ASSEMBLY LANGUAGE PROGRAMMING
CSE - 2214

Instructors

Asma Enayet
Lecturer
Dept. of CSE
University of Dhaka

Tahsin Aziz
Lecturer
Dept. of CSE
Ahsanullah University of Science and Technology

CHAPTER 4

Assembly Language Syntax

Each statement is either

- An instruction
  - Which the assembler translates into machine code
  - For example, MOV CX,5
- Or an assembler directive
  - Which instructs the assembler to perform some specific task, such as allocating memory space for a variable or creating a procedure
  - For example, MAIN PROC

Assembly Language Syntax

- Both instructions and directives have up to four fields:

name operation operand(s) comment

Name Field

- It is used for instruction labels, procedure names and variable names
- The assembler translates names into memory addresses.
- Names can be 1 to 31 characters long
- They may consist of letters, digits and the special characters ?, @, $, % etc
- Embedded blanks are not allowed
- If a period is used, it must be the first character.
- Names may not begin with a digit.

Name Field

Examples of legal names

- @message
- Counter
- $1000
- .TEST

Operation Field

- For an instruction, the operation field contains a symbolic operation code (opcode).
  - The assembler translates a symbolic opcode into a machine language opcode
  - For example, MOV, ADD, SUB etc.
- In an assembler directive, the operation field contains a pseudo-operation code (pseudo-op)
  - Pseudo-ops are not translated into machine code; they simply tell the assembler to do something

Operand Field

- For an instruction, the operand field specifies the data that are to be acted on by the operation.
  - An instruction may have zero, one or two operands.
  - In a two operand instruction, the first operand is the destination operand (Either memory or register)
  - The second operand is the source operand.

Operand Field

For example,

  - NOP
  - INC AX
  - ADD WORD1,2

- For an assembler directive, the operand field usually contains more information about the directive.

Data representation

- The assembler translates all data representation into binary numbers.
- In assembly language program, data can be
- Numbers
  - *Binary*
    - It is written as a bit string followed by the letter “B” or “b”
    - For example, 1010B
  - *Decimal*
    - It is a string of decimal digits ending with an optional “D” or “d”
    - For example, 64223, -2144D

Data representation

- *Hex numbers*
  - It must begin with a decimal digit and end with the letter “H” or “h”.
  - For example, 1B4DH

- Characters
  - Characters and character strings must be enclosed in single or double quotes.
  - They are translated into their ASCII codes.
  - For example , “A”, ‘hello’

Variables

- Same as high-level languages
- Defined by data-defining pseudo-op given in the following table

Pseudo-op	Stands for
DB	Define byte
DW	Define word
DD	Define doubleword
DQ	Define quadword
DT	Define tenbytes

Byte Variable

- Name DB initial_value
- For example,

ALPHA DB 4

  - This causes the assembler to associate a memory byte with the name ALPHA and initialize it to 4
- A ? Used in place with initial value sets aside an uninitialized byte.
- For example,

MSG DB ?

Word Variable

- Name DW initial_value
- For example,

```
ALPHA        DW        -2
```

  - This causes the assembler to associate a memory word with the name ALPHA and initialize it to -2
- A ? used in place with initial value sets aside an uninitialized word.
- For example,

```
MSG      DW      ?
```

High and Low bytes of a word

- The high and low bytes of a word variable can be referred.
- Example
    - WORD1   DW   1234h
  - The low byte of WORD1 contains 34h and the high byte contains 12h.
  - The low byte has symbolic address WORD1 and the high byte has address WORD1+1

Character strings

- An array of ASCII codes can be initialized with a string of characters.
- Example
    - LETTERS DB ‘ABC’

      Is equivalent to
    - LETTERS DB 41h,42h,43h

EQU (Equates)

- To assign a name to a constant, we can use the EQU pseudo-op.
- The syntax is
- Name EQU constant

Example

- LF EQU 0AH
- Assigns the name LF to 0Ah, the ASCII code of the line feed character. The name LF can now be used in place of 0Ah anywhere in the program.
- Thus the assembler translated the instructions
  - MOV DL,0Ah
  - And
  - MOV DL,LF

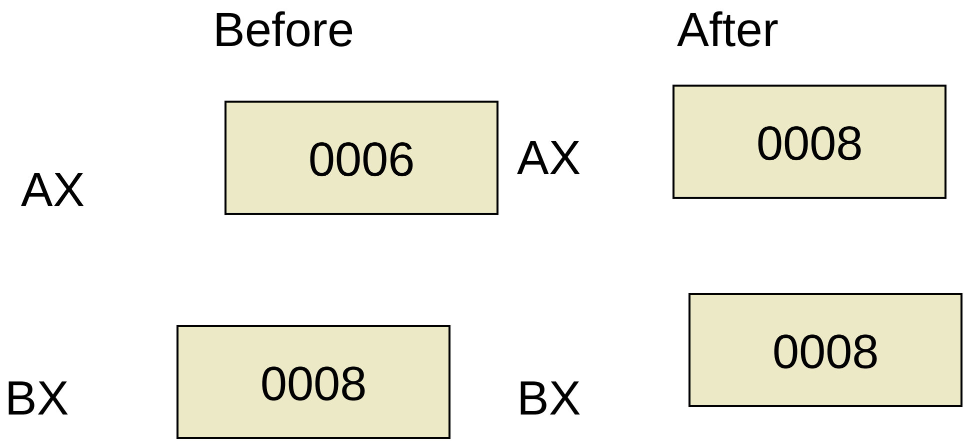
  Into the same machine instruction

EQU (Equates) continued….

- The symbol on the right of an EQU can also be a string.
- Example
  - PROMPT EQU   ‘TYPE YOUR NAME’
- Then instead of
  - MSG   DB   ‘TYPE YOUR NAME’
- We can write
  - MSG   DB   PROMPT
- No memory is allocated for EQU names

MOV

- It is used to transfer data between registers, between a register and a memory location
- The syntax is
- MOV destination, source
- Example
    - MOV AX,BX
    - MOV AH,’A’



Before

AX 0006

BX 0008

After

AX 0008

BX 0008

MOV Continued……

- Legal combination of operands for MOV

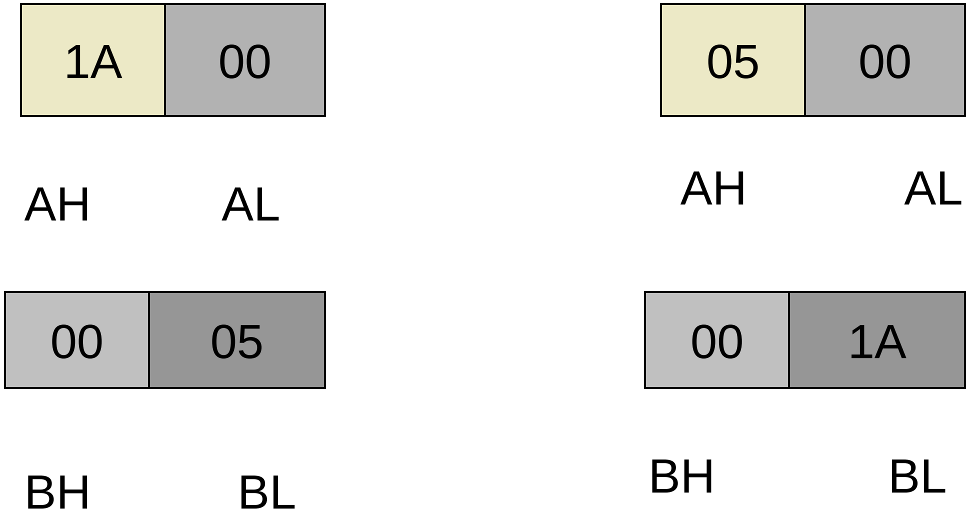
Destination Operand

Source Operand	General Register	Segment Register	Memory Location	Constant
General Register	Yes	Yes	Yes	No
Segment Register	Yes	No	Yes	No
Memory Location	Yes	Yes	No	No
Constant	Yes	No	Yes	No

XCHG

- It is used to exchange the contents of two registers, or a register and a memory location.
- The syntax is
  - XCHG destination, source
- Example
  - XCHG AH, BL
- This instruction swaps the contents of AH and BL.
  - XCHG AX, WORD1
- This instruction swaps the contents of AX and memory location WORD1

XCHG AH,BL



1A 00

AH AL

00 05

BH BL

05 00

AH AL

00 1A

BH BL

XCHG continued….

- Legal combination of operands for XCHG

Destination Operand

Source Operand	General Register	Memory Location
General Register	Yes	Yes
Memory Location	Yes	No

Restrictions on MOV and XCHG

- A MOV or XCHG between memory locations is not allowed.
- Example
  - MOV WORD1, WORD2
- Is illegal.
- But this restriction is get around by the use of registers as follows
  - MOV   AX, WORD2
  - MOV WORD1, AX

ADD and SUB

- Are used to add or subtract the contents of two registers, a register and a memory location, or to add or subtract a number to (from) a register or memory location.
- The syntax is
  - ADD destination, source
  - SUB destination, source
- Example
  - ADD WORD1, AX
  - SUB   AX, DX

ADD and SUB continued…..

- Legal combination of operands for ADD & SUB

Destination Operand

Source Operand	General Register	Memory Location
General Register	Yes	Yes
Memory Location	Yes	No
Constant	Yes	Yes

INC and DEC

- INC is used to add 1 to the contents of a register or memory location
- DEC subtracts 1 from a register or memory location.
- The syntax is
  - INC destination
  - DEC destination
- Example
  - INC WORD1
  - Adds 1 to the contents of WORD1

NEG

- It is used to negate the contents of the destination.
- NEG does this by replacing the contents by its two’s complement,
- The syntax is
  - NEG destination
- The destination may be a register or memory location
- Example
  - NEG BX

Type agreement of operands

- The operands of the two operand instructions must be of same type……….either both bytes or words.
- Example
  - MOV AX, BYTE1   ;illegal
  - MOV AH, ‘A’   is allowed.
- Source most be a byte. Place 41h in AH
  - And MOV AX, ‘A’
- Source must be a word. Place 0041h in AX

Translation of High-Level Language to Assembly Language

- Example-1
  - B = A
    - MOV AX, A
    - MOV B, AX
- Example-2
  - A = 5-A
    - MOV AX, 5
    - SUB AX,
    - MOV A,A
- Or use NEG

  NEG A

  ADD A,

- A = B – 2xA

Memory Model

- The size of code and data in a program is determined by specifying a memory model using the .MODEL directive
- The syntax is
  - .MODEL memory_model
- 5 types
  - SMALL
  - MEDIUM
  - COMPACT
  - LARGE
  - HUGE

Memory Model Continued

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	**Code in more than one segment** **Data in more than one segment** **No array larger than 64k bytes**
HUGE	**Code in more than one segment** **Data in more than one segment** **Arrays may be larger than 64k**

Segments

- Data segment
  - Contains all the variable and constants definitions
  - .DATA directive followed by variable declarations
- Stack segment
  - It is used to set aside a block of memory to store stack
  - .STACK size
- Code segment
  - It contains a program’s instructions
  - Inside a code segment, instructions are organized as procedures.
  - Syntax
    - name PROC

The INT instruction

- To invoke a DOS or BIOS routine, INT is used
- Syntax
  - INT   interrupt_number
  - Where interrupt_number specifies a routine.

INT 21h

- Is used to invoke a large number of DOS functions
- A particular DOS function is requested by placing a function number in the AH register and invoking 21h.

Function number	Routine
1	Single-key input
2	Single-character output
9	Character string output

Function 1:Single-key input

- Input: AH = 1
- Output: AL = ASCII code if character key is pressed

  = 0 if non-character key is pressed
- Example
  - MOV AH, 1
  - INT 21h

Function 2:Single-key output

- Input: AH = 2

  DL = ASCII code of the display character or control character

- Example:

```
MOV AH, 2
MOV DL, ‘?’
INT 21h
```

Function 9: Displaying a string

- Input: DX = Offset address of the string

  the string must end with a ‘$’ character
- It expects the offset address of the character string to be in DX.
- For that we use LEA (Load Effective Address) instruction
- Example

```
MSG    DB   ‘Hello!!!$’
LEA    DX, MSG
MOV AH, 9
INT 21h
```

Structure of an Assembly Language Program

```
TITLE PGM1_1: SAMPLE PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
……………………
.CODE
MAIN PROC
……………….
MAIN ENDP
END MAIN
```

First Program: displaying a character

```
.MODEL SMALL
.STACK 100H
.CODE
MAIN PRAOC

    ;display a charcter

    MOV AH, 2 ;display character function
    MOV DL, '?' ; character is ?
    INT 21H ; display it

   ; return to DOS

    MOV AH,4CH
    INT 21H

MAIN ENDP
END MAIN
```

Thank You