Computer Concept : Assembler

I. What is assembler

Assembler is a program that convert assembly language program into machine code equivalent with information for the loader.

Assembler is machine dependent , since the database of the assembler is machine dependent.

II. Assembler Functionality

- Handle assembler directives
- Assign machine address to variables and symbols
- Translate assembly opcode into machine code

III. Forward Referencing

Forward Reference : Variable definition can be placed anywhere in the program even after being used.

Example

```
      LDAA APPLE
APPLE DS   1F
```

What it means

Line 1 : Load variable named "APPLE" into Accumulator A

Line 2 : Define storage named as "APPLE" a value $1F_{Hex}$ ($1 \times 16^1 + 15 \times 16^0 = 31_{10}$)

In the above example, the variable APPLE is defined after it is being used, this is forward referencing.

Forward referencing is allowed in most assembly language to increase freedom / flexibility in programming.

IV. 1-pass and 2-pass

Example

```
PETER ORG  1000
      LDAA APPLE
APPLE DS   1F
      END
```

What it means

Line 1 : A program named as "PETER" begin (ORG) at address 1000_{Hex}

Line 2 : Load variable named "APPLE" into Accumulator A

Line 3 : Define storage named as "APPLE" a value $1F_{Hex}$

Line 4 : END

What happen when assembler read this program

First a register called Location Counter (LOC) will act as a pointer to keep track of the memory location of the program

By looking at the LOC , the size of each instruction can be known

"PETER" is the name of the whole program

"ORG" indicate the value after ORG is the starting address of the program

1000_{Hex} is the starting address, the program will be loaded into address 1000_H in the physical memory, therefore this is an absolute program.

"END" indicate the end of the program.

When the assembler read in the program

It match the statements with the database , "LDAA" "ORG" "DS" "END" are pre-defined in the database.

Line 1 : It will first know that "ORG" indicate the start of the program, so "PETER" is the name of the whole program.

"PETER" will be passed to loader for further processing.

Line 2 : Next, it will know "LDAA" is a statement that the byte after this word is a value to be loaded into accumulator A.

But it does not understand what is "APPLE" in this stage.

Line 3 : Then it will know that "DS" means define storage, the byte before DS is the name of

the variable, and the byte after DS is the value to be stored.

Line 4 : It finally know the program ended by readimg "END".

It can be seen that the program can not be performed since the program does not know what is "APPLE" in line 2.

Then a second reading of the program is performed.

This time, the assembler know what "APPLE" is.

Therefore, this is called 2-pass assembler, it scan the program twice.

V. Functionality of 1-pass and 2-pass

1-pass

- Assign address to all statements
- Assign address to all symbols at time when they are defined
- Save address values in Symbol Table
- Process assembler directives

2-pass

- Assemble instructions (assemble instruction start HERE !!!)
- Look up Symbol Table to fillin unknowns due to forward reference
- Process assembler directives not being done during Pass 1
- Write Object code with loader infomation

That is, the assembler really start to work (translate assembly code into machine code) during pass-2, not pass-1 !

VI. Absolute Program and Relocatable Program

`PETER ORG 1000` means program "PETER" need to be loaded into memory location 1000_{Hex} , this is called absolute program

In muti-programming system, multi-programm share computer resources (such as memory location), then it is not good to pre-define the program location (you don't know is the location free or not !)

Therefore, the actual location of the program is not known until load time, and the location is dynamically assigned by OS.

Then the program should contain relocating information (to linkup the whole program body), and these programs are relocatable program.

Program relocation can be achieve by using Modification Flag M

Example
In absolute program

```
PETER ORG  1000Hex
      LDAA APPLE
APPLE DS  1F
      END
```

Assume instruction 4byte
Line 1 is in address 1000
Line 2 is in address 1004
Line 3 is in address 1008
Line 4 is in address 1012

In relocatable program

```
PETER ORG  X
      LDAA APPLE , M = 1
APPLE DS  1F    , M = 1
      END         , M = 1
```

Line 1 is in address X (assigned by OS , not known until load time)
Line 2 is then in address $X + 4$
Line 3 is then in address $X + 8$
Line 4 is then in address $X + 12$
Modification flag M are used to denote that line 2 to line 4 are program that need to be relocated with respect to line 1

VII. Control Section and Linking

In a large software project, several programmer work on different part of the program.

A control section is a subdivision of the program that can be assembled/loaded separately.

The reason of this is that different coder has different speed, some programmer works faster, if those programmer need to wait for the slow guy to do software testing then it is a waste of time.

Therefore the control section provide flexibility in large software project.

The disadvantage of control section is that it increase the integration cost in linking.

Linking = binding all subdivision into one single system

Linker and Loader will be discussed in another document.

$-END-$