INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

System Software
CSN-252
Assembler

2.3.1 (Literals)
2.3.2 (Symbol-Defining Statements)
- L. L. Beck

Machine-independent Assembler features

- Presence or absence of these features depend on issues such as programmer convenience

Literals

- **Write the value of a constant operand as a part of the instruction that uses it**? – more convenient for the programmer

Literals (contd.)

- Literal is identified with the prefix = (assembler language notation of SIC/XE)

Example

```
1062  WLOOP   TD      OUTPUT          E32011
              :
1076  OUTPUT  BYTE    X'05'

1062  WLOOP   TD      =X'05'          E32011?
                      (1-byte literal)
        *     = X'05'                 05
```

- No need to define the constant elsewhere in the program and make a label for it.
- Called literal because the value is stated "literally" in the instruction

Immediate vs. literal

- **Immediate** addressing: the operand value is assembled as part of the instruction
- **Literal:**
  - the assembler generates the specified value as a constant at some other location.
  - The address of the generated constant is used as the target address for the machine instruction.

```
LDA  #5          010005
LDA  =X'05'      032 - - -
```

Literals

- All of the literal operands used in the program are gathered together into one or more ***literal pools***, normally at the end of the program.
- **Assembler directive LTORG** can be used to place literals into a pool at some other location in the object program. This pool consists of all literals used since the previous LTORG.
- **Advantage** – literals can be placed in a pool near to the place of use. This avoids the use of extended format.
- Literals placed in a pool by LTORG will not be repeated in the pool at the end of the program.
- Assemblers recognize duplicate literals and store only one copy

IIT ROORKEE

5

```
0    copy     start   0          1022  wrrec   clear   x
0    first    stl     retadr             ldt     length
3             ldb     #length      loop  td      output
              base    length             jeq     loop
6             lda     eof                ldch    buffer,x
9             sta     buffer             wd      output
c             lda     #3                 tixr    t
f             sta     length             jlt     loop
12            +jsub   wrrec              rsub
16            j       @retadr      output byte    x'05'
19   eof      byte    c'eof'             end     first
1c   retadr   resw    1
1f   length   resw    1
22   buffer   resb    4096

          :
Object code of LDA?   032010
```

```
1   copy     start  0          15    wrrec  clear  x
2   first    stl    retadr     16           ldt    length
3            ldb    #length    17    loop   td     =x'05'
4            base   length     18           jeq    loop
5            lda    =c'eof'    19           ldch   buffer,x
6            sta    buffer     20           wd     =x'05'
7            lda    #3         24           tixr   t
8            sta    length     25           jlt    loop
9            +jsub  wrrec      27           rsub
10           j      @retadr    28    output byte   x'05'
11  eof      byte   c'eof'     29           end    first
12  retadr   resw   1
13  length   resw   1
14  buffer   resb   4096

          :
```

Q. Is this program correct?

```
1   copy     start  0          15    wrrec  clear  x
2   first    stl    retadr     16           ldt    length
3            ldb    #length    17    loop   td     =x'05'
4            base   length     18           jeq    loop
5            lda    =c'eof'    19           ldch   buffer,x
6            sta    buffer     20           wd     =x'05'
7            lda    #3         24           tixr   t
8            sta    length     25           jlt    loop
9            +jsub  wrrec      27           rsub
10           j      @retadr    29           end    first
11           ltorg
12  retadr   resw   1
13  length   resw   1
14  buffer   resb   4096

          :
```

Object code of LDA?

```
1    copy       start          0
2    0000       first          stl      retadr
3    0003                      ldb      #length
4                              base     length
5    0006                      lda      =c'eof'          032010
6    0009                      sta      buffer
7    000c                      lda       #3
8    000f                      sta      length
9    0012                      +jsub    wrrec
10   0016                      j        @retadr
11                             ltorg
12   0019       *              =c'eof '                  454f46
12   001c       retadr         resw     1
13   001f       length         resw     1
14    0022      buffer         resb     4096

           :
```

LITTAB

Literal name operand value length address

- Organized as hash table

Literals

- **How to recognize duplicates ?**

  - compare character strings defining them?

- Literals whose value depends upon their location in the program ?
- Example – literal that refer to the current location of the location counter (often denoted by *).

```
BASE    *
LDB     =*
```

- Loads the beginning address of the program in register B.
- **If it is used at two different places** in the program it should appear twice as its value will be different for the two cases.

- Pass 1:

  Literal in operand field?

  Search the LITTAB.

  If present no action

  else add to LITTAB (leaving the address unassigned)

  If (LTORG or end of the program) scan LITTAB and assign addresses and update location counter

- Pass 2:

  If (literal) in instruction

  search the LITTAB and get address;

  assemble instruction;

  Insert data values at the appropriate places in the object program.

  If (the literal value represents an address in the program) generate the appropriate modification record.

'C' preprocessor

```
# define MAX 1024
int main()
{
   printf("The value of MAX is %d",  MAX);
}

# define MAX(A, B) (A < B) ? B : A
int main()
{
int i = MAX(1, 2);
}
```


Assembler directive EQU – allows the programmer to define symbols and specify their values

```
symbol      EQU      value
```

- value may be a **constant or an expression** involving constants and previously defined symbols.
- Using symbolic names in place of numeric values - **improves readability**

```
Example:    +LDT     #4096
```

- What is 4096?

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