CSC 222: Computer Organization & Assembly Language

6 – Assembly Language Basics

Outline

- Assembly Language – Basic Elements
  - Statement Syntax: Name Field, Operation Field, Operand Field, Comments
  - Program Data
  - Variables
  - Named Constants
- A Few Basic Instructions
- Translation of High Level Language to Assembly Language
- Program Structure
- Input Output Instructions

References

- ***Chapter 3, 4,*** Ytha Yu and Charles Marut, “Assembly Language Programming and Organization of IBM PC”
- ***Chapter 3***, Assembly Language for Intel Based-Computers

Basic Elements

Statements

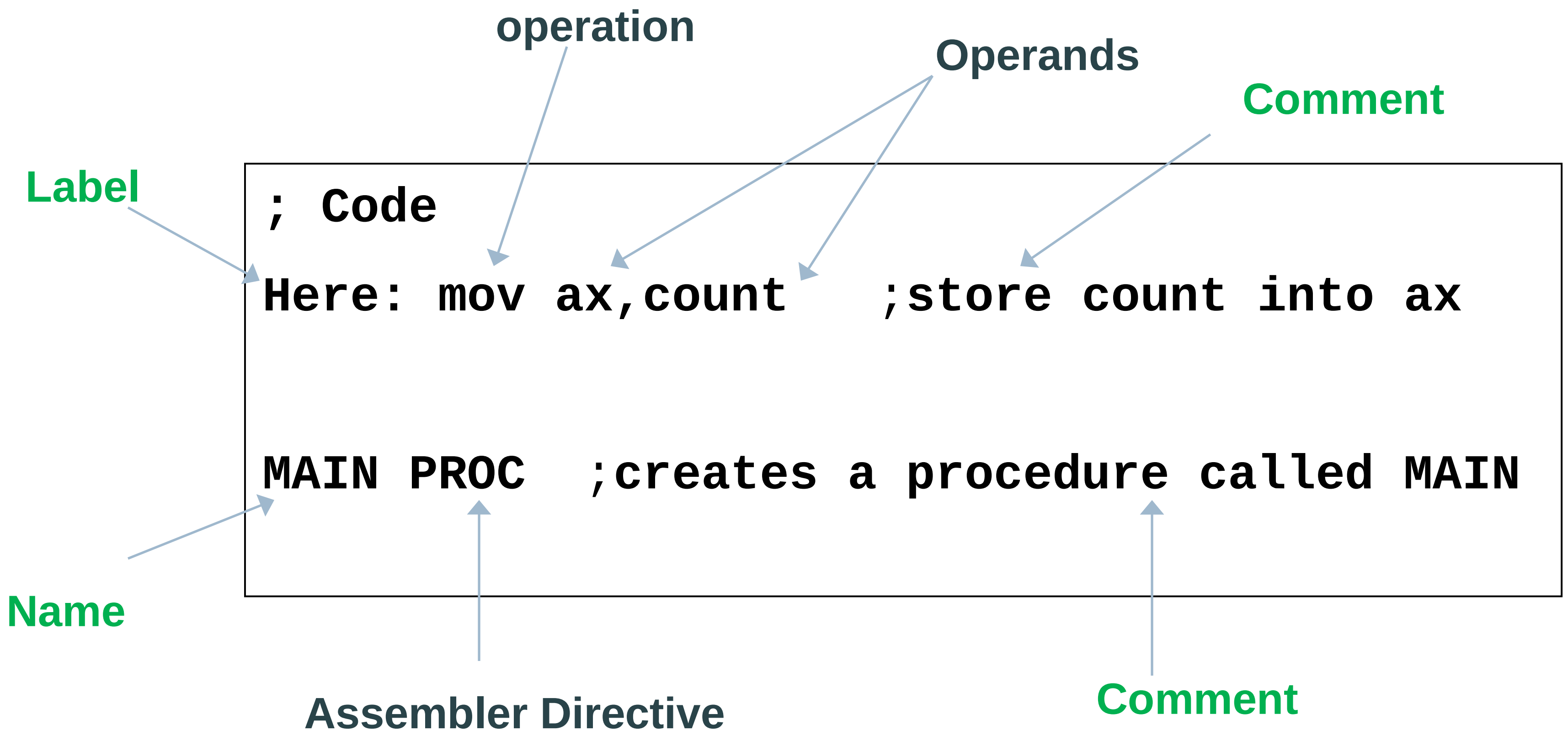
- Syntax:

  name   operation operand(s) comments

  - name and comment are optional
  - Number of operands depend on the instruction
  - One statement per line
  - At least one blank or tab character must separate the field.

  - Each statement is either:
  - Instruction (translated into machine code)
  - Assembler Directive (instructs the assembler to perform some specific task such as allocating memory space for a variable or creating a procedure)

Statement Example



```
; Code
Here: mov ax,count    ;store count into ax


MAIN PROC  ;creates a procedure called MAIN
```

Name/Label Field

- The assembler translates names into memory addresses.
- Names can be 1 to 31 character long and may consist of letter, digit or special characters. If period is used, it must be first character.
- Embedded blanks are not allowed.
- May not begin with a digit.
- Not case sensitive

Examples of legal names	Examples of illegal names
COUNTER_1	TWO WORDS
@character	2abc
.TEST	A45.28
DONE?	YOU&ME

Operation Field: Symbolic operation (Op code)

- Symbolic op code translated into Machine Language op code
- ***Examples***: ADD, MOV, SUB

- In an assembler directive, the operation field represents Pseudo-op code
- Pseudo-op is not translated into Machine Language op code, it only tells assembler to do something.
- ***Example***: **PROC** psuedo-op is used to create a procedure

Operand Field

- An instruction may have zero, one or more operands.
- In two-operand instruction, first operand is destination, second operand is source.
- For an assembler directive, operand field represents more information about the directive
- ***Examples***

```
NOP             ;no operand, does nothing
INC AX          ;one operand, adds 1 to the contents of AX
    ADD AX, 2        ;two operands, adds value 2 to the
contents of AX
```

Comments

- Optional
- Marked by semicolon in the beginning
- Ignored by assembler
- Good practice

Program Data

- Processor operates only on binary data.
- In assembly language, you can express data in:
  - Binary
  - Decimal
  - Hexadecimal
  - Characters
- Numbers
  - For Hexadecimal, the number must begin with a decimal digit. E.g.: write 0ABCh not only ABCH.
  - Cannot contain any non-digit character. E.g.: 1,234 not allowed
- Characters enclosed in single or double quotes.
  - ASCII codes can be used
  - No difference in “A” and 41h

Contd..

- Use a radix symbol (suffix) to select binary, octal, decimal, or hexadecimal

```
6A15h         ; hexadecimal
0BAF1h        ; leading zero required
32q           ; octal
1011b         ; binary
35d           ; decimal (default)
```

Variables

- Each variable has a data type and is assigned a memory address by the program.
- Possible Values:
  - Numeric, String Constant, Constant Expression, ?
  - **8 Bit Number Range**: Signed (-128 to 127), Unsigned (0-255)
  - **16 Bit Number Range:** Signed (-32,678 to 32767), Unsigned (0-65,535)
  - **?** To leave variable uninitialized

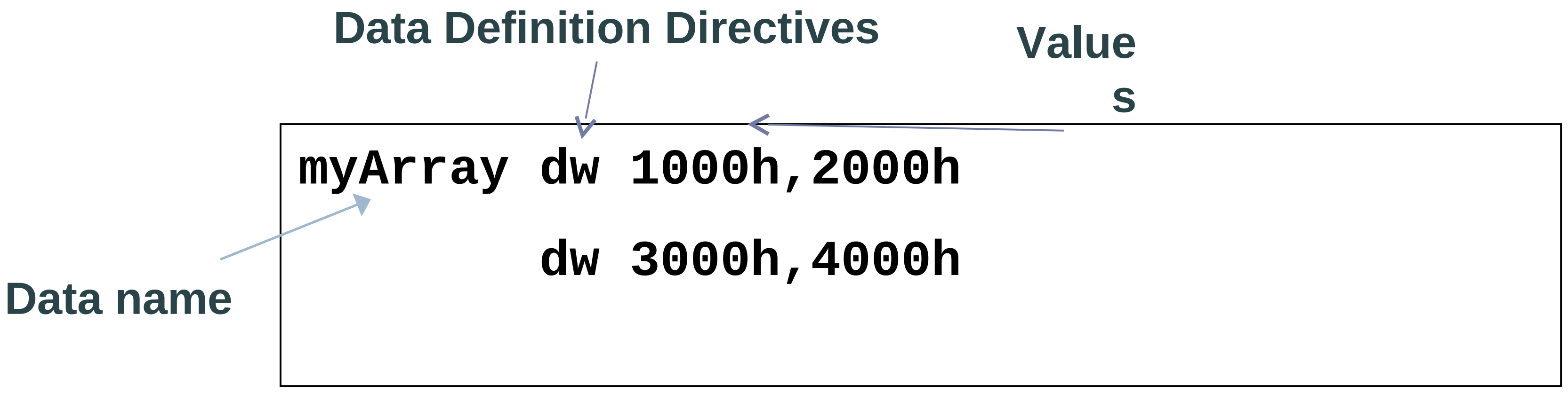
Contd..

- Syntax

  variable_name type initial_value

  variable_name type value1, value2, value3

- Data Definition Directives Or Data Defining Pseudo-ops
  - DB, DW, DD, DQ, DT



```
myArray dw 1000h,2000h
        dw 3000h,4000h
```

Remember: you can skip variable name!

Contd..

Pseudo-ops	Description	Bytes	Examples
DB	Define Byte	1	var1 DB ‘A’ Var2 DB ? array1 DB 10, 20,30,40
DW	Define Word	2	var2 DW ‘AB’ array2 DW 1000, 2000
DD	Define Double Word	4	Var3 DD -214743648

Note:
Consider

var2 DW 10h

Still in memory the value saved will be 0010h

Arrays

- Sequence of memory bytes or words
- **Example 1:**

  B_ARRAY DB 10h, 20h, 30h

Symbol	**Address**	**Contents**
B_ARRAY	0200h	10h
B_ARRAY+1	0201h	20h
B_ARRAY+2	0202h	30h

***If B_ARRAY is assigned offset address 0200h by assembler**

Example 2

- W_ARRAY DW 1000, 40, 29887, 329

***If W_ARRAY is assigned offset address 0300h by assembler**

Symbol	**Address**	**Contents**
W_ARRAY	0300h	1000d
W_ARRAY+ 2	0302h	40d
W_ARRAY+ 4	0304h	29887d
W_ARRAY+ 6	0306h	329d

- **High & Low Bytes of a Word**

  WORD1 DW 1234h
- Low Byte = 34h, symbolic address is WORD1
- High Byte = 12h, symbolic address is WORD1+1

Character String

LETTERS DB ‘ABC’

Is equivalent to

LETTERS DB 41h, 42h, 43h

- Assembler differentiates between upper case and lower case.
- Possible to combine characters and numbers.

MSG DB ‘HELLO’, 0Ah, 0Dh, ‘$’

Is equivalent to

MSG DB 48h, 45h, 4Ch, 4Ch, 4Fh, 0Ah, 0Dh, 24h

Example 3

- Show how character string “RG 2z” is stored in memory starting at address 0.
- Solution:

Addres s	Character	ASCII Code (HEX)	ASCII Code (Binary) [Memory Contents]
0	R	52	0101 0010
1	G	47	0100 0111
2	Space	20	0010 0000
3	2	32	0011 0010
4	z	7A	0111 1010

Named Constants

- Use symbolic name for a constant quantity
- **Syntax**:

```
name    EQU    constant
```

- **Example**:

```
LF      EQU    0Ah
```

- No memory allocated

A Few Basic Instructions

MOV

- Transfer data
  - Between registers
  - Between register and a memory location
  - Move a no. directly to a register or a memory location
- Syntax

  MOV *destination, source*
- Example

  MOV *AX, WORD1*

	Before	***After***
AX	0006	0008
WORD1	0008	0008

- **Difference?**
  - MOV AH, ‘A’
  - MOV AX, ‘A’

Legal Combinations of Operands for MOV

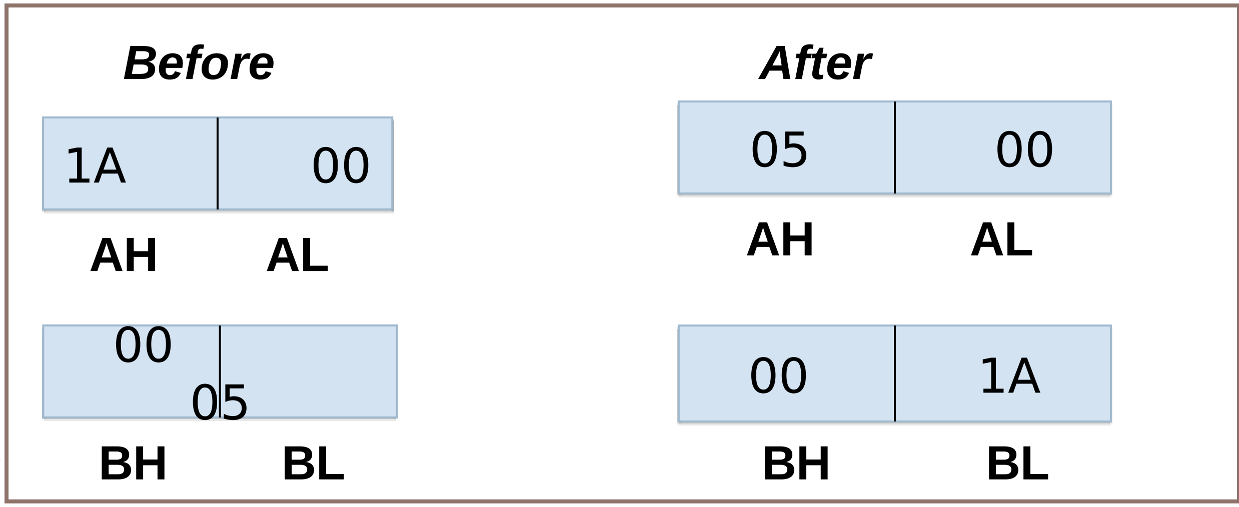
Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Segment Register	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Segment Register	YES
Memory Location	Constant	YES

XCHG

- Exchange the contents of
  - Two registers
  - Register and a memory location
- Syntax

  XCHG *destination, source*
- Example

  XCHG *AH, BL*



Legal Combinations of Operands for XCHG

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO

ADD Instruction

- To add contents of:
  - Two registers
  - A register and a memory location
  - A number to a register
  - A number to a memory location
- Example

ADD WORD1, AX

	Before	***After***
AX	01BC	01BC
WORD1	0523	06DF

SUB Instruction

- To subtract the contents of:
  - Two registers
  - A register and a memory location
  - A number from a register
  - A number from a memory location
- Example

  SUB AX, DX

	Before	***After***
AX	0000	FFFF
DX	0001	0001

Legal Combinations of Operands for ADD & SUB instructions

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Constant	YES

Contd..

ADD BYTE1, BYTE2   ILLEGAL instruction

- Solution?

  MOV AL, BYTE2

  ADD BYTE1, AL

- **What can be other possible solutions?**

- **How can you add two word variables?**

INC & DEC

- **INC** (increment) instruction is used to add 1 to the contents of a register or memory location.
  - Syntax: INC *destination*
  - Example: INC WORD1

- **DEC** (decrement) instruction is used to subtract 1 from the contents of a register or memory location.
  - Syntax: DEC *destination*
  - Example: DEC BYTE1

- Destination can be 8-bit or 16-bits wide.
- Destination can be a register or a memory location.

Contd..

INC WORD1

	Before	***After***
WORD1	0002	0003

DEC BYTE1

	Before	***After***
BYTE1	FFFE	FFFD

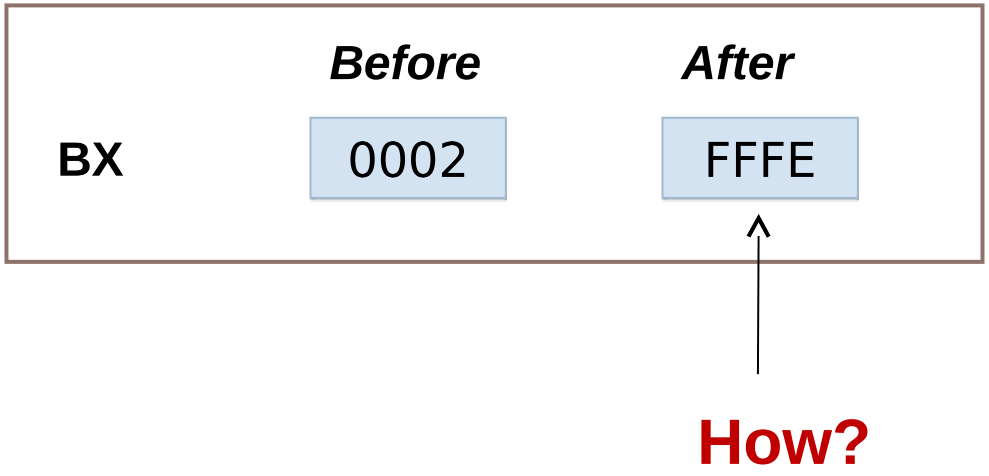
NEG

- Used to negate the contents of destination.
- Replace the contents by its 2’s complement.
- Syntax

  NEG *destination*

- Example

  NEG BX



Translation

Examples

- Consider instructions: MOV, ADD, SUB, INC, DEC, NEG
- **A** and **B** are two word variables
- Translate statements into assembly language:

Statement	Translation
B = A	MOV AX, A MOV B, AX
A = 5 - A	MOV AX, 5 SUB AX, A MOV AX, A **OR** NEG A ADD A, 5

Contd..

Statement	**Translation**
A = B - 2 x A	MOV AX, B SUB AX, A SUB AX, A MOV AX, A

- **Remember:** Solution not unique!
- **Be careful!** Word variable or byte variable?

Program Structure

Program Segments

- Machine Programs consists of
  - Code
  - Data
  - Stack
- Each part occupies a memory segment.
- Same organization is reflected in an assembly language program as **Program Segments**.
- Each program segment is translated into a memory segment by the assembler.

Memory Models

- Determines the size of data and code a program can have.
- Syntax:

.MODEL memory_model

Model	Description
SMALL	code in one segment, data in one segment
MEDIUM	code in more than one segment, data in one segment
COMPACT	code in one segment, data in more than one segment
LARGE	Both code and data in more than one segments No array larger than 64KB
HUGE	Both code and data in more than one segments

Data Segment

- All variable definitions
- Use **.DATA** directive
- For Example:

```
.DATA
WORD1 DW 2
 BYTE1 DB 10h
```

Stack Segment

- A block of memory to store stack
- Syntax

  .STACK size

  - Where size is optional and specifies the stack area size in bytes
  - If size is omitted, 1 KB set aside for stack area

- For example:

  .STACK 100h

Code Segment

- Contains a program’s instructions
- Syntax

  .CODE name

  - Where name is optional
  - Do not write name when using SMALL as a memory model

Putting it Together!

```
ORG 0100h

.MODEL SMALL
.STACK 100h

.DATA
   ;data definition go here
.CODE
   ;instructions go here
```

DOS Interrupt 21H

- **Option 1 – Inputs a single character from keyboard and echoes it to the monitor.**
- **Registers used:**
  - **AH = 1**
  - **AL = the character inputted from keyboard.**
- **Ex:**
  - **MOV AH,1**
  - **INT 21H**

DOS Interrupt 21H

- **Option 2 – Outputs a single character to the monitor.**
- **Registers used:**
  - **AH = 2**
  - **DL = the character to be displayed.**
- **Ex:**
  - **MOV AH,2**
  - **MOV DL,’A’**
  - **INT 21H**

DOS Interrupt 21H

- **Option 9 – Outputs a string of data, terminated by a $ to the monitor.**
- **Registers used:**
  - **AH = 9**
  - **DX = the offset address of the data to be displayed.**
- **Ex:**
  - **MOV AH,09**
  - **MOV DX,OFFSET MESS1**
  - **INT 21H**

DOS Interrupt 21H

- **Option 0AH – Inputs a string of data from the keyboard.**
- **Registers used:**
  - **AH = 0Ah**
  - **DX = the offset address of the location where string will be stored.**
- **DOS requires that a buffer be defined in the data segment. It should be defined as follows:**
  - **1st byte contains the size of the buffer.**
  - **2nd byte is used by DOS to store the number of bytes stored.**

DOS Interrupt 21H

- **Option 4CH – Terminates a process, by returning control to a parent process or to DOS.**
- **Registers used:**
  - **AH = 4CH**
  - **AL = binary return code.**
- **Ex:**
  - **MOV AH,4CH**
  - **INT 21H**

Program to Display “Hello World!”

```
.MODEL SMALL
.STACK 100H
.DATA
   STRING_1 DB "HELLO,WORLD $"
.CODE
   MAIN PROC


      MOV AX,@DATA
      MOV DS,AX



      LEA DX,STRING_1
      MOV AH,9
      INT 21H


      MOV AH,4CH
      INT 21H

   MAIN ENDP

END MAIN
```

Print a character (assembly code)

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
  MAIN PROC

    MOV AH,2
    MOV DL,"@"
    INT 21H



    MOV AH,4CH
    INT 21H


  MAIN ENDP
END MAIN
```

Display two input character

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
   MAIN PROC



      MOV AH,1
      INT 21H

      MOV BL,AL



      MOV AH,2
      MOV DL,0DH
      INT 21H

      MOV DL,0AH
      INT 21H



      MOV AH,1
      INT 21H

      MOV BH,AL
```

```
        MOV AH,2
            MOV DL,0DH
            INT 21H

            MOV DL,0AH
            INT 21H



            MOV AH,2
            MOV DL,BL
            INT 21H



            MOV AH,2
            MOV DL,0DH
            INT 21H

            MOV DL,0AH
            INT 21H



            MOV AH,2
            MOV DL,BH
            INT 21H



            MOV AH,4CH
            INT 21H


        MAIN ENDP

    END MAIN
```