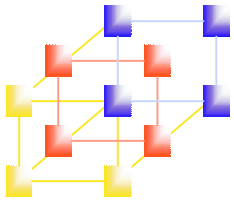
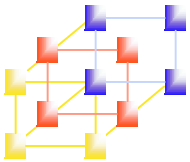


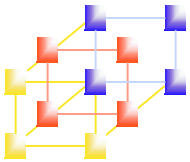
Chapter 2 Assemblers

-- 2.3 Machine-Independent Assembler Features



Outline

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

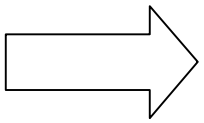


Literals

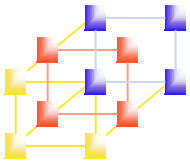
- Consider the following example

```
        :
        LDA     FIVE
        :
FIVE    WORD    5
        :
```

- It is convenient to write the value of a constant operand as a part of instruction



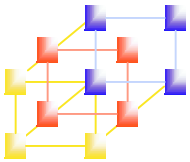
```
        :
        LDA     =X'05'
        :
```



Literals

- A literal is identified with the prefix =, followed by a specification of the literal value
- Examples: (Figure 2.10, pp.68)

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

                                    nixbpe  disp
                                  000000 110010  010

93                     LTORG

     002D *            =C'EOF'          454F46

215  1062 WLOOP  TD    =X'05'           E32011

230  106B        WD    =X'05'           DF2008

     1076 *            =X'05'           05
```



Literals vs. Immediate Operands

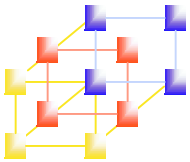
- Literals
  - The assembler generates the specified value as a constant at some other memory location

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

- Immediate Operands
  - The operand value is assembled as part of the machine instruction

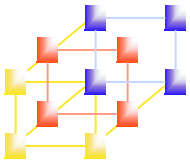
```
55    0020                LDA    #3     010003
```

- We can have literals in SIC, but immediate operand is only valid in SIC/XE.



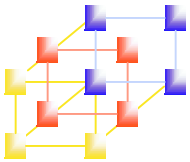
Literal Pools

- Normally literals are placed into a pool at the end of the program
  - see Fig. 2.10 (after the END statement)
- In some cases, it is desirable to place literals into a pool at some other location in the object program
  - Assembler directive LTORG
    - When the assembler encounters a LTORG statement, it generates a *literal pool* (containing all literal operands used since previous LTORG)
  - Reason: keep the literal operand close to the instruction
    - Otherwise PC-relative addressing may not be allowed



Duplicate literals

- The same literal used more than once in the program
  - Only one copy of the specified value needs to be stored
  - For example, =X’05’ in Figure 2.10 (pp. 68)
- How to recognize the duplicate literals
  - Compare the character strings defining them
    - `Easier to implement, but has potential problem (see next)`
    - `e.g. =X’05’`
  - Compare the generated data value
    - `Better, but will increase the complexity of the assembler`
    - `e.g. =C’EOF’ and =X’454F46’`

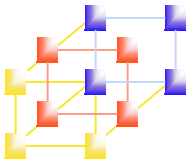


Problem of duplicate-literal recognition

- '*' denotes a literal refer to the current value of program counter
  - `BUFEND EQU *` (P.68 Fig. 2.10)
- There may be some literals that have the same name, but different values

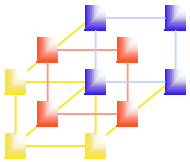
```
BASE    *
LDB     =*        (cf. P.58 #LENGTH)
```

  - The literal =* repeatedly used in the program has the same name, but different values
- The literal "=*" represents an "address" in the program, so the assembler must generate the appropriate "Modification records".



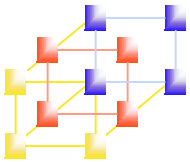
Literal table

- LITTAB
  - Content
    - `Literal name`
    - `Operand value and length`
    - `Address`
  - LITTAB is often organized as a hash table, using the literal name or value as the key



Implementation of Literals

- Pass 1
  - Build LITTAB with literal name, operand value and length, leaving the address unassigned
  - When LTORG or END statement is encountered, assign an address to each literal not yet assigned an address
    - `The location counter is updated to reflect the number of bytes occupied by each literal`
- Pass 2
  - Search LITTAB for each literal operand encountered
  - Generate data values using BYTE or WORD statements
  - Generate Modification record for literals that represent an address in the program



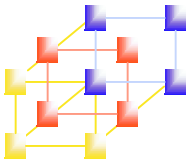
Example: (pp. 67, Figure 2.9) SYMTAB & LITTAB

SYMTAB

Name	Value
COPY	0
FIRST	0
CLOOP	6
ENDFIL	1A
RETADR	30
LENGTH	33
BUFFER	36
BUFEND	1036
MAXLEN	1000
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WREC	105D
WLOOP	1062

LITTAB

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



Symbol-Defining Statements

- Assembler directive **EQU**
  - Allows the programmer to define symbols and specify their values

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

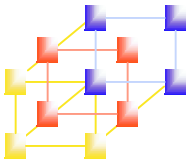
```
                        +LDT  #4096
```

  - with

```
    MAXLEN              EQU   4096
                +LDT    #MAXLEN
```

  - Define mnemonic names for registers
    - `A  EQU  0`
    - `X  EQU  1`

    `RMO A,X`
  - Expression is allowed
    - `MAXLEN  EQU  BUFEND-BUFFER`

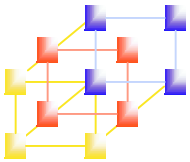


Assembler directive ORG

- Assembler directive **ORG**
  - Allow the assembler to reset the PC to values

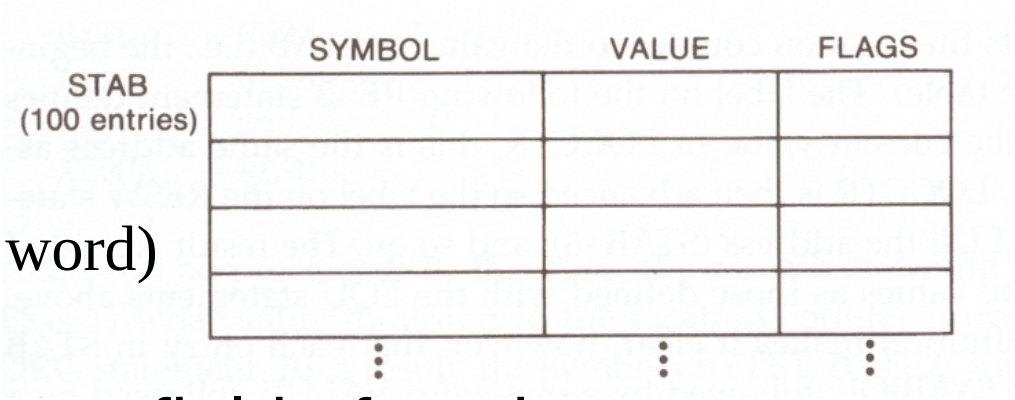
    Syntax: ORG value
  - When ORG is encountered, the assembler resets its LOCCTR to the specified value
  - ORG will affect the values of all labels defined until the next ORG
  - If the previous value of LOCCTR can be automatically remembered, we can return to the normal use of LOCCTR by simply write

    ORG



Example: using ORG

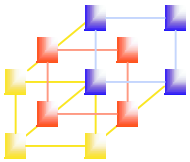
- In the data structure
  - SYMBOL: 6 bytes
  - VALUE: 3 bytes (one word)
  - FLAGS: 2 bytes



- We want to refer to every field of each entry
- If EQU statements are used

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

Offset from STAB



Example: using ORG

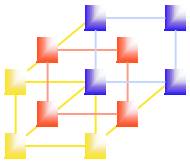
- If ORG statements are used

```
STAB            RESB    1100
                ORG     STAB        <-- Set LOCCTR to STAB
SYMBOL          RESB    6
VALUE           RESW    1           <-- Size of each field
FLAGS           RESB    2
                ORG     STAB+1100   <-- Restore LOCCTR
```

- We can fetch the VALUE field by

```
        LDA     VALUE,X
```

  - X = 0, 11, 22, … for each entry



Forward-Reference Problem

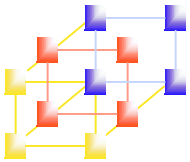
- Forward reference is not allowed for either EQU or ORG.
  - All terms in the value field must have been defined previously in the program.
  - The reason is that all symbols must have been defined during Pass 1 in a two-pass assembler.

• Allowed:

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

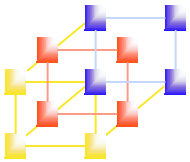
• Not allowed:

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



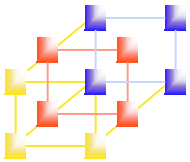
Expression

- The assemblers allow “the use of expressions as operand”
  - The assembler evaluates the expressions and produces 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`
  - Examples
    - `MAXLEN  EQU  BUFEND-BUFFER`
    - `STAB  RESB  (6+3+2)*MAXENTRIES`



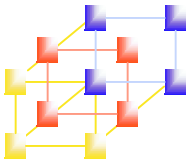
Relocation Problem in Expressions

- 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`
      - `MAXLEN  EQU  1000`
    - `Relative terms in pairs with opposite signs for each pair`
      - `MAXLEN  EQU  BUFEND-BUFFER`
  - Relative
    - `All the relative terms except one can be paired as described in “absolute”. The remaining unpaired relative term must have a positive sign.`
      - `STAB  EQU  OPTAB + (BUFEND – BUFFER)`



Restriction of Relative Expressions

- No relative terms may enter into a multiplication or division operation
  - 3 * BUFFER
- Expressions that do not meet the conditions of either “absolute” or “relative” should be flagged as errors.
  - BUFEND + BUFFER
  - 100 - BUFFER



Handling Relative Symbols in SYMTAB

- 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.

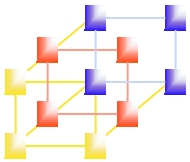
Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

- Absolute value

```
BUFEND - BUFFER
```

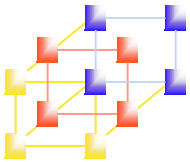
- Illegal

```
BUFEND + BUFFER
100 - BUFFER
3 * BUFFER
```



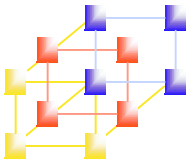
Program Blocks

- Allow the generated machine instructions and data to appear in the object program in a different order
  - Separating blocks for storing code, data, stack, and larger data block
- Program blocks v.s. Control sections
  - Program blocks
    - `Segments of code that are rearranged within a single object program unit`
  - Control sections
    - `Segments of code that are translated into independent object program units`



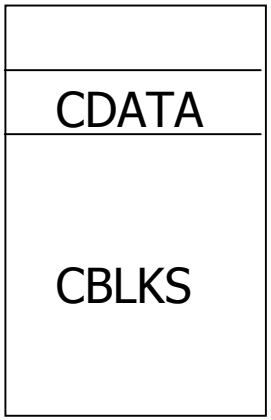
Program Blocks

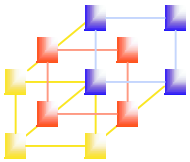
- Assembler directive: **USE**
  - `USE [blockname]`
  - At the beginning, statements are assumed to be part of the unnamed (default) block
  - If no USE statements are included, the entire program belongs to this single block
  - Each program block may actually contain several separate segments of the source program
  - Example: pp. 79, Figure 2.11



Program Blocks

- Assembler rearrange these segments to gather together the pieces of each block and assign address
  - Separate the program into blocks in a particular order
  - Large buffer area is moved to the end of the object program
  - Program readability is better if data areas are placed in the source program close to the statements that reference them.
- Example: pp, 81, Figure 2.12
  - Three blocks are used
    - `default: executable instructions`
    - `CDATA: all data areas that are less in length`
    - `CBLKS: all data areas that consists of larger blocks of memory`

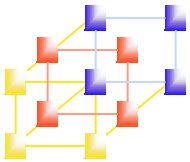




Example: pp. 81, Figure 2.12

(default) block — Block number

0000	0	COPY	START	0	
0000	0	FIRST	STL	RETADR	172063
0003	0	CLOOP	JSUB	RDREC	4B2021
0006	0		LDA	LENGTH	032060
0009	0		COMP	#0	290000
000C	0		JEQ	ENDFIL	332006
000F	0		JSUB	WRREC	4B203B
0012	0		J	CLOOP	3F2FEE
0015	0	ENDFIL	LDA	=C'EOF'	032055
0018	0		STA	BUFFER	0F2056
001B	0		LDA	#3	010003
001E	0		STA	LENGTH	0F2048
0021	0		JSUB	WRREC	4B2029
0024	0		J	@RETADR	3E203F
0000	1		USE	CDATA	← CDATA block
0000	1	RETADR	RESW	1	
0003	1	LENGTH	RESW	1	
0000	2		USE	CBLKS	← CBLKS block
0000	2	BUFFER	RESB	4096	
1000	2	BUFEND	EQU	*	
1000		MAXLEN	EQU	BUFEND-BUFFER	

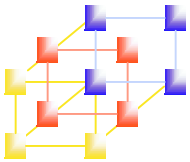


Example: pp. 81, Figure 2.12

(default) block

0027	0	RDREC	USE		
0027	0		CLEAR	X	B410
0029	0		CLEAR	A	B400
002B	0		CLEAR	S	B440
002D	0		+LDT	#MAXLEN	75101000
0031	0	RLOOP	TD	INPUT	E32038
0034	0		JEQ	RLOOP	332FFA
0037	0		RD	INPUT	DB2032
003A	0		COMPR	A,S	A004
003C	0		JEQ	EXIT	332008
003F	0		STCH	BUFFER,X	57A02F
0042	0		TIXR	T	B850
0044	0		JLT	RLOOP	3B2FEA
0047	0	EXIT	STX	LENGTH	13201F
004A	0		RSUB		4F0000
0006	1		USE	CDATA	
0006	1	INPUT	BYTE	X'F1'	F1

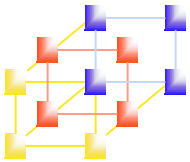
CDATA block



Example: pp. 81, Figure 2.12

(default) block

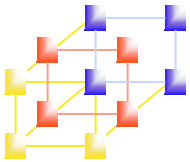
```
004D    0           USE
004D    0   WRREC   CLEAR   X           B410
004F    0           LDT     LENGTH      772017
0052    0   WLOOP   TD      =X'05'      E3201B
0055    0           JEQ     WLOOP       332FFA
0058    0           LDCH    BUFFER,X    53A016
005B    0           WD      =X'05'      DF2012
005E    0           TIXR    T           B850
0060    0           JLT     WLOOP       3B2FEF
0063    0           RSUB                4F0000
0007    1           USE     CDATA       <-- CDATA block
                    LTORG
0007    1   *       =C'EOF              454F46
000A    1   *       =X'05'              05
                    END     FIRST
```



Rearrange Codes into Program Blocks

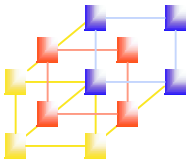
- Pass 1
  - A separate location counter for each program block
    - `Save and restore LOCCTR when switching between blocks`
    - `At the beginning of a block, LOCCTR is set to 0.`
  - Assign each label an address relative to the start of the block
  - Store the block name or number in the SYMTAB along with the assigned relative address of the label
  - Indicate the block length as the latest value of LOCCTR for each block at the end of Pass1
  - Assign to each block a starting address in the object program by concatenating the program blocks in a particular order

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



Rearrange Codes into Program Blocks

- Pass 2
  - Calculate the address for each symbol relative to the start of the object program by adding
    - `The location of the symbol relative to the start of its block`
    - `The starting address of this block`



Example of Address Calculation (P.81)

```
20 0006 0     LDA    LENGTH     032060
```

- The value of the operand (LENGTH)
  - Address 0003 relative to Block 1 (CDATA)
  - Address 0003+0066=0069 relative to program
  - When this instruction is executed
    - PC = 0009
  - disp = 0069 – 0009 = 0060
  - op nixbpe disp

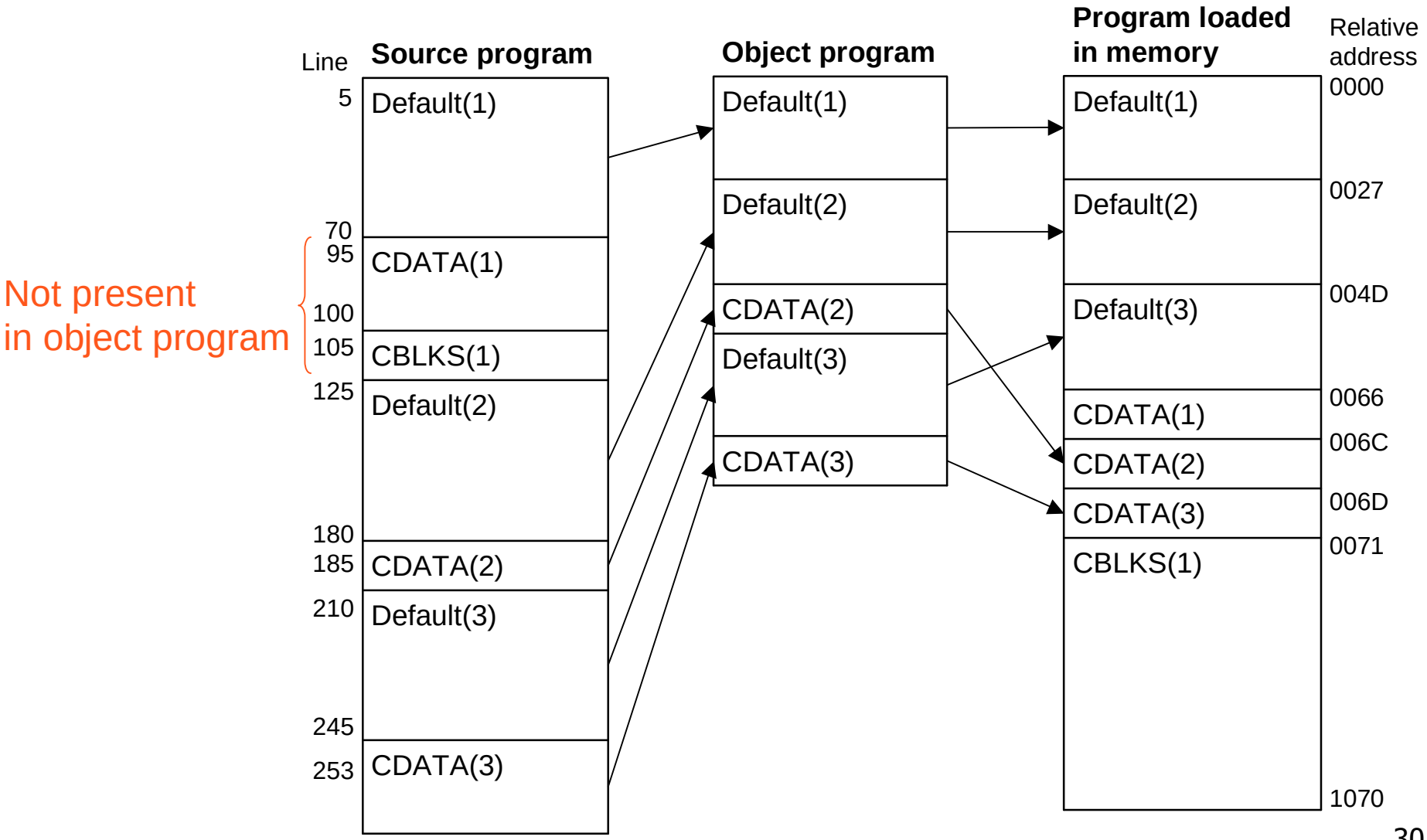
    000000 110010 060 => 032060

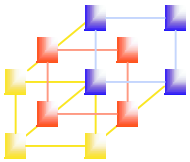
SYMTAB

label name	block num	addr.	Flag
LENGTH	1	0003	
….	….	….	….

Program Blocks Loaded in Memory

(P.84 Fig. 2.14)



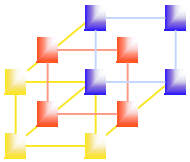


Object Program

- It is not necessary to physically rearrange the generated code in the object program
  - The assembler just simply insert the proper load address in each Text record.
  - The loader will load these codes into correct place

```
HCOPY  000000001071
T0000001E1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
T00001E090F20484B20293E203F
T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000
T00006C01F1
T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T00006D04454F4605
E000000
```

Default(1)
Default(2)
CDATA(2)
Default(3)
CDATA(3)

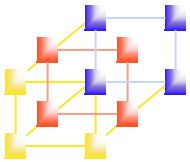


Control Sections and Program Linking

- Control sections
  - can be loaded and relocated independently of the other control sections
  - are most often used for subroutines or other logical subdivisions of a program
  - the programmer can assemble, load, and manipulate each of these control sections separately
  - because of this, there should be some means for linking control sections together
  - assembler directive: **CSECT**

    ```
    secname     CSECT
    ```

  - separate location counter for each control section



External Definition and Reference

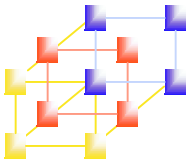
- Instructions in one control section may need to refer to instructions or data located in another section
- External definition

  EXTDEF name [, name]

  - EXTDEF names symbols that are defined in this control section and may be used by other sections
  - Ex: EXTDEF BUFFER, BUFEND, LENGTH
- External reference

  EXTREF name [,name]

  - EXTREF names symbols that are used in this control section and are defined elsewhere
  - Ex: EXTREF RDREC, WRREC
- To reference an external symbol, extended format instruction is needed (why?)

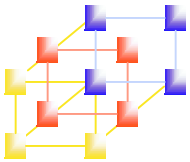


Example: pp. 86, Figure 2.15

Implicitly defined as an external symbol

first control section

```
COPY      START   0                 COPY FILE FROM INPUT TO OUTPUT
          EXTDEF  BUFFER,BUFEND,LENGTH
          EXTREF  RDREC,WRREC
FIRST     STL     RETADR            SAVE RETURN ADDRESS
CLOOP     +JSUB   RDREC             READ INPUT RECORD
          LDA     LENGTH            TEST FOR EOF (LENGTH=0)
          COMP    #0
          JEQ     ENDFIL            EXIT IF EOF FOUND
          +JSUB   WRREC             WRITE OUTPUT RECORD
          J       CLOOP             LOOP
ENDFIL    LDA     =C'EOF'           INSERT END OF FILE MARKER
          STA     BUFFER
          LDA     #3                SET LENGTH = 3
          STA     LENGTH
          +JSUB   WRREC             WRITE EOF
          J       @RETADR           RETURN TO CALLER
RETADR    RESW    1
LENGTH    RESW    1                 LENGTH OF RECORD
          LTORG
BUFFER    RESB    4096              4096-BYTE BUFFER AREA
BUFEND    EQU     *
MAXLEN    EQU     BUFFEND-BUFFER
```

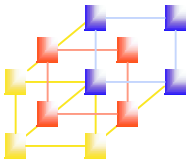


Example: pp. 86, Figure 2.15

Implicitly defined as an external symbol

second control section

```
RDREC       CSECT
.
.           SUBROUTINE TO READ RECORD INTO BUFFER
.
            EXTREF   BUFFER,LENGTH,BUFFEND
            CLEAR    X                    CLEAR LOOP COUNTER
            CLEAR    A                    CLEAR A TO ZERO
            CLEAR    S                    CLEAR S TO ZERO
            LDT      MAXLEN
RLOOP       TD       INPUT                TEST INPUT DEVICE
            JEQ      RLOOP                LOOP UNTIL READY
            RD       INPUT                READ CHARACTER INTO REGISTER A
            COMPR    A,S                  TEST FOR END OF RECORD (X'00')
            JEQ      EXIT                 EXIT LOOP IF EOR
           +STCH     BUFFER,X             STORE CHARACTER IN BUFFER
            TIXR     T                    LOOP UNLESS MAX LENGTH HAS
            JLT      RLOOP                     BEEN REACHED
EXIT       +STX      LENGTH               SAVE RECORD LENGTH
            RSUB                          RETURN TO CALLER
INPUT       BYTE     X'F1'                CODE FOR INPUT DEVICE
MAXLEN      WORD     BUFFEND-BUFFER
```

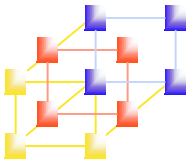


Example: pp. 86, Figure 2.15

Implicitly defined as an external symbol

third control section

```
WRREC       CSECT

.
.           SUBROUTINE TO WRITE RECORD FROM BUFFER

            EXTREF      LENGTH,BUFFER
            CLEAR       X                   CLEAR LOOP COUNTER
            +LDT        LENGTH
WLOOP       TD          =X'05'              TEST OUTPUT DEVICE
            JEQ         WLOOP               LOOP UNTIL READY
            +LDCH       BUFFER,X            GET CHARACTER FROM BUFFER
            WD          =X'05'              WRITE CHARACTER
            TIXR        T                   LOOP UNTIL ALL CHARACTERS HAVE
            JLT         WLOOP                    BEEN WRITTEN
            RSUB                            RETURN TO CALLER
            END         FIRST
```

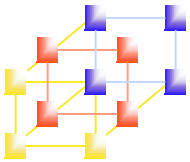


External Reference Handling

- Case 1 (P.87)

```
15    0003     CLOOP     +JSUB   RDREC       4B100000
```

  - The operand RDREC is an external reference.
  - The assembler
    - has no idea where RDREC is
    - inserts an address of zero
    - can only use extended format to provide enough room (that is, relative addressing for external reference is invalid)
  - The assembler generates information for each external reference that will allow the loader to perform the required linking.



External Reference Handling

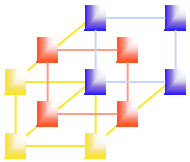
- Case 2

```
190 0028    MAXLEN      WORD      BUFEND-BUFFER      000000
```

  - There are two external references in the expression, BUFEND and BUFFER.
  - The assembler
    - inserts a value of zero
    - passes information to the loader
      - Add to this data area the address of BUFEND
      - Subtract from this data area the address of BUFFER

- Case 3
  - On line 107, BUFEND and BUFFER are defined in the same control section and the expression can be calculated immediately.

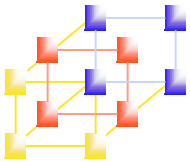
```
107     1000      MAXLEN      EQU       BUFEND-BUFFER
```



Object Code of Figure 2.15

0000	COPY	START	0	
		EXTDEF	BUFFER,BUFFEND,LENGTH	
		EXTREF	RDREC,WRREC	
0000	FIRST	STL	RETADR	172027
0003	CLOOP	+JSUB	RDREC	4B100000
0007		LDA	LENGTH	032023
000A		COMP	#0	290000
000D		JEQ	ENDFIL	332007
0010		+JSUB	WRREC	4B100000
0014		J	CLOOP	3F2FEC
0017	ENDFIL	LDA	=C'EOF'	032016
001A		STA	BUFFER	0F2016
001D		LDA	#3	010003
0020		STA	LENGTH	0F200A
0023		+JSUB	WRREC	4B100000
0027		J	@RETADR	3E2000
002A	RETADR	RESW	1	
002D	LENGTH	RESW	1	
		LTORG		
0030	*	=C'EOF'		454F46
0033	BUFFER	RESB	4096	
1033	BUFEND	EQU	*	
1000	MAXLEN	EQU	BUFEND-BUFFER	

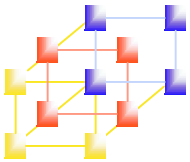
Case 1



Object Code of Figure 2.15

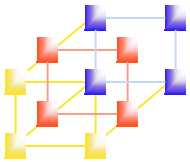
```
0000      RDREC     CSECT
                    .
                    .   SUBROUTINE TO READ RECORD INTO BUFFER
                    .
                      EXTREF   BUFFER,LENGTH,BUFEND
0000                  CLEAR    X                      B410
0002                  CLEAR    A                      B400
0004                  CLEAR    S                      B440
0006                  LDT      MAXLEN                 77201F
0009      RLOOP       TD       INPUT                  E3201B
000C                  JEQ      RLOOP                  332FFA
000F                  RD       INPUT                  DB2015
0012                  COMPR    A,S                    A004
0014                  JEQ      EXIT                   332009
0017                  +STCH    BUFFER,X               57900000
001B                  TIXR     T                      B850
001D                  JLT      RLOOP                  3B2FE9
0020      EXIT        +STX     LENGTH                 13100000
0024                  RSUB                            4F0000
0027      INPUT       BYTE     X'F1'                  F1
0028      MAXLEN      WORD     BUFFEND-BUFFER         000000
```

Case 2



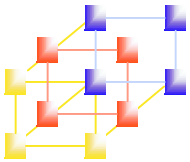
Object Code of Figure 2.15

```
0000        WRREC     CSECT

                      .
                      .   SUBROUTINE TO WRITE RECORD FROM BUFFER
                      .

                        EXTREF    LENGTH,BUFFER
0000                    CLEAR     X                          B410
0002                   +LDT       LENGTH                     77100000
0006        WLOOP       TD       =X'05'                     E32012
0009                    JEQ       WLOOP                     332FFA
000C                   +LDCH      BUFFER,X                   53900000
0010                    WD       =X'05'                     DF2008
0013                    TIXR      T                         B850
0015                    JLT       WLOOP                     3B2FEE
0018                    RSUB                                4F0000
                        END       FIRST
001B        *          =X'05'                                05
```



Records for Object Program

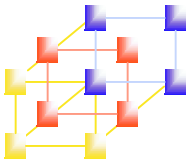
- The assembler must include information in the object program that will cause the loader to insert proper values where they are required
- Define record (EXTDEF)
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (hexadeccimal)
  - Col.14-73 Repeat information in Col. 2-13 for other external symbols
- Refer record (EXTREF)
  - Col. 1 R
  - Col. 2-7 Name of external symbol referred to in this control section
  - Col. 8-73 Name of other external reference symbols



Records for Object Program

- Modification record
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified (hexiadecimal)
  - Col. 8-9 Length of the field to be modified, in half-bytes (hexadeccimal)
  - Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field

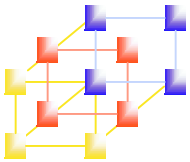
  - Control section name is automatically an external symbol, i.e. it is available for use in Modification records.



Object Program of Figure 2.15

COPY

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D172027B1000003202329000033200 7B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T000030034 54F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000
```



Object Program of Figure 2.15

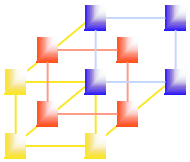
RDREC

```
HRDREC  000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E
```

BUFEND - BUFFER

WRREC

```
HWRREC  000000001C
RLENGTHBUFFER
T0000001CB41077100000E3201232FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```



Expressions in Multiple Control Sections

- Extended restriction
  - Both terms in each pair of an expression must be within the same control section
    - `Legal:  BUFEND-BUFFER`
    - `Illegal: RDREC-COPY`
- How to enforce this restriction
  - When an expression involves external references, the assembler cannot determine whether or not the expression is legal.
  - The assembler evaluates all of the terms it can, combines these to form an initial expression value, and generates Modification records.
  - The loader checks the expression for errors and finishes the evaluation.