Lab: 09

Department of Computer Science

Iqra University Islamabad

Computer Organization and Assembly Language

Maqsood Ahmed

ID: 38186

4.3.1 Examples on 32-bit Addressing Modes

The provided assembly code demonstrates different 32-bit memory addressing modes and the LEA (Load Effective Address) instruction. Here's the annotated code and the expected outcomes:

Assembly Code: `addressing.asm`

```
TITLE Memory Addressing Examples       (File: addressing.asm)
.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc

.data
arrayB  BYTE  "COE 205",0
arrayW  WORD  100h,200h,300h, 400h
arrayD  DWORD 01234567h,89ABCDEFh

.code
main PROC
    ; Direct Memory Addressing
    mov al, arrayB           ; same as [arrayB]
    mov ah, arrayB[5]        ; same as [arrayB+5]
    mov bx, arrayW[2]        ; same as [arrayW+2]
    mov ecx,[arrayD]         ; same as arrayD
    mov edx,[arrayD+2]       ; same as arrayD[2]

    ; Register Indirect Addressing
    mov ecx, OFFSET arrayB + 3
    mov edx, OFFSET arrayW + 1
    mov bx, [ecx]            ; address in [ecx]
    mov al, [edx]            ; address in [edx]

    ; Based Addressing
    mov edx, 4
    mov al, arrayB[edx]
    mov bx, arrayW[edx]
    mov ecx,arrayD[edx]

    ; Scaled Indexed Addressing
    mov esi, 1
    mov arrayB[esi*2], 'S'
    mov arrayW[esi*2], 102h
    mov arrayD[esi*4], 0

    ; Load Effective Address (LEA)
    lea eax, arrayB
    lea ebx,[eax + LENGTHOF arrayB]
    lea ecx,[ebx + esi*8]
    lea edx, arrayD
```

```
    exit
main ENDP
END main
```

Execution Steps and Predicted Values:

1. **Direct Memory Addressing:**

```
mov al, arrayB
```

- **AL = 'C' = 43h**

```
mov ah, arrayB[5]
```

- **AH = '0' = 30h**

```
mov bx, arrayW[2]
```

- **BX = 0200h**

```
mov ecx, [arrayD]
```

- **ECX = 01234567h**

```
mov edx, [arrayD+2]
```

- **EDX = CDEF0123h (Note: This seems unusual because `[arrayD+2]` usually means offset by 2 bytes, not 2 DWORDs. This implies the address was interpreted differently.)**

2. **Register Indirect Addressing:**

```
mov ecx, OFFSET arrayB + 3
```

- **ECX = 404003h (Assuming base address 404000h for arrayB)**

```
mov edx, OFFSET arrayW + 1
```

- **EDX = 404009h (Assuming base address 404008h for arrayW)**

```
mov bx, [ecx]
```

- **BX = 0320h**

```
mov al, [edx]
```

- **AL = 01h**

3. **Based Addressing:**

```
mov edx, 4
mov al, arrayB[edx]
```

- **AL = '2' = 32h**

```
mov bx, arrayW[edx]
```

- **BX = 0300h**

```
mov ecx, arrayD[edx]
```

- **ECX = 89ABCDEFh**

4. **Scaled Indexed Addressing:**

```
mov esi, 1
mov arrayB[esi*2], 'S'
```

- **arrayB[2] = 'S'**

```
mov arrayW[esi*2], 102h
```

- **arrayW[2] = 102h**

```
mov arrayD[esi*4], 0
```

- **arrayD[4] = 00000000h**

5. **Load Effective Address (LEA):**

```
lea eax, arrayB
```

- **EAX = 404000h**

```
lea ebx, [eax + LENGTHOF arrayB]
```

- **EBX = 404008h (Assuming LENGTHOF arrayB is 8 bytes)**

```
lea ecx, [ebx + esi*8]
```

- **ECX = 404010h**

```
lea edx, arrayD
```

- **EDX = 404010h (Assuming arrayD is at 404010h)**

4.4 LOOP Instruction

The LOOP instruction uses ECX as a counter, decrementing it each iteration and jumping to the specified label if ECX is not zero.

4.5 Copying a String

Assembly Code: `CopyStr.asm`

```
TITLE Copying a String                    (File: CopyStr.asm)
; Demonstrates LOOP instruction and array indexing

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0)

.code
main PROC
    mov  esi, 0     ; used to index source and target
    mov  ecx, SIZEOF source
L1:
    mov  al, source[esi]      ; get a character from source
    mov  target[esi], al      ; store it in the target
    inc  esi      ; increment index
    loop L1      ; repeat for entire string

    exit
main ENDP
END main
```

Execution Analysis:

- **Value of the target string after loop L1:**

The target string will be a copy of the source string: `"This is the source string",0`**.**

- **Number of iterations for loop L1:**

The number of iterations equals the size of the source string, including the null terminator. The length of `"This is the source string"` **is 24 characters + 1 null terminator = 25 iterations.**