Assembly Language Programming

Format of Assembly Language Instructions

[Label] Operation [Operands] [; Comment]

- **Example: Examples of instructions with varying numbers of fields.**
- [Label] Operation [Operands] [; Comment]

```
L1:     cmp     bx, cx      ; Compare bx with cx  all fields present
        add     ax, 25      ;operation and 2 operands
        inc     bx          ;operation and 1 operand
        ret                 ;operation field only
      ; Comment: whatever you wish !!  comment field only
```

Program Statements

`name operation operand(s) comment`

- Operation is a predefined or reserved word
  - mnemonic - symbolic operation code
  - directive - pseudo-operation code
- Space or tab separates initial fields
- Comments begin with semicolon
- Most assemblers are not case sensitive

Program Data and Storage

- Pseudo-ops to define data or reserve storage
  - DB - byte(s)
  - DW - word(s)
  - DD - doubleword(s)
  - DQ - quadword(s)
  - DT - tenbyte(s)
- These directives require one or more operands
  - define memory contents
  - specify amount of storage to reserve for run-time data

Defining Data

- Numeric data values
  - 100 - decimal
  - 100B - binary
  - 100H - hexadecimal
  - '100' - ASCII
  - "100" - ASCII
- Use the appropriate DEFINE directive (byte, word, etc.)
- A list of values may be used - the following creates 4 consecutive words

```
DW 40CH,10B,-13,0
```

- A ? represents an uninitialized storage location

```
DB 255,?,-128,'X'
```

Naming Storage Locations

- Names can be associated with storage locations

```
ANum DB -4
 DW 17
ONE
UNO DW 1
X DD ?
```

- These names are called variables
- ANum refers to a byte storage location, initialized to FCh
- The next word has no associated name
- ONE and UNO refer to the same word
- X is an unitialized doubleword

Arrays

- Any consecutive storage locations of the same size can be called an array

```
X DW 40CH,10B,-13,0
Y DB 'This is an array'
Z DD -109236, FFFFFFFFH, -1, 100B
```

- Components of X are at X, X+2, X+4, X+8
- Components of Y are at Y, Y+1, …, Y+15
- Components of Z are at Z, Z+4, Z+8, Z+12

DUP

- Allows a sequence of storage locations to be defined or reserved
- Only used as an operand of a define directive

```
DB 40 DUP (?)
DW 10h DUP (0)
DB 3 dup ("ABC")
db 4 dup(3 dup (0,1), 2 dup('$'))
```

Word Storage

- Word, doubleword, and quadword data are stored in reverse byte order (in memory)

```
Directive      Bytes in Storage
DW 256         00 01
DD 1234567H    67 45 23 01
DQ 10          0A 00 00 00 00 00 00 00
X DW 35DAh     DA 35
```

Low byte of X is at X, high byte of X is at X+1

Named Constants

- Symbolic names associated with storage locations represent addresses
- Named constants are symbols created to represent specific values determined by an expression
- Named constants can be numeric or string
- Some named constants can be redefined
- No storage is allocated for these values

Equal Sign Directive

- name = expression
  - expression must be numeric
  - these symbols may be redefined at any time

```
maxint = 7FFFh
count = 1
DW count
count = count * 2
DW count
```

EQU Directive

- name EQU expression
  - expression can be string or numeric
  - Use < and > to specify a string EQU
  - these symbols cannot be redefined later in the program

```
sample EQU 7Fh
aString EQU <1.234>
message EQU <This is a message>
```

Data Transfer Instructions

- MOV *target, source*
  - reg, reg
  - mem, reg
  - reg, mem
  - mem, immed
  - reg, immed
- Sizes of both operands must be the same
- reg can be any non-segment register except IP cannot be the target register
- MOV's between a segment register and memory or a 16-bit register are possible

Sample MOV Instructions

```
b db 4Fh
w dw 2048

mov bl,dh
mov ax,w
mov ch,b
mov al,255
mov w,-100
mov b,0
```

- When a variable is created with a define directive, it is assigned a default size attribute (byte, word, etc)
- You can assign a size attribute using LABEL

```
LoByte LABEL BYTE
aWord DW 97F2h
```

Addresses with Displacements

```
b db 4Fh, 20h, 3Ch
w dw 2048, -100, 0

mov bx, w+2
mov b+1, ah
mov ah, b+5
mov dx, w-3
```

- Type checking is still in effect

- The assembler computes an address based on the expression
- *NOTE: These are address computations done at assembly time*

  MOV ax, b-1

  will not subtract 1 from the value stored at b

Exchange Command

- XCHG *target, source*
  - reg, reg
  - reg, mem
  - mem, reg
- MOV and XCHG cannot perform memory to memory moves
- This provides an efficient means to swap the operands
  - No temporary storage is needed
  - Sorting often requires this type of operation
  - This works only with the general registers

Arithmetic Instructions

ADD *dest, source*

SUB *dest, source*

INC *dest*

DEC *dest*

NEG *dest*

- *Operands must be of the same size*
- *source* can be a general register, memory location, or constant
- *dest* can be a register or memory location
  - except operands cannot both be memory

Program Segment Structure

- Data Segments
  - Storage for variables
  - Variable addresses are computed as offsets from start of this segment
- Code Segment
  - contains executable instructions
- Stack Segment
  - used to set aside storage for the stack
  - Stack addresses are computed as offsets into this segment
- Segment directives

```
.data
.code
.stack size
```

Memory Models

- .Model *memory_model*
  - tiny: code+data <= 64K (.com program)
  - small: code<=64K, data<=64K, one of each
  - medium: data<=64K, one data segment
  - compact: code<=64K, one code segment
  - large: multiple code and data segments
  - huge: allows individual arrays to exceed 64K
  - flat: no segments, 32-bit addresses, protected mode only (80386 and higher)

Program Skeleton

```
.model small
.stack 100H
.data
  ;declarations
.code
main proc
  ;code
main endp
  ;other procs
end main
```

- Select a memory model
- Define the stack size
- Declare variables
- Write code
  - organize into procedures
- Mark the end of the source file
  - optionally, define the entry point

Sample Program

```
.MODEL SMALL
.STACK 100H
.DATA
      LF      EQU     0AH
      CR      EQU     0DH
      MSG1    DB      'INSIDE MAIN PROGRAM',LF,CR,'$'
      MSG2    DB      'INSIDE PROC1',LF,CR,'$'
      MSG3    DB      'INSIDE PROC2',LF,CR,'$'
.CODE
MAIN PROC FAR
            MOV     AX,@DATA
            MOV     DS,AX
            LEA     DX,MSG1
            MOV     AH,09H
            INT     21H
            CALL    PROC1
            MOV     AH,4CH
            INT     21H
MAIN ENDP
```

```
PROC1 PROC
            LEA     DX,MSG2
            MOV     AH,09H
            INT     21H
            CALL    PROC2
            RET
PROC1 ENDP
PROC2 PROC
            LEA     DX,MSG3
            MOV     AH,09H
            INT     21H
            RET
PROC2 ENDP
END MAIN
```