Writing Program For Use With An Assembler

Assembler Use

- Models that are unique to particular assembler
- With full segment definitions that allow complete control over the assembly process and are universal to all assemblers
  - Ref: Ex.4.18 & 4.19 of T2

Writing Program For Use With An Assembler

- Write a simple, complete program and explain the function of various parts of the program
  1. Define problem and write the algorithm
  2. Set up data structure
     - Will data in memory or reg?
     - Type of data
     - How many data items are there?
     - Data is signed or unsigned?
     - May Complex data structure…like array/record is necessary?
  3. Use logical segment to define data structure to be used for the program

Example

```
0000          DATA SEGMENT
0000 94             HI_TEMP DB 94H
0001 ??             A_TEMP  DB ?
0002          DATA ENDS
```

4. Initialization checklist at the start of CS (Identify any instruction required to initialize variables/ Segment reg,etc?)

EX: Initialize DS register

```
CODE SEGMENT
        ASSUME CS: CODE,DS:DATA
        MOV AX,DATA
        MOV DS,AX
CODE ENDS
```

5. Determine the instructions required
   - To implement each of the major actions in the algorithms
   - How data must be positioned for these instructions

6. Finally get data into correct position

7. Perform operations

8. Store result.

Directives

- Indicates how an operand or section of a program is to be processed by the assembler
- Some directives generate & stores information in memory
- Storing Data in a memory Segment
  - (Label SIZE data defined/?)
  - DB
  - DW
  - DD
  - DQ
  - DT
  - Ex: 0000 DATA1 DB 1,2,3----------?
    0006 DATA2 DW 12,13----------?
- EQU

```
TEN EQU 10
MOV AL,TEN
```

Memory organization

- **.MODEL** directive
  - Specifies the memory configuration
  - From tiny (64KB),SMALL(128K)….
  - Directive followed by size of memory system
- SEGMENT and ENDS directive
  - To define logical segments
  - Ex:

```
0000 STACK_SEG  SEGMENT   'STACK'
             STACK_SEG   ENDS
```

TITLE and .MODEL Directives

- **TITLE** line (optional)
  - Contains a brief heading of the program and the disk file name
- **.686** processor directive
  - Used **before** the **.MODEL** directive
  - Program can use instructions of Pentium P6 architecture

.STACK, .DATA, & .CODE Directives

- **.STACK** directive
  - Tells the assembler to define a runtime stack for the program
  - The size of the stack can be optionally specified by this directive
  - The runtime stack is required for procedure calls
- **.DATA** directive
  - Defines an area in memory for the program data
  - The program's variables should be defined under this directive
  - Assembler will allocate and initialize the storage of variables
- **.CODE** directive
  - Defines the code section of a program containing instructions
  - Assembler will place the instructions in the code area in memory

INCLUDE, PROC, ENDP, and END

- **INCLUDE** directive
  - Causes the assembler to include code from another file
- **PROC** and **ENDP** directives
  - Used to define procedures
  - As a convention, we will define ***main*** as the first procedure
  - Additional procedures can be defined after ***main***
- **END** directive
  - Marks the end of a program
  - Identifies the name (***main***) of the program's startup procedure

MACRO

- Macro.. At the top of CS
- Define before its use
- LOCAl directive
  - May have 35 labels seperated by commas
  - Immediately following MACRO statement

JMP & CALL Revisited Vs MACRO

- Ex

```
MOVE MACRO A,B
    PUSH Ax
    MOV AX,B
    MOV A,AX
    POP AX
ENDM
MOVE VAR1,VAR2
```

Writing Program For Use With An Assembler

```
.MODEL SMALL
.STACK 200
.DATA
        NUM1 DW   0FA62H
        NUM2 DB   94H
.CODE
.STARTUP

        MOV  AX,  NUM1        ;load AX with number NUM1
        AND  AX,  0FFDFH      ;Reset 6th bit of AX
        OR   AL,  20H         ;Set 6th bit of AL
        XOR  NUM1, 0FF00H     ;Complement the high order byte of
                              ; NUM1
        NOT NUM2              ;Complement NUM2
        XOR AX, AX            ;Clear AX
        MOV AX, NUM1
        AND AX, 0008H         ; Isolate bit 4 of NUM1
        XOR AX, 0080H         ;Complement 4th bit of AX
.EXIT
END
```

Effects of executing program

Statement	Destination Content		Status Flags							
	Before	After	OF	DF	IF	SF	ZF	AF	PF	CF
1. MOV AX, NUM1										
2. AND AX, 0FFDFH										
3. OR AL, 20H										
4. XOR NUM1, 0FF00H										
5. NOT NUM2										
6. XOR AX, AX										
7. MOV AX, NUM1										
8. AND AX, 0008H										
9. XOR AX, 0080H										

```
.MODEL SMALL
.STACK 200
.DATA
        RADIX DB 10             ;radix: 10 for decimal
        NUM   DW 0EFE4H         ;the number to be converted
                                ;put here any other number.
        ;Note that: 0EFE4H = 61412(10)
        TEMP  DB    10 DUP(?)   ;Used to simulate a stack

.CODE
.STARTUP
        MOV  AX, NUM            ;load AX with number NUM
                                ;display AX in decimal
        MOV  CX, 0              ;clear digit counter
        XOR  BH, BH             ;clear BH
        MOV  BL, RADIX          ;set for decimal
        XOR  SI, SI             ;Clear SI register
DISPX1:
        MOV  DX, 00             ;clear DX
        DIV  BX                 ;divide DX:AX by 10
        MOV  TEMP[SI], DL       ;save remainder
        INC  SI
        INC  CX                 ;count remainder
        OR   AX, AX             ;test for quotient of zero
        JNZ  DISPX1             ;if quotient is not zero
        DEC  SI

DISPX2:
        MOV  DL, TEMP[SI]
        MOV  AH, 06H            ;get remainder
        ADD  DL, 30H            ;select function 06H
        INT  21H                ;convert to ASCII
        DEC  SI                 ;display digit
        DEC  CX
        JNZ  DISPX2             ;repeat for all digits
.EXIT
END                             ;exit to dos
```

Effects of executing program

Statement	Destination Content		Status Flags							
	Before	After	OF	DF	IF	SF	ZF	AF	PF	CF