing the literal pool before this buffer, avoids the need to use extended format instructions when referring to literals.
  - The need for an assembler directive LTORG arises when it is desirable to keep the literal operand close to the instruction that uses it.

Machine-Independent Assembler Features – Literals

- **LTORG**
  - Duplicate Literals – same literal in more than one place in the program and store only one copy of the specified data value
  - Both instructions refer to the same address in the literal pool for their operand.

```
210     105D    WRREC   CLEAR   X           B410
212     105F            LDT     LENGTH      774000
215     1062    WLOOP   TD      =X'05'      E32011
220     1065            JEQ     WLOOP       332FFA
225     1068            LDCH    BUFFER,X    53C003
230     106B            WD      =X'05'      DF2008
235     106E            TIXR    T           B850
240     1070            JLT     WLOOP       3B2FEF
245     1073            RSUB                4F0000
255                     END     FIRST
        1076    *       =X'05'              05
```

Figure 2.10 Program from Fig. 2.9 with object code.

How is a literal handled by an assembler?

Machine-Independent Assembler Features – Literals

- The basic data structure needed is a literal table (LITTAB).
- In LITTAB for each literal contains
  - the literal name,
  - the operand value and length
  - the address assigned to the operand when it is placed in a literal pool.
- LITTAB is organised as hash table, using the literal name or value as the key

LITTAB

Literal	Hex Value	Length	Address
C'EOF'	454F46	3	002D
X'05'	05	1	1076

Machine-Independent Assembler Features – Literals

- In **Pass 1**, each literal operand is recognised
  - The assembler searches LITTAB for the specified literal name (or value)
  - If the literal is already present in the LITTAB: no action is required
  - If the literal not present: the literal is added to LITTAB (leaving the address
  - unassigned)
  - When Pass 1 encounters a LTORG statement or the end of the program, the assembler makes a scan of the literal table.
  - At this time each literal currently, in the table is assigned an address
  - As these addresses are assigned, the location counter is updated to reflect the number of bytes occupied by each literal

Machine-Independent Assembler Features – Literals

- In **Pass 2**, the operand address for using in generating object code is obtained by searching LITTAB for each literal operand is encountered
- The data values specified by the literals in each literal pool are inserted at the appropriate places in the object program exactly as if these values had been generated by BYTE or WORD statements.
- If a literal value represents an address in the program (eg: location counter value), the assembler must also generate the appropriate Modification Record

Symbol-Defining Statements

Symbol-Defining Statements

- Allow the programmer to define symbols and their values
- Address label
- The label is the symbol name and the assigned address is its value
  - FIRST STL RETADR
- Assembler directive EQU
  - symbol EQU value
- This statement enters the symbol into SYMTAB and assigns to it the value specified
- The value can be a constant or an expression
- Assembler directive ORG
  - ORG value

Use of EQU

- To improve the program **readability**, avoid using the magic numbers, make it easier to find and change constant values
  - Replace

    +LDT #4096
  - with

    MAXLEN EQU 4096

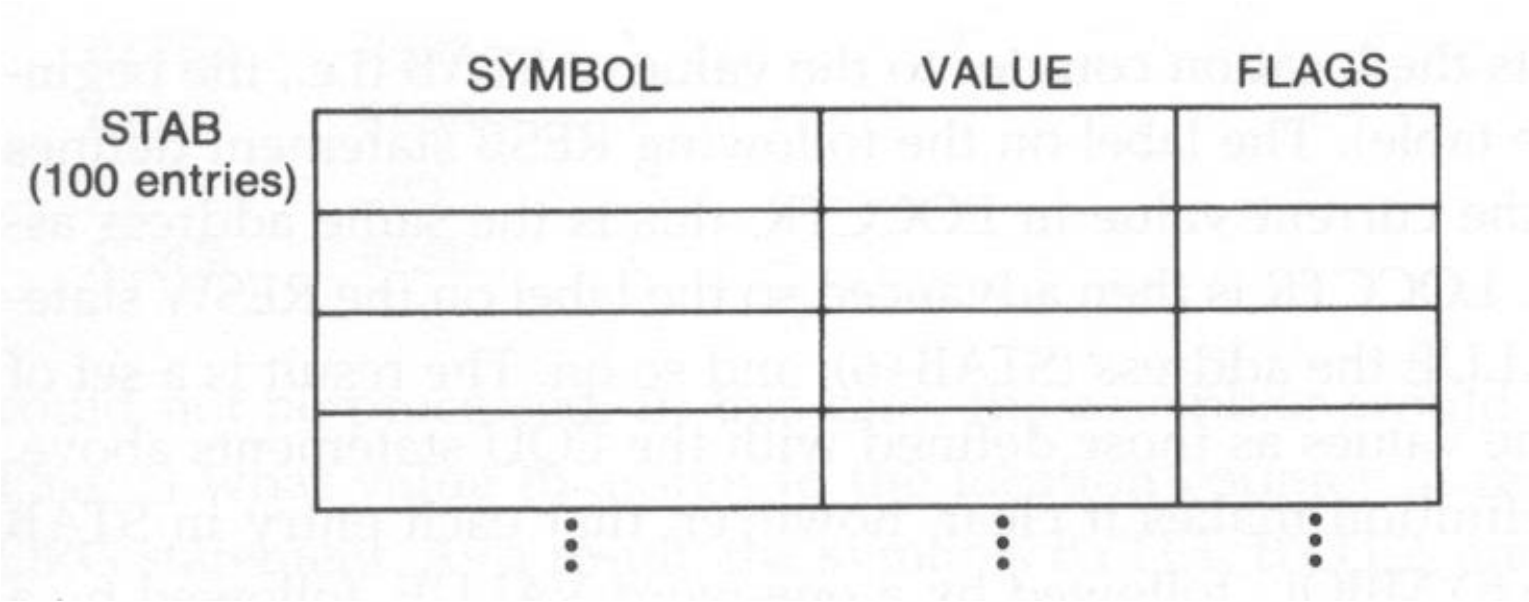
    +LDT #MAXLEN
- To define mnemonic names for registers
  - A EQU 0
  - X EQU 1
  - BASE EQU R1
  - COUNT EQU R2

ORG Directive

- Indirect value assignment:
  - ORG value
- When ORG is encountered, the **assembler resets its LOCCTR** to the specified value
- ORG will affect the values of all labels defined until the next ORG
- If the previous value of LOCCTR can be automatically remembered, we can return to the normal use of LOCCTR by simply write
  - ORG

Example of Using ORG

- Consider the following data structure
  - SYMBOL: 6 bytes
  - VALUE: 3 bytes (one word)
  - FLAGS: 2 bytes
- we want to refer to every field of each entry



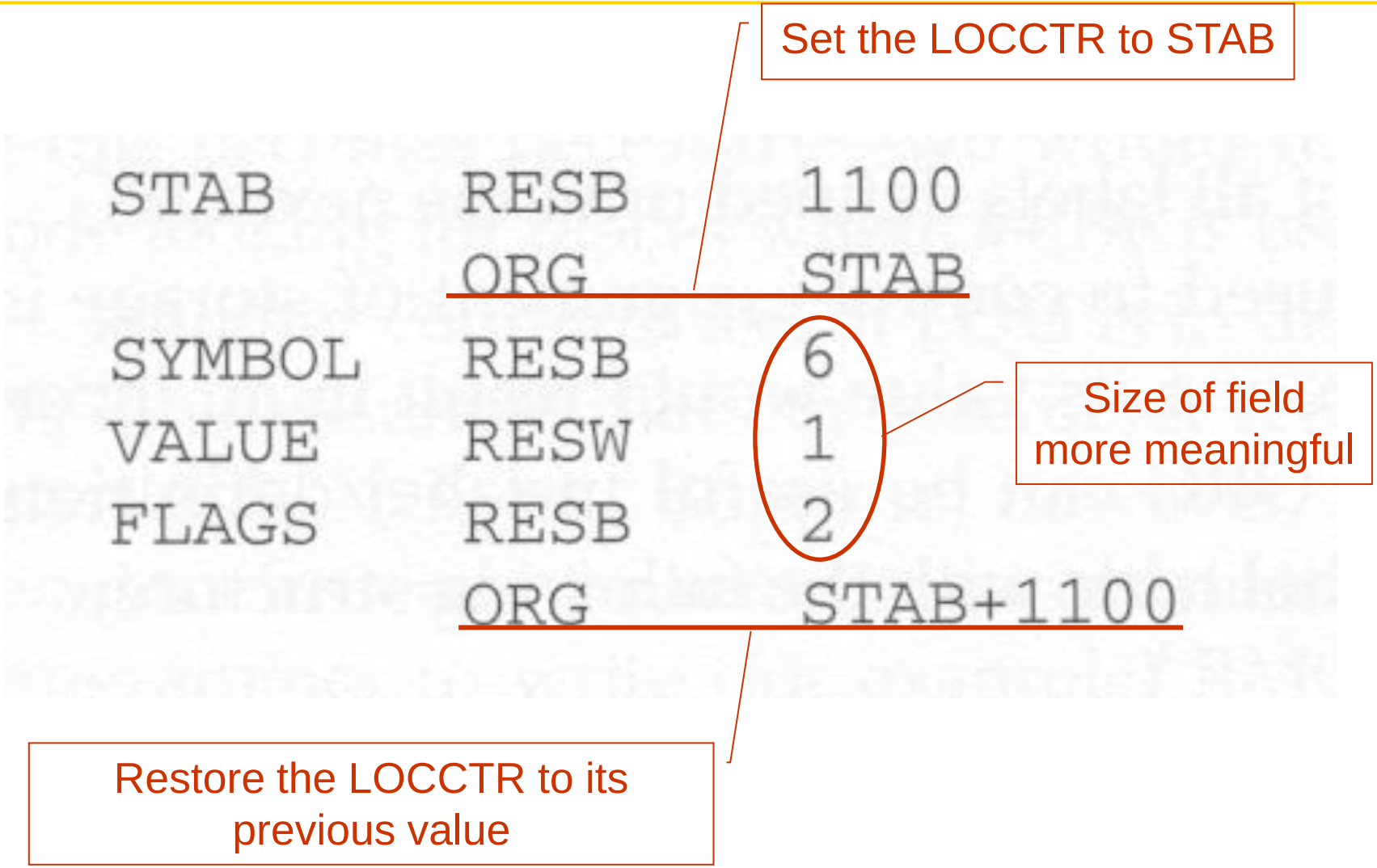
Not Using ORG

```
STAB      RESB    1100
SYMBOL    EQU     STAB
VALUE     EQU     STAB+6
FLAGS     EQU     STAB+9
```

Offsets from STAB
Less readable and meaningful

- We can fetch the VALUE field by
  - LDA VALUE,X
  - X = 0, 11, 22, … for each entry

Using ORG


Set the LOCCTR to STAB
STAB RESB 1100
ORG STAB
SYMBOL RESB 6
VALUE RESW 1
FLAGS RESB 2
ORG STAB+1100
Size of field more meaningful
Restore the LOCCTR to its previous value

Forward-Reference Problem

- Forward reference is not allowed her for EQU and ORG.
- That is, all terms in the value field must have been defined previously in the program.
- The reason is that all symbols must have been defined during Pass 1 in a two-pass assembler.

```
ALPHA    RESW    1
BETA     EQU     ALPHA
```

Allowed

```
BETA     EQU     ALPHA
ALPHA    RESW    1
```

Not allowed

Forward-Reference Problem

```
        ORG     ALPHA
BYTE1   RESB    1
BYTE2   RESB    1
BYTE3   RESB    1
        ORG
ALPHA   RESB    1
```

Not allowed

```
ALPHA   EQU     BETA
BETA    EQU     DELTA
DELTA   RESW    1
```

Not allowed

Expressions

Expressions

- A single term as an instruction operand can be replaced by an expression.

```
STAB    RESB    1100
                 ⇩
STAB    RESB    11*100
                 ⇩
STAB    RESB    (6+3+2)*MAXENTRIES
```

- The assembler has to evaluate the expression to produce a single operand address or value.
- Expressions consist of
  - Operator
    - +,-,*,/ (division is usually defined to produce an integer result)
  - Individual terms
    - Constants
    - User-defined symbols
    - Special terms, e.g., *, the current value of LOCCTR

Relocation Problem in Expressions

- Values of terms can be
  - Absolute (independent of program location)
    - constants
  - Relative (to the beginning of the program)
    - Address labels
    - * (value of LOCCTR)
- Expressions can be
  - Absolute
    - Only absolute terms
    - Relative terms in pairs with opposite signs for each pair
  - Relative
    - All the relative terms except one can be paired as described in “absolute”. The remaining unpaired relative term must have a positive sign.
- No relative terms may enter into a **multiplication or division** operation
- Expressions that do not meet the conditions of either “absolute” or “relative” should be flagged as errors.

Absolute Expression

- Relative term or expression implicitly represents (S+r)
  - S: the starting address of the program
  - r: value of the term or expression relative to S
- For example
  - BUFFER: S+r1
  - BUFEND: S+r2
- The expression, BUFEND-BUFFER, is absolute.
  - MAXLEN = (S+r2)-(S+r1) = r2-r1 (no S here)
  - MAXLEN means the length of the buffer area
- Illegal expressions:
- BUFEND+BUFFER //Not opposite terms
- 100-BUFFER // Not in pairs
- 3*BUFFER // Multiplication

Absolute or Relative

- To determine the type of an expression, we must keep track of the types of all symbols defined in the program.
- We need a "flag" in the SYMTAB for indication.

Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

Program Blocks

Program Blocks

- Program blocks
  - refer to segments of code that are rearranged within a single object program unit
  - **USE [blockname]**
  - At the beginning, statements are assumed to be part of the unnamed (default) block
  - If no USE statements are included, the entire program belongs to this single block
  - Example: Figure 2.11
  - Each program block may actually contain several separate segments of the source program

Program Blocks

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	
110						

- Three blocks
  - First: unnamed, i.e., default block
    - Line 5~ Line 70 + Line 123 ~ Line 180 + Line 208 ~ Line 245
  - Second: CDATA
    - Line 92 ~ Line 100 + Line 183 ~ Line 185 + Line 252 ~ Line 255
  - Third: CBLKS
    - Line 103 ~ Line 107

```
123   0027  0              USE
125   0027  0     RDREC    CLEAR   X           B410
130   0029  0              CLEAR   A           B400
132   002B  0              CLEAR   S           B440
133   002D  0              +LDT    #MAXLEN     75101000
135   0031  0     RLOOP    TD      INPUT       E32038
140   0034  0              JEQ     RLOOP       332FFA
145   0037  0              RD      INPUT       DB2032
150   003A  0              COMPR   A,S         A004
155   003C  0              JEQ     EXIT        332008
160   003F  0              STCH    BUFFER,X    57A02F
165   0042  0              TIXR    T           B850
170   0044  0              JLT     RLOOP       3B2FEA
175   0047  0     EXIT     STX     LENGTH      13201F
180   004A  0              RSUB                4F0000
183   0006  1              USE     CDATA
185   0006  1     INPUT    BYTE    X'F1'       F1
195                .
200                .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205                .
208   004D  0              USE
210   004D  0     WRREC    CLEAR   X           B410
212   004F  0              LDT     LENGTH      772017
215   0052  0     WLOOP    TD      =X'05'      E3201B
220   0055  0              JEQ     WLOOP       332FFA
225   0058  0              LDCH    BUFFER,X    53A016
230   005B  0              WD      =X'05'      DF2012
235   005E  0              TIXR    T           B850
240   0060  0              JLT     WLOOP       3B2FEF
245   0063  0              RSUB                4F0000
252   0007  1              USE     CDATA
253                        LTORG
      0007  1     *        =C'EOF'             454F46
      000A  1     *        =X'05'              05
255                        END     FIRST
```

How assembler handles Program Blocks?

- During Pass 1
  - Assembler accomplishes the logical rearrangement of code by maintaining a **separate location counter** for each program block.
  - When the **block begins** first, the **location counter** for a bloc is **initialized to 0**
  - When switching to another block, the current value of the location counter is saved and when resuming a previous block, the saved value is restored
  - Each label in the program is assigned an address that is relative to the start of the block that contains it.

How assembler handles Program Blocks?

- During Pass 1
  - When labels are entered into the **symbol table**, the block name or number is stored along with the assigned relative address.
  - At the end of Pass1 the latest value of location counter for each block indicates the length of that block
  - The assembler can then assign to each block a starting address in the object program (beginning with relative location 0)

How assembler handles Program Blocks?

- During Pass 2
  - The address of each symbol can be computed by adding the assigned block starting address and the relative address of the symbol to that block

How assembler handles Program Blocks?

- The column headed Loc/Block shows the relative address within a program block assigned to each source line and a block number indicating which program block is involved.

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	
110			.			

The value of MAXLEN in line 107, shown without a block number indicates that it is an absolute symbol, whose value is not relative to the start of the program

How assembler handles Program Blocks?

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	
110			.			

- Example

20 0006 0 LDA LENGTH 032060

- The address 0003 is relative to block1 CDATA
- Address 0003+0066 = 0069 relative to the program
- When this instruction is executed pc= 0009
- Thus disp = 0069 – 0009 =0060
- op nixbpe disp
- 000000 110010 060 => 032060

How assembler handles Program Blocks?

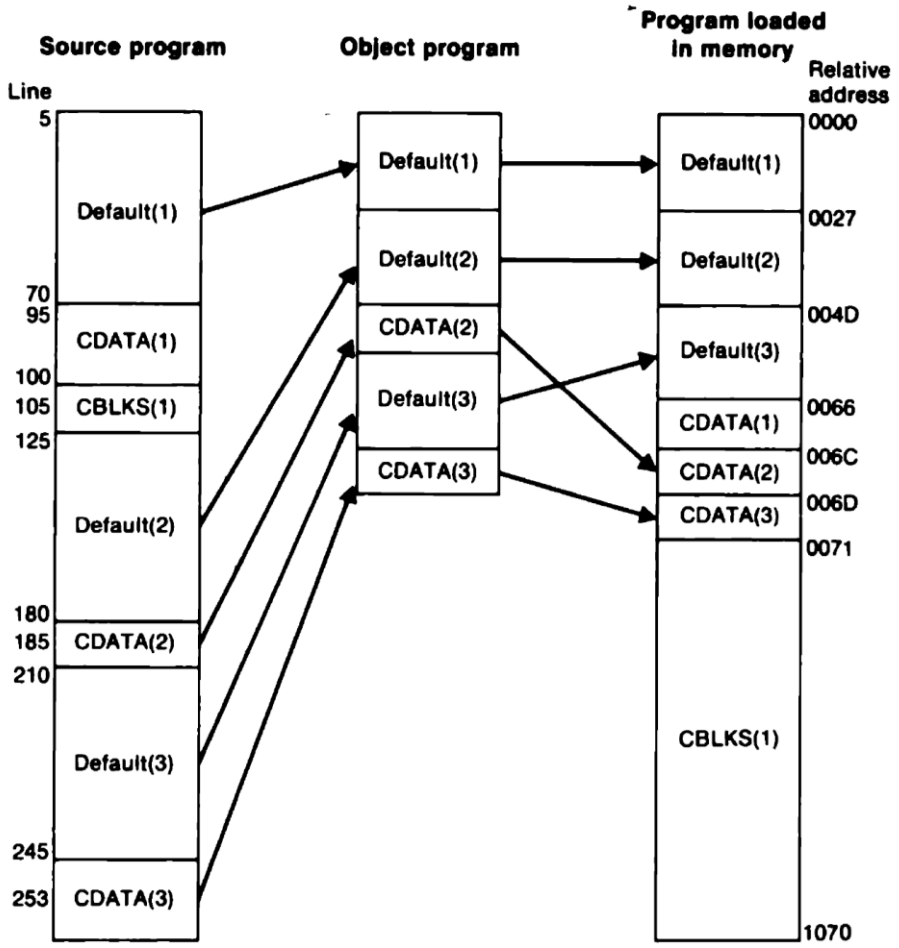


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

How assembler handles Program Blocks?

```
HCOPY  000000001071
T000000 1E 172063 4B 2021 032060 290000 332006 4B 203B 3F 2FEE 032055 0F 2056 010003
T00001E 09 0F 2048 4B 20 29 3E 203F
T000027 1D B4 10 B4 00 B4 40 75 101000 E32038 332FFA DB 2032 A004 332008 57 A02F B850
T000044 09 3B 2FEA 13 201F 4F 0000
T00006C 01 F1
T00004D 19 B4 10 77 2017 E3201B 332FFA 53 A016 DF 2012 B8 503B 2FEF 4F 0000
T00006D 04 45 4F 46 05
E000000
```

Figure 2.13 Object program corresponding to Fig. 2.11.

Thank You

Prof. Sarju S
Department of Computer Science and Engineering
St. Joseph's College of Engineering and Technology, Palai
sarju.s@sjcetpalai.ac.in