CSC 3210 Computer Organization and Programming

CHAPTER3: ASSEMBLY LANGUAGE FUNDAMENTALS

Outline

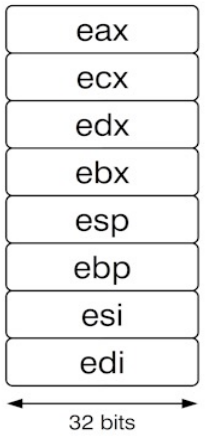
- **Basic Elements of Assembly Language**
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants
- 64-Bit Programming

Basic Language Elements

- **Parts of assembly program:**
  - Main procedure (entry point)
  - eax register
  - **mov & add** instruction (eax=11)
  - INVOKE (call, unix) ExitProcess, 0
  - Comments begin with **semicolon ;**

AddTwo program:

```
1: main PROC
2:    mov eax,5                ; move 5 to the eax register
3:    add eax,6                ; add 6 to the eax register
4:
5:    INVOKE ExitProcess,0     ; end the program
6: main ENDP
```

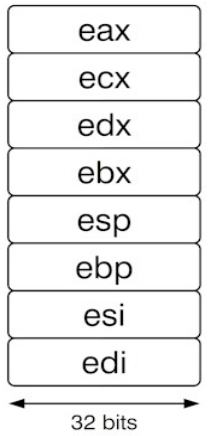


Basic Language Elements

- No output?
- **Debugger**!
  - See what is in CPU registers and flags
    - Need to manage data representation and instruction formats
- Can NOT run this program

AddTwo program:

```
1: main PROC
2:    mov eax,5                ; move 5 to the eax register
3:    add eax,6                ; add 6 to the eax register
4:
5:    INVOKE ExitProcess,0     ; end the program
6: main ENDP
```



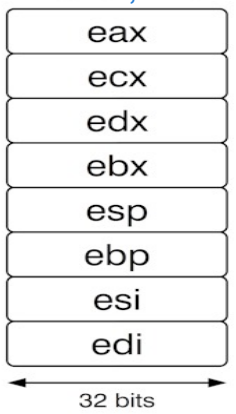
AddTwo program:

```
1: main PROC
2:    mov eax,5                ; move 5 to the eax register
3:    add eax,6                ; add 6 to the eax register
4:
5:    INVOKE ExitProcess,0     ; end the program
6: main ENDP
```

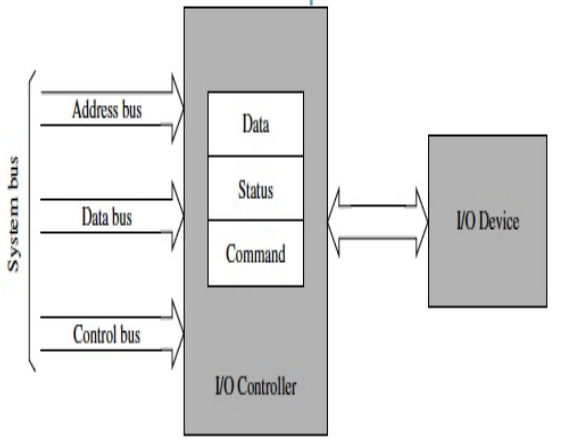
- Instructions **operand** can be
- **register**,
- **memory location** or
- **immediate**

  - **Immediate operand:**
    - Even though the data are in memory, it is **located in the code segment**, **not** in the data segment
    - **Signed** integer
    - Has a limited **range**

mov eax,5



INVOKE scanf
INVOKE printf



Lecture-8-Chapter3-Assembly-Language-fundamentals-with-...

AddTwo program:

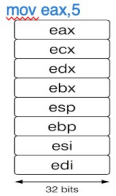
```
1: main PROC
2:    mov eax,5                ; move 5 to the eax register
3:    add eax,6                ; add 6 to the eax register
4:
5:    INVOKE ExitProcess,0     ; end the program
6: main ENDP
```

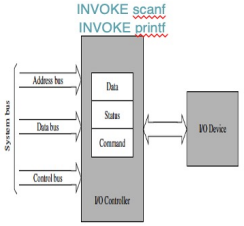
- Instructions operand can be
  - register,
  - memory location or → variable
  - immediate

Hard coded values

- **Immediate operand:**
  - Even though the data are in memory, it is **located in the code segment**, **not** in the data segment
  - **Signed** integer
  - Has a limited **range**

5, 6, 7, ...





6

Click to add notes

windows10_fall21 [Running]

Project4

```
mov eax, 15
mov ebx, 14
mov ecx, 13
mov edx, 12

INVOKE ExitProcess, 0
in ENDP   ; end of main function

D main    ; end of whole program
```

MOV destination, source
ADD destination, source
SUB destination, source

d = d - s

mov eax, 10

Hard coded value
immediate values

No issues found Ln: 18 Ch: 13 Col: 16 TABS CRLF

Call Stack

Name	Lang
Project4.exe!main() Line 21	Un...
[External Code]	
ntdll.dll![Frames below may be incorrect and/or missing, no sy...	Un...

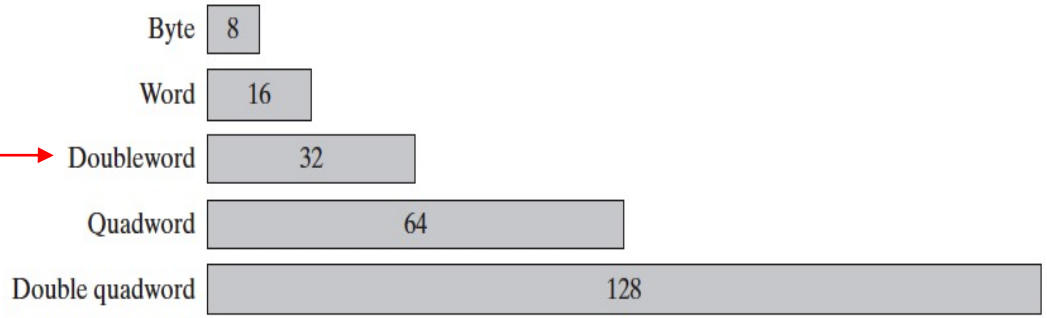
Add to Source Control

Basic Language Elements: Adding a Variable

- **Declarations**
  - Identify Code and Data areas (directives)
    - **.data**
    - **.code**
  - Segments (how about .stack)
- Variable declaration (sum)
  - DWORD (keyword): size of 32 bits
  - No checking in what gets into inside the variable (not like C/Java)
  - Must **defined** and **initialized** to **zero**

- Syntax rules imposed by MS-**MASM**

```
1: .data                        ; this is the data area
2: sum DWORD 0                  ; create a variable named sum
3:
4: .code                        ; this is the code area
5: main PROC
6:    mov eax,5                 ; move 5 to the eax register
7:    add eax,6                 ; add 6 to the eax register
8:    mov sum,eax
9:
10:   INVOKE ExitProcess,0      ; end the program
11: main
```



Basic Elements of Assembly Language

- **Integer constants**
- Integer expressions
- Character and string constants
- Reserved words and identifiers
- Directives
- instructions
  - Labels
  - Mnemonics
  - Operands
- Comments
- Examples

Basic Elements of Assembly Language: Integer Constants

- Optional **leading** **+** or **–** sign
- binary, decimal, hexadecimal, or octal digits
- Common radix characters:
  - h – hexadecimal
  - d – decimal
  - b – binary
  - o/q – octal
  - r – encoded real

➢ **Examples**:

```
26                  ; decimal
26d                 ; decimal
11010011b           ; binary
42q                 ; octal
42o                 ; octal
1Ah                 ; hexadecimal
0A3h                ; hexadecimal
```

Basic Elements of Assembly Language: Integer Expressions

- Operators and **precedence** levels:

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

```
4 + 5 * 2          Multiply, add
12 -1 MOD 5        Modulus, subtract
-5 + 2             Unary minus, add
(4 + 2) * 6        Add, multiply
```

Basic Elements of Assembly Language: Integer Expressions

- **Examples**:

Expression	Value
`16 / 5` **Integer division?**	3
`-(3 + 4) * (6 - 1)`	-35
`-3 + 4 * 6 - 1`	20
`25 mod 3`	1

Suggestion: Use parentheses in expressions **to clarify the order** of **operations** so you don't have to remember precedence rules.

Basic Elements of Assembly Language: Character and String Constants

- **Enclose character** in single or double quotes
  - 'A', "x"
  - ASCII character = 1 byte
- **Enclose strings** in single or double quotes
  - "ABC"
  - 'xyz'
  - Each character occupies **a single byte**
- **Embedded** quotes:
  - 'Say "Goodnight," Gracie'

Basic Elements of Assembly Language: Real Numbers

How about REAL numbers?

Following are examples of valid decimal reals:

```
2.
+3.0
-44.2E+05
26.E5
```

2r

Basic Elements of Assembly Language: Reserved Words

- Reserved words **cannot be used as identifiers**
  - Instruction mnemonics, directives, type attributes, operators, predefined symbols
  - See MASM reference in Appendix A

- Instruction mnemonics, 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 BYTE and WORD
- Operators, used in constant expressions
- Predefined symbols, such as @data, which return constant integer values at assembly time

Basic Elements of Assembly Language: Identifiers

- 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.

- The following names **are legal**, but **not as desirable**
  - _ @ ? $

_lineCount $first @myFile

Basic Elements of Assembly Language: Directives

- **Commands** that are **recognized** and **acted** upon by the assembler
  - Not part of the Intel instruction set
  - Used to declare **code**, **data** areas, select **memory model**, declare **procedures**, etc.
  - **not case sensitive**, **Ex:**

.data,
.DATA,
.Data
are equivalent.

Basic Elements of Assembly Language: Directives

- **Directive .vs Instruction:**

```
myVar DWORD 26

mov eax,myVar
```

- Directive:
DWORD directive tells the assembler to reserve space in the program for a doubleword variable.

- Instruction:
MOV instruction, executes at runtime, copying myVar to the EAX register

- Different assemblers have different directives
  - NASM not the same as MASM, for example

All **assemblers** for **Intel processors** share the **same instruction set**, **BUT** they usually have different sets of directives.
Ex: Microsoft assembler's **REPT** directive **is not recognized** by some other **assemblers**.

Basic Elements of Assembly Language: Directives

See Appendix A (p-591, section A5) for more directives types

- label **PROC**
  Marks start of a procedure block called label.
- name **ENDP**
  Marks the end of procedure name previously begun with PROC.
- **INVOKE** expression [[, arguments]]
  Calls the procedure at the address given by expression, passing the arguments on the stack or in registers according to the standard calling conventions of the language type.
- **.DATA**
  indicates the start of initialized data
- **.CODE**
  indicates the start of a code segment
- **.STACK** indicates the start of a stack segment

```
1: .data                        ; this is the data area
2: sum DWORD 0                  ; create a variable named sum
3:
4: .code                        ; this is the code area
5: main PROC
6:    mov eax,5                 ; move 5 to the eax register
7:    add eax,6                 ; add 6 to the eax register
8:    mov sum,eax
9:
10:   INVOKE ExitProcess,0      ; end the program
11: main ENDP
```

Basic Elements of Assembly Language: Directives

Appendix A contains a useful reference for directives and operators.

Basic Elements of Assembly Language: Instructions

- **Assembled** into machine code by assembler
- **Executed** at runtime by the CPU
- We use the Intel IA-32 instruction set
- An instruction contains:
  - Label (optional)
  - Mnemonic (required)
  - Operand (depends on the instruction)
  - Comment (optional)

```
        [label:] mnemonic [operands] [;comment]
repeat:   inc      result   ;increment result by 1
```

The **label** repeat can be used to **refer to this particular statement.**

Instructions: Labels

- **Act as place markers**
  - Marks the address (offset) of **code** and **data**
- Follow identifier rules

1. Code label

- target of **jump** and **loop** instructions
- **ex1**: **L1:** (followed by colon)

- **Ex2** :

```
target:
        mov ax,bx
        ...
        jmp target
```

A **code label** can **share the same line with an instruction**, or it can be **on a line by itself**:

```
L1: mov ax,bx
```

Instructions: Labels

2. Data label

- must be unique
- **Ex**: one data → **count DWORD 100** (not followed by colon)
- **Ex**: multiple data →

```
array  DWORD 1024, 2048
       DWORD 4096, 8192
```

or

```
array  DWORD 1024, 2048, 4096, 8192
```

Instructions: Mnemonics

- Instruction **Mnemonics**
  - examples: MOV, ADD, SUB, MUL, JMP, CALL

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