Machine Independent Assembler Features

Literal, Symbol, Expression

Literals

- Motivation
  - 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.

Literal Example (1)

```
 5     COPY      START   0               COPY FILE FROM INPUT TO OUTPUT
10     FIRST     STL     RETADR          SAVE RETURN ADDRESS
13               LDB     #LENGTH         ESTABLISH BASE REGISTER
14               BASE    LENGTH
15     CLOOP     +JSUB   RDREC           READ INPUT RECORD
20               LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
25               COMP    #0
30               JEQ     ENDFIL          EXIT IF EOF FOUND
35               +JSUB   WRREC           WRITE OUTPUT RECORD
40               J       CLOOP           LOOP
45     ENDFIL    LDA     =C'EOF'         INSERT END OF FILE MARKER
50               STA     BUFFER
55               LDA     #3              SET LENGTH = 3
60               STA     LENGTH
65               +JSUB   WRREC           WRITE EOF
70               J       @RETADR         RETURN TO CALLER
93               LTORG
95     RETADR    RESW    1
100    LENGTH    RESW    1               LENGTH OF RECORD
105    BUFFER    RESB    4096            4096-BYTE BUFFER AREA
106    BUFEND    EQU     *
107    MAXLEN    EQU     BUFEND-BUFFER   MAXIMUM RECORD LENGTH
```

Use = to represent a literal

Original Program (1)

```
 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
```

Literal Program (1)'s Object Code

```
 5     0000    COPY      START   0
10     0000    FIRST     STL     RETADR          17202D
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
```

Notice that the object code is the same as the previous one.

Literal Example (2)

```
195         .
200         .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210  WRREC  CLEAR   X               CLEAR LOOP COUNTER
212         LDT     LENGTH
215  WLOOP  TD      =X'05'          TEST OUTPUT DEVICE
220         JEQ     WLOOP           LOOP UNTIL READY
225         LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
230         WD      =X'05'          WRITE CHARACTER
235         TIXR    T               LOOP UNTIL ALL CHARACTERS
240         JLT     WLOOP             HAVE BEEN WRITTEN
245         RSUB                    RETURN TO CALLER
255         END     FIRST
```

Original Program (2)

```
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    OUTPUT  BYTE    X'05'       05
255                     END     FIRST
```

Literal Program (2)'s Object Code

```
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
```

Notice that the object code is the same as the previous one.

Difference between Literal and Immediate Operand

- Immediate addressing
  - The operand value is assembled as part of the machine instruction.
- 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 target address for the machine instruction.
  - The effect of using a literal is exactly the same as if the programming had defined the constant explicitly and used the label assigned to the constant as the instruction operand.

Literal Pool

- 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.
- Sometimes, it is desirable to place literals into a pool at some other location in the object program.
  - LTORG directive is introduced for this purpose.
  - When the assembler encounters a LTORG, it creates a pool that contains all of the literals used since the previous LTORG.

LTORG Example

- If we do not use a LTORG on line 93, the literal =C'EOF' will be placed at the end of the program.
- This operand will then begin at address 1073 and be too far away from the instruction that uses it. PC-relative addressing mode cannot be used.
- LTORG thus is used when we want to keep the literal operands close to the instruction that uses it.

Duplicate Literals

- Duplicate literals – the same literal used in more than one place in the program
- For duplicate literals, we should store only one copy of the specified data value to save space.
- Most assembler can recognize duplicate literals.
  - E.g., There are two uses of =X'05' on lines 215 and 230 respectively.
  - However, just one copy is generated on line 1076.

Recognize Duplicate Literals

- Comparing the character string defining them
  - For example, =X'05' and =X'05'
- Comparing the generated data value
  - For example, =C'EOF' and =X'454F46'
  - More intelligent
  - However, usually the benefit is not great enough to justify the added complexity.

The Problem with String-Defining Literals

- We should be careful about the literal whose value depends on their locations in the program.
- E.g., (* usually denotes the current value of the location counter)

```
BASE        *
LDB         =*
```

- If * appears on line 13, it would specify 0003. If it appears on line 55, it would specify an operand with value 0020.

Literal Processing (1)

- Need a literal table LITTAB. For each literal used, the table contains:
  - Literal name
  - The operand value and length
  - The address assigned to the operand when it is placed in a literal pool.

Literal Processing (2)

- Pass 1:
  - When encountering an literal, try to find it in LITTAB. If it can be found, do nothing.
  - Otherwise, the literal is added to the LITTAB
    - Name, operand value and length can be entered now
    - Only the address field is left blank.
  - When encountering a LTORG or the end of the program, the assembler makes a scan of LITTAB. At this time, each literal currently in this table is assigned an address. The location counter then should be updated to reflect the number of bytes occupied by each literal.

Literal Processing (3)

- Pass 2:
  - For each literal operand encountered, we search it in LITTAB and find its assigned address.
  - 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 represents an address in the program (e.g., a location counter value), the assembler must also generate the Modification record.

Symbols

- We can use the EQU directive to define a symbol's value.
  - So far, the only symbols defined in a program are labels, whose values are the addresses assigned to them.
  - E.g., MAXLEN EQU 4096.
- The value assigned to a symbol may be a constant, or any expression involving constants and previously defined symbols.

Usages of Symbols (1)

Establish symbolic names to improve readability in place of numeric values.

E.g., +LDT #4096 can be changed to :

MAXLEN EQU 4096

+LDT #MAXLEN

- When the assembler encounters the EQU statement, it enters MAXLEN into SYMTAB (with value 4096).
- During assembly of the LDT instruction, the assembler searches SYMTAB for the symbol MAXLEN, using its value as the operand in the instruction.

Usages of Symbols (2)

- Define mnemonic names for registers:

(1)

```
A       EQU     0
X       EQU     1
L       EQU     2
```

(2)

```
BASE    EQU     R1
COUNT   EQU     R2
INDEX   EQU     R3
```

No Forward Reference Allowed

- For EQU , all symbols used on the right hand side of the statement must have been defined previously in the program.
- This is because in the two-pass assembler, we require that all symbols must be defined in pass 1.

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

Allowed

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

Not allowed

ORG

- is a assembler directive
- allows the assembler to reset 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 next ORG.
- if previous value of LOCCTR can be automatically remembered, we can return to normal use of LOCCTR by simply writing ORG.
- while using ORG, forward reference is not allowed.

ORG

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

Expression

- So far, when we define the value of a symbol or label, only one term is used.
  - E.g., 106 BUFEND EQU *
- Actually, we can also use expressions which contains many terms to define a symbol's value.
  - E.g., 107 MAXLEN EQU BUFEND - BUFFER

```
          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               .
```

Relative v.s. Absolute Value

- Generally, +, -, *, /, operations are allowed to be used in an expression.
- Division is defined to produce an integer.
- Regarding program relocation, a symbol's value can be classified as
  - Relative
    - Its value is relative to the beginning of the object program, and thus its value is dependent of program location.
    - E.g., labels or reference to location counter (*)
  - Absolute
    - Its value is independent of program location
    - E.g., a constant.

Relative v.s. Absolute Expression

- Depending on the type of value they produce, expressions are classified as:
  - Absolute
    - An expression that contains only absolute terms, or
    - An expression that contains relative terms but the relative terms occur in pairs and the terms in each pair have opposite signs. (/ and * operations are not allowed)
  - Relative
    - An expression in which all relative terms except one can be paired and the remaining unpaired term must have a positive sign. (/ and * operations are not allowed)

Absolute Expression Example

- 107 MAXLEN EQU BUFEND – BUFFER
- Although BUFEND and BUFFER are relative terms (because their values will change when the program is loaded into a different place in memory), the expression (BUFEND – BUFFER) is an absolute expression.
- Why? the value of this expression is 0x1000, which is the same no matter where this program is loaded.
- Why? Because BUFEND and BUFFER can be represented as (startingaddr + x) and (startingaddr + y), BUFEND – BUFFER becomes (x – y), which is a constant.

Illegal Expressions

- BUFEND + BUFFER
- 100 – BUFFER
- 3 * BUFFER
- These expressions represent neither absolute nor location within the program
- Therefore, these expression are considered illegal.

Enhanced Symbol Table

- To determine the type of an expression, we must keep track of the types of all symbols defined in the program,
- Therefore, we need a flag in the symbol table to indicate type of value (absolute or relative) in addition to the value itself.

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