National University of Sciences and Technology

School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Lab Engineer: MS. Bareera Anam

|  |  |
| --- | --- |
| **CS-235: Computer Organization and Assembly Language** | |
| **Faculty Member** | **Semester** |
|  | 4th |
| **Class/Section** | **Date** |
| BSCS-13\_E | Monday, 3 March 2025 |

**Lab 3: Memory Access in Assembly Language**

**Objective**

The aim of this lab is to:

* Understand data transfer and manipulation instructions.
* Analyze memory access operations in RISC vs. CISC architectures.
* Study the Fetch-Decode-Execute cycle in memory operations.
* Explore the impact of pipelining on memory access.

**Instruction Cycle and Memory Access Overview**

Memory access is a crucial component of the instruction cycle. Each instruction in assembly language follows the steps:

* **Fetch**: The CPU retrieves the instruction from memory.
* **Decode**: The control unit interprets the instruction.
* **Execute**: The CPU performs the operation, including memory access if needed.
* **Write-back**: The result is stored in a register or memory.

**Comparison: RISC vs. CISC Memory Access**

* **RISC (Reduced Instruction Set Computer)**:
* Uses a load/store architecture where memory access is limited to load/store instructions.
* Instructions are uniform in length and execute in a single cycle.
* **CISC (Complex Instruction Set Computer)**:
* Instructions can access memory directly, reducing the number of instructions required.
* Complex addressing modes increase execution time per instruction.

**Memory Data Representation**

* BYTE (8-bit) → Stores individual bytes.
* WORD (16-bit) → Stores two consecutive bytes.
* DWORD (32-bit) → Stores four consecutive bytes.
* TEXTEQU → Defines a text constant.

 Understand Memory Offsets

* Each value is stored at a specific location (offset) in memory.
* The memory address increases sequentially as we store different variables.

 How Memory is Represented in Hexadecimal

* The mbyte values (05h, 12h, 100h) will be stored in three consecutive bytes.
* The mword values (50h, 60h) will be stored in two consecutive words.
* The mdouble value (0A0908070h) takes up four consecutive bytes.
* The greetings string is stored as ASCII characters followed by 0 (null termination).

|  |  |  |  |
| --- | --- | --- | --- |
| **offset** | **Content** | **Offset** | **Content** |
| 00 |  | 10 |  |
| 01 |  | 11 |  |
| 02 |  | 12 |  |
| 03 |  | 13 |  |
| 04 |  | 14 |  |
| 05 |  | 15 |  |
| 06 |  | 16 |  |
| 07 |  | 17 |  |
| 08 |  | 18 |  |
| 09 |  | 19 |  |
| 0A |  | 1A |  |
| 0B |  | 1B |  |
| 0C |  | 1C |  |
| 0D |  | 1D |  |
| 0E |  | 1E |  |
| 0F |  | 1F |  |
|  |  | 20 |  |

**Exercise 1:**

In the memory list shown, insert the values of the variables as declared below, in hexadecimal format:

.data

mbyte BYTE 05,12,100

Align 2

mword WORD 50h, 60h

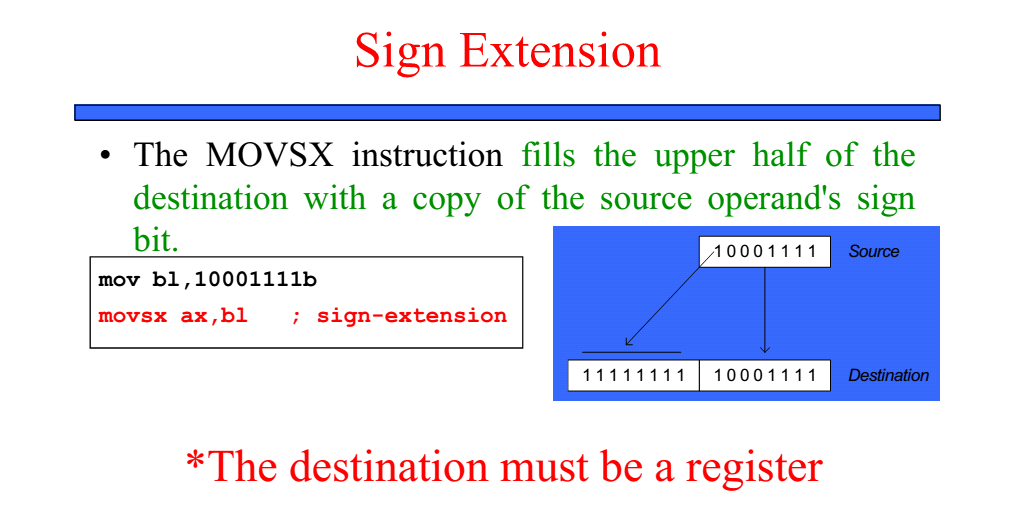
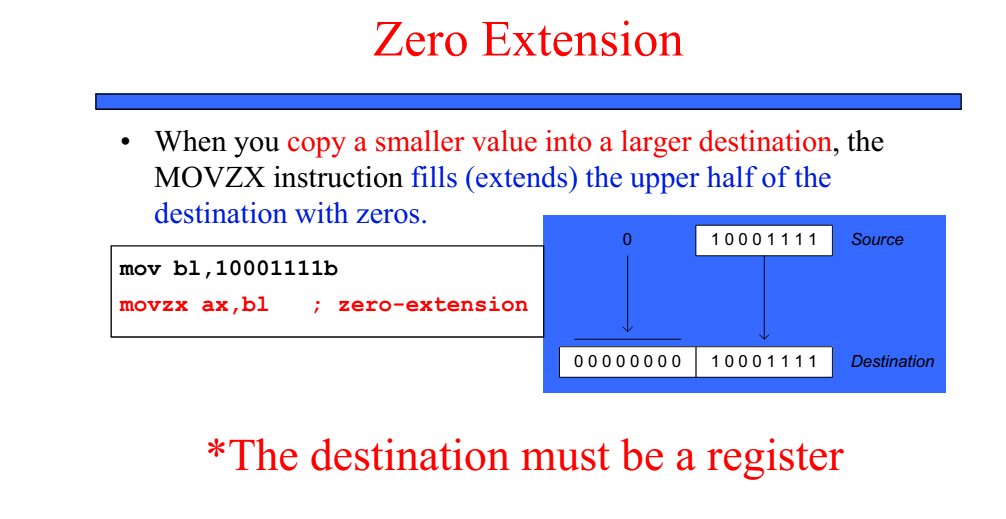
mdouble DWORD 0A0908070h

greetings BYTE "Hi There",0

Response TEXTEQU <'Hi, Thanks.'>

Reply BYTE Response

Note: No ASCII table is to be used, wait to fill the text character codes after the exercise where the textstrings have been used, and the .lst file can be used to see these codes). **DO NOT FORGET**



**Fetch-Decode-Execute Analysis**

Step-by-Step Execution of Each Instruction

* Fetch: The CPU retrieves the instruction from memory.
* Decode: The CPU identifies the instruction and determines the necessary data (operands).
* Execute: The instruction is carried out.

**Exercise 2:**

Without writing any code, write down the expected contents of the register after the instruction is executed:

* Mov al,mbyte-1 ;AL=
* Mov sx ax, mbyte+1 ;AX=

**Exercise 3: Displaying Memory and Registers**

In this exercise we will learn about and use some new procedures that can be called to display register or memory contents.

Writing strings to display:

Declare/define the string: mystring BYTE “How are you?”0

Get the offset of string into EDX: mov edx, offset mystring

Call the procedure to display: call **writestring**

(**Note** that writestring only works with EDX holding the offset)

Writing register constents to display:

Get the contents to display in AL,AX, or EAX.

Ensure the bits not used are set/reset so as to improve readability.

Call the procedure to display: a. **writeint** to print in decimal format

b. **writehex** to print in hexadecimal format

c. **writebin** to print in binary format

d. **writechar** to print a character, the LSD of EAX

(**Note** that all these writexxx display EAX contents)

A call to **crlf** adds carraige return followed by a linefeed, eg., call crlf.

**Step1:** Write code to get Byte No 2 of mbyte into AL and byte No 1 of mbyte into AH. Ensure that the higher order bits of EAX are cleared. Display EAX to verify that the correct bytes are in the locations specified. Use all four write procedures to see the various output formats, with a call to crlf after each writexxx to make the output easy to read.

(**Note** Use the data given in Exercise No. 1)

Note down the outputs bvelow:

* EAX=
* EAX=
* EAX=
* ------= ;Explain what is it.

**Step2:** Extend program to display the length and size of the string variable “greetings” in decimal format, and then print the first string

**Step3:** Display the second string defined by the TEXTEQU operator