Introduction to IBM PC Assembly Language

Outline

- Assembly language syntax
- Program data
- Variables
- Named constants
- A few basic instructions
- Translation of high-level language to assembly language
- Program structure
- Input and output instructions

Assembly Language Syntax

- Assembler
  - Assembly language programs are translated into machine language instructions by assembler.
  - Assembly language code is not case sensitive.
- Statements
  - Program consists of statements one per line.
  - Two types: instruction and assembler directive.
  - Statements have four fields:
    - name operation operand(s) comment
    - Instruction- START: MOV CX,5 ;initialize counter.
    - Assembler directive- MAIN PROC

Name Field

- Used for instruction labels, procedure names and variable names.
- Assembler translates name into memory address.
- Names can be 1 to 31 characters long and may consist of letters, digits 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.

Name Field

Legal names	Illegal Names
COUNTER_1	TWO WORDS
@character	2abc
.TEST	A45.28
DONE?	YOU&ME

Operation Field

- Contains symbolic (Operation code).
- Assembler translates 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 code is not translated into machine code, it only tells assembler to do something.
- Example: PROC pseudo-op is used to create a procedure.

Operand Field

- Specifies the data that are to be acted on by the operation.
- 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 one to the contents of AX.
ADD WORD1, 2; two operands, adds value two to the contents of memory
                 location WORD1.
```

Comment Field

- Say something about what the statement does.
- Marked by semicolon in the beginning.
- Assembler ignores anything after semicolon.
- Optional.
- 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.

Variables

- Each variable has a data type and is assigned a memory address by the program.
- Possible Values:
  - 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.
- Mainly three types
  - Byte variables.
  - Word variables.
  - Arrays.

Data Defining Pseudo-Ops

Examples	Bytes	Description	Pseudo-ops
var1 DB 'A' Var2 DB ? array1 DB 10, 20,30,40	1	Define Byte	**DB**
var2 DW 'AB' array2 DW 1000, 2000	2	Define Word	**DW**
Var3 DD -214743648	4	Define Double Word	**DD**
Var DQ ?	8	Define Quad Word	**DQ**
Var DT ?	10	Define Ten Bytes	**DT**

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
- XCHG
- ADD
- SUB
- INC
- DEC
- NEG

MOV

- Transfer data:
  - Between registers.
  - Between register and a memory location.
  - Move a number directly to a register or a memory location.
- Syntax:
  - MOV destination, source
- Example:
  - MOV AX, WORD1

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

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

	AH	AL	BH	BL
Before	1Ah	00h	00h	05h
After	05h	00h	00h	1Ah

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

- To add contents of:
  - Two registers.
  - A register and a memory location.
  - A number to a register.
  - A number to a memory location.
- Syntax:
  - ADD destination, source
- Example:
  - ADD WORD1, AX

	Before	**After**
AX	01BCh	01BCh
WORD1	0523h	06DFh

SUB

- To subtract the contents of:
  - Two registers.
  - A register and a memory location.
  - A number from a register.
  - A number from a memory location.
- Syntax:
  - SUB destination, source
- Example:
  - SUB AX, DX

	Before	**After**
AX	0000h	FFFFh
DX	0001h	0001h

Legal Combinations of Operands for ADD and SUB

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

Memory to Memory Instruction

- ADD BYTE1, BYTE2   ;Illegal instruction
- Solution?
  - MOV AL, BYTE2

    ADD BYTE1, AL

INC

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

	Before	**After**
WORD1	0002h	0003h

DEC

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

	Before	**After**
BYTE1	FFFEh	FFFDh

NEG

- Used to negate the contents of destination.
- Replace the contents by its 2's complement.
- Syntax:
  - NEG destination
- Example:
  - NEG BX

	Before	**After**
BX	0002h	FFFEh

Translation of High-level Language to Assembly Language

- 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 A, AX	NEG A ADD A, 5
A = B – 2 x A	MOV AX, B SUB AX, A SUB AX, A MOV A, AX	

Program Structure

- Machine Programs consists of:
  - Stack.
  - Code.
  - Data.
- 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. Array may be larger than 64KB

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

Data Segment

- All variable definitions.
- Constant definitions are often made here.
- Use .DATA directive.
- For Example:
  - .DATA

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

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.
- Inside a code segment instructions are organized as procedures.

The Format of a Code

```
.MODEL SMALL
.STACK 100h
.DATA
    ;data definition go here.
.CODE
MAIN PROC
    ;instructions go here.
MAIN ENDP
    ;other procedures go here.
END MAIN
```

Input and Output Instructions

Function Number	Routine	Code
1	Single key input	MOV AH,1 ;input key function INT 21H ;ASCII code in AL
2	Single character output	MOV AH,2 ;display character function MOV DL,'?';character is ? INT 21H; display character
9	Character string output	MSG DB 'HELLO$'

LEA Instruction

- It means Load Effective Address.
- To get the offset address of the character string in DX we use LEA instruction.
- Syntax:
  - LEA destination, source
- Destination is a general register and source is a memory location.
- Example:
  - LEA DX, MSG