Assembler Directives

The **assembler directives** do not emit machine language but, as the name indicates, direct the assembler to perform certain operations only during the assembly process.

Here are a number of directives that we shall discuss.

CSECT	Identifies the start or continuation of a control section.
DSECT	Identifies the start or continuation of a dummy control section, which is used to pass data to subroutines.
EJECT	Start a new page before continuing the assembler listing.
END	End of the assembler module or control section.
EQU	Equate a symbol to a name or number.
LTORG	Begin the literal pool.
PRINT	Sets some options for the assembly listing.
SPACE	Provides for line spacing in the assembler listing.
START	Define the start of the first control section in a program.
TITLE	Provide a title at the top of each page of assembler listing.
USING	Indicates the base registers to use in addressing.

CSECT

By definition, a **control section** (CSECT), is “a block of coding that can be relocated (independent of other coding) without altering the operating logic of the program.”*****

Every program to be executed must have at least one control section.

If the program has only one control section, as is usually the case, we may begin it with either a **CSECT** or **START** directive.

According to Abel, a **START** directive “defines the start of the **first control section** in a program”******, though he occasionally contradicts himself.

We shall later discuss reasons why a program might need more than one control section. In this case, it is probably best to use only the **CSECT** directive.

* The definition is taken from page 109 of **Programming Assembler Language** by Peter Abel, 3rd Edition, ISBN 0 – 13 –728924 – 3. The segment in quotes is taken directly from Abel, who also has it in quotes. The source is some IBM document.

** Abel, page 577. But see page 40. Abel has trouble giving a definition.

DSECT

A **DSECT (Dummy Section)** is used to describe a data area without actually reserving any storage for it.

This is used to pass arguments from one program to another.

Consider a main program and a subroutine.

The main program will use the standard data definitions to lay out the data.

The subroutine will use a **DSECT**, with the same structure, in order to reference the original data.

The calling mechanism will pass the address of the original data.

The subroutine will associate that with its **DSECT** and use the structure found in the **DSECT** to generate proper addresses for the arguments.

We shall discuss Dummy Sections in more detail later.

END

The **END** statement must be the last statement of an assembler control section.

The form of the statement is quite simple. It is

```
    END  Section_Name
```

So, our first program had the following structure.

```
LAB1     CSECT
```

Some program statements

```
         END LAB1
```

Note that it easily could have been the following.

```
LAB1     START
```

Some program statements

```
         END LAB1
```

EQU

The **EQU** directive is used to equate a name with an expression, symbolic address, or number. Whenever this name is used as a symbol, it is replaced.

We might do something, such as the following, which makes the symbol **R12** to be equal to 12, and replaced by that value when the assembler is run.

```
R12     EQU  12
```

There are also uses in which symbolic addresses are equated. Consider this example.

```
PRINT   DC   CL133''

P       EQU  PRINT  Each symbol references the same address
```

One can also use the location counter, denoted by "*****", to set the symbol equal to the current address. This example sets the symbol **RETURN** to the current address.

```
RETURN   EQU   *          BRANCH TO HERE FOR NORMAL RETURN
```

The Location Counter

As the assembler reads the text of a program, from top to bottom, it establishes the amount of memory required for each instruction or item of data.

The **Location Counter** is used to establish the address for each item. Consider an instruction or data item that requires N bytes for storage.

The action of the assembler can be thought of as follows:

1. The assembler produces the binary machine language equivalent of the data item or instruction. This bit of machine language is N bytes long.
2. The machine language fragment is stored at address LC (Location Counter).
3. The Location Counter is incremented by N. The new value is used to store the next data item or instruction.

The location counter is denoted by the asterisk "*****". One might have code such as.

```
SAVE    DS    CL3
KEEP    EQU   *+5
```

Suppose the symbol **SAVE** is associated with location X'3012'. It reserves 3 bytes for storage, so the location counter is set to X'3015' after assembling the item.
The symbol **KEEP** is now associated with X'3015' + X'5' = X'301A'

LTORG

The Literal Pool contains a collection of anonymous constant definitions, which are generated by the assembler. The LTORG directive defines the start of a literal pool.

While some textbooks may imply that the LTORG directive is not necessary for use of literals, your instructor's experience is different. It appears that an explicit LTORG directive is required if the program uses literal arguments.

The classic form of the statement is as follows, where the "L" of "LTORG" is to be found in column 10 of the listing.

```
          LTORG *
```

Generally, this statement should be placed near the end of the listing, as in the next example taken from an actual program.

```
                      240 *    LITERAL POOL
                      241 ************************************
000308                242          LTORG *
000308 00000001       243                =F'1'
000000                244          END   LAB1
```

Here, line 243 shows a literal that is inserted by the assembler.

PRINT

This directive controls several options that impact the appearance of the listing.

Two common variants are:

```
PRINT ON,NOGEN,NODATA    WE USE THIS FOR NOW
PRINT ON,GEN,NODATA      USE THIS WHEN STUDYING MACROS
```

The first operand is the listing option. It has two values: **ON** or **OFF**.

ON – Print the program listing from this point on. This is the normal setting.
OFF – Do not print the listing.

The second operand controls the listing of macros, which are single statements that expand into multiple statements. We shall investigate them later.

The two options for this operand are **NOGEN** and **GEN**.

GEN – Print all the statements that a macro generates.
NOGEN – Suppress the generated code. This is the standard option.

The third operand controls printing of the hexadecimal values of constants.

DATA Print the full hexadecimal value of all constants.
NODATA Print only the leftmost 16 hex digits of the constants.

USING

A typical use would be found in our first lab assignment.

```
BALR  R12,0                ESTABLISH
USING *,R12                ADDRESSABILITY
```

The structure of this pair of instructions is entirely logical, though it may appear as quite strange.

First note that the **`USING *,R12`** is a directive, so that it does not generate binary machine language code.

The **`BALR  R12,0`** is an **incomplete subroutine call**. It loads the address of the next instruction (the one following the **`USING`**, since that is not an instruction) into R12 in preparation for a **Branch and Link** that is never executed.

The **`USING *`** part of the directive tells the assembler to use **`R12`** as a base register and begin displacements for addressing from the next instruction.

The mechanism, base register and offset, is used by IBM in order to save space. It serves to save memory space.

We shall study it later.

Directives Associated with the Listing

Here is a list of some of the directives used to affect the appearance of the printed listing that usually was a result of the program execution process.

In our class, this listing can be seen in the Output Queue, but is never actually printed on paper. As a result, these directives are mostly curiosities.

EJECT This causes a page to be ejected before it is full. The assembler keeps a count of lines on a page and will automatically eject when a specified count (maybe 66) is reached. One can issue an early page break.

SPACE This tells the assembler to place a number of blank lines between each line of the text in the listing. Values are 1, 2, .

```
SPACE
SPACE  1    Each causes normal spacing of the lines
SPACE  2    Double spacing; one blank line after each line of text
SPACE  3    Triple spacing; 2 blank lines after each line of text.
```

TITLE This allows any descriptive title to be placed at the top of each listing page. The title is placed between two single quotes.

```
TITLE   'THIS IS A GOOD TITLE'
```