Lecture 6

Statement Format - Assembly language of Intel 8088

[Label:] opcode operand(s) ; comment string

- Label is optional
- Operands are separated by commas in operand field
- Base and index register specifications, as also direct addresses specified as numeric offsets from segment base are enclosed in these parentheses
- Segment override is specified in operand to which it applies

CS : 12H [SI]

Assembler Directives

- ORG, EQU and END are similar to ORIGIN, EQU and END directives

- Start directive is not supported since ORG has the same function

Declarations

- Constants and reservation of storage are both achieved in same directive

```
A   DB      25              ;Reserve byte & initialize
B   DW      ?               ;Reserve word, no initialization
C   DD      6DUP(0)         ;6 double words,  all 0's
```

- DB reserves 1 byte
- DQ reserve quad-word bytes (8 bytes)
- DT reserve ten bytes (10 bytes)

EQU and PURGE

- EQU defines symbolic names to represent values or other symbolic names
- Names so defined can be ‘undefined’ through a PURGE statement
- Such a name can be reused for other purposes

```
XYZ     DB      ?
ABC     EQU     XYZ ;ABC represents name XYZ
PURGE   ABC     ;ABC no longer XYZ
ABC     EQU     25   ; ABC now stands for ’25’
```

SEGMENT, ENDS and ASSUME – 1/3

- SEGMENT and ENDS directives demarcate the segments in assembly program
- To assemble a symbolic reference, assembler must determine offset of symbol from start of the segment containing it

SEGMENT, ENDS and ASSUME – 2/3

- Programmer must perform following actions in the assembly program
  - Load a segment register with segment base
  - Let the assembler know which segment register contains the segment base
- Second task is performed using the ASSUME directive

SEGMENT, ENDS and ASSUME – 3/3

- ASSUME <register> : <segment name>
  - It tells the assembler that it can assume the address of the indicated segment to be present in <register>

- ASSUME <register> : NOTHING
  - This cancels any prior assumptions indicated for <register>

Example

```
SAMPLE_DATA     SEGMENT
ARRAY           DW          100 DUP ?
SUM             DW          0
SAMPLE_DATA     ENDS

SAMPLE_CODE     SEGMENT

                ASSUME      DS : SAMPLE_DATA
        HERE :  MOV         AX, SAMPLE_DATA
                MOV         DS, AX
                MOV         AX, SUM
                ---
SAMPLE_CODE     ENDS
                END
```

PROC, ENDP, NEAR and FAR – 1/2

- PROC and ENDP delimit the body of the procedure
- NEAR and FAR appearing in the operand field of PROC indicate whether the procedure is to be assembled as a near or far call

- NEAR – the procedure is in the same segment as the call instruction
- FAR – the procedure is in a different segment
- RET must appear in the body of the procedure to return execution control to the calling program

PROC, ENDP, NEAR and FAR – 2/2

```
SAMPLE_CODE   SEGMENT
CALCULATE             PROC          FAR     ;a FAR procedure
                      ---
                      RET
CALCULATE             ENDP
SAMPLE_CODE   ENDS

PGM                   SEGMENT
                      ---
                      CALL    CALCULATE             ;a far CALL
                      ---
PGM                   ENDS
                      END
```

PUBLIC and EXTRN – 1/3

- PUBLIC – A symbolic name declared in one assembly module can be accessible in other modules

- EXTRN – used by a module that wants to use PUBLIC symbolic

  EXTRN **<symbolic name> : <type>**

- For labels of DC, DS statements, the type can be BYTE, WORD, DWORD, QWORD, and TBYTE
- For labels of instructions, type can be FAR or NEAR

PUBLIC and EXTRN – 2/3

```
;Module #1:

                public  Var1, Var2, Proc1
DSEG            segment para public 'data'
Var1            word    ?
Var2            word    ?
DSEG            ends

CSEG            segment para public 'code'
                assume  cs:cseg, ds:dseg
Proc1           proc     near
                mov     ax, Var1
                add     ax, Var2
                mov     Var1, ax
                ret
Proc1           endp
CSEG            ends
                end
```

PUBLIC and EXTRN – 3/3

```
;Module #2:
                extern  Var1:word, Var2:word, Proc1:near
CSEG            segment para public 'code'
                  .
                  .
                  .
                mov     Var1, 2
                mov     Var2, 3
                call    Proc1
                  .
                  .
                  .
CSEG            ends
                end
```

Kindly go through these…

- OFFSET, TYPE, SIZE and LENGTH
- PTR
- Forward references
- Segment registers
- Cross references