Basic Elements of Assembly Language

Contents of Lecture:

References for Lecture:

KIP R. IRVINE, Assembly Language for x86 Processors, 7th Edition, Chapter 3: Assembly Language Fundamentals

Integer Literals:

- ❖ An integer literal (also known as an **integer constant**) is made up of an optional leading sign, one or more digits, and an optional radix character that indicates the number's base:

  [{+ | - }] digits [radix]

- ❖ **For example:**
  - ✓ 26 is a valid integer literal. It doesn't have a radix, so we assume it's in decimal format. If we wanted it to be 26 hexadecimal, we would have to write it as 26h.
  - ✓ Similarly, the number 1101 would be considered a decimal value until we added a "b" at the end to make it 1101b (binary).
  - ✓ Here are the possible radix values:

h	hexadecimal	r	encoded real
q/o	octal	t	decimal (alternate)
d	decimal	y	binary (alternate)
b	binary		

- ❖ If no radix given, assumed to be decimal
- ❖ A hexadecimal literal beginning with a letter must have a leading zero to prevent the assembler from interpreting it as an identifier

❖ **Correct examples:**
- ✓ 26
- ✓ 26d
- ✓ 11010011b
- ✓ 42q
- ✓ 42o
- ✓ 1Ah
- ✓ 0A3h
- ✓ -6455
- ✓ 456h
- ✓ 0AAAAh

❖ **Wrong examples:**
- ✓ FFFFh
- ✓ 1,234
- ✓ 0ab

Integer Expressions:

❖ A constant integer expression is a mathematical expression involving integer literals and arithmetic operators.

❖ Each expression must evaluate to an integer. The arithmetic operators are listed in following Table:

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

❖ What the order of the following expressions?
- ✓ 4 + 5 * 2 ………………………………
- ✓ 12 -1 MOD 5 ………………………………
- ✓ -5 + 2 ………………………………
- ✓ (4 + 2) * 6 ………………………………

❖ The following are examples of valid expressions calculate their values:

Expression	Value
`16 / 5`	
`-(3 + 4) * (6 - 1)`	
`-3 + 4 * 6 - 1`	
`25 mod 3`	

Character Literals:

❖ A character literal is a single character enclosed in single or double quotes.
❖ The assembler stores the value in memory as the character's binary ASCII code.
❖ Examples are:
  - ✓ 'A'
  - ✓ "d"

So, when you write the character constant 'A', it's stored in memory as the number 65 (or 41 hex).

String Literals:

❖ A string constant is a sequence of characters (including spaces) enclosed in single or double quotes.

❖ **Examples:**
  - ✓ 'ABC'
  - ✓ 'X'
  - ✓ "Good night, Gracie"
  - ✓ '4096'

❖ Just as character constants are stored as integers, we can say that string literals are stored in memory as sequences of integer byte values. So the string literal "ABCD" contains the four bytes 41h, 42h, 43h, and 44h.

Reserved Words:

- ❖ Reserved words have special meaning and can only be used in their correct context.
- ❖ Reserved words are not case-sensitive.
- ❖ There are different types of reserved words:
  - ✓ Instruction, such as MOV, ADD, and MUL
  - ✓ Register names
  - ✓ Directives, which tell the assembler how to assemble programs
  - ✓ Attributes, which provide size and usage information for variables and operands.
    - ▪ Examples are DB and DW
  - ✓ Operators, used in constant expressions
  - ✓ Predefined symbols, such as @data, which return constant integer values at assembly time

Identifiers:

- ❖ An identifier is a programmer-chosen name. It might identify a variable, a constant, a procedure, or a code label.

- ❖ There are a few rules on how they can be formed:
  - ✓ They may contain between 1 and 247 characters.
  - ✓ They are not case sensitive.
  - ✓ The first character must be a letter (A..Z, a..z), underscore (_), @ , ?, or $.
  - ✓ Subsequent characters may also be digits.
  - ✓ An identifier cannot be the same as an assembler reserved word.

- ❖ Examples
  - ✓ Var1, Count, $first, _main, MAX, open_file, myFile, xVal, _12345

Directives:

- ❖ A directive is a command embedded in the source code that is recognized and acted upon by the assembler.

- ❖ Directives do not execute at runtime, but they let you define variables, macros, and procedures.

- ❖ Directives are not case sensitive.
  - ✓ For example, .data, .DATA, and .Data are equivalent

- ❖ Different assemblers have different directives
  - ✓ TASM != MASM, for example

Instructions:

- An instruction is a statement that becomes executable when a program is assembled into machine code by assembler and executed at runtime by the CPU.

- An instruction contains (fields):

Label	(optional)
Operation	(required)
Operand/s	(depends on the instruction)
Comment	(optional)

- Basic syntax
  [Label:] Operation [operand/s] [; comment]

- **Labels:**
  - A label is an identifier that acts as a place marker for instructions and data.
  - A label placed just before an instruction implies the instruction's address. Used to give one instruction or more instructions a name.

  - There are two types of labels:
    - Data labels: (A data label identifies the location of a variable in data segment)
    - Code labels: (where instructions are located)

  - A label in the code area of a program must end with a colon (:) character.
  - Code labels are used as targets of jumping and looping instructions.

  - Label names follow the same rules we described for identifiers
  - Label never starts by numbers.
  - We cannot use space at the name.
  - We cannot use the dot (.) unless at the beginning.
  - Not case sensitive.

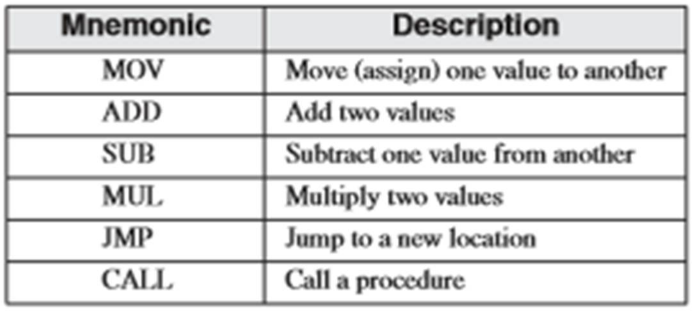
  - **Example** for accept name:
    Start: – counter: - @character: – sum_of_digits: - $1000: – done?: - .test:

  - **Example** for reject name:
    Two words:
    2abc:
    a45.ab:

❖ **<u>Operation (instruction):</u>**

- ➢ An instruction is a short word that identifies an instruction. This means the instruction that will be executed.

- ➢ Assembly language instruction mnemonics such as mov, add, and sub provide hints about the type of operation they perform.
- ➢ Following are examples of instruction mnemonics

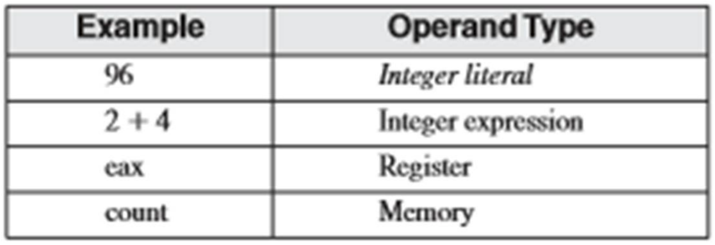


Mnemonic	Description
MOV	Move (assign) one value to another
ADD	Add two values
SUB	Subtract one value from another
MUL	Multiply two values
JMP	Jump to a new location
CALL	Call a procedure

❖ **<u>Operand/s:</u>**

- ➢ An operand is a value that is used for input or output for an instruction (number of operands depend on operation).

- ➢ Assembly language instructions can have between zero and three operands, each of which can be a register, memory operand, integer expression, or input–output port.



Example	Operand Type
96	*Integer literal*
2 + 4	Integer expression
eax	Register
count	Memory

Assembly language instructions having varying numbers of operands:

- ➢ The STC instruction has no operands:

```
stc                 ; set Carry flag
```

- ➢ The INC instruction has one operand:

```
inc  eax            ; add 1 to EAX
```

- ➢ The MOV instruction has two operands:

```
mov count,ebx       ; move EBX to count
```

- ➢ The IMUL instruction has three operands

```
imul eax,ebx,5      ;EBX is multiplied by 5, and the product is
                    stored in the EAX register.
```

❖ Comment:

- Comments are an important way for the writer of a program to communicate information about the program’s design to a person reading the source code (Comments are good to explain the program's purpose).

Comments can be specified in two ways:

- **Single-line comments**: beginning with a semicolon character (;). All characters following the semicolon on the same line are ignored by the assembler

- **Block comments (Multi-line comments)**:
  - ✓ begin with COMMENT directive and a programmer-chosen character
  - ✓ end with the same programmer-chosen character

- Here is an example

```
COMMENT   !
          This line is a comment
          This line is also a comment
!

COMMENT   &
          This line is a comment
          This line is also a comment
&
```