Addressing Modes

The addressing modes are the supported methods for accessing a value in memory
The basic addressing modes are:

- Register --> source of data is a CPU register
- Immediate --> source of data is an immediate value
- Memory --> source of data is a memory location

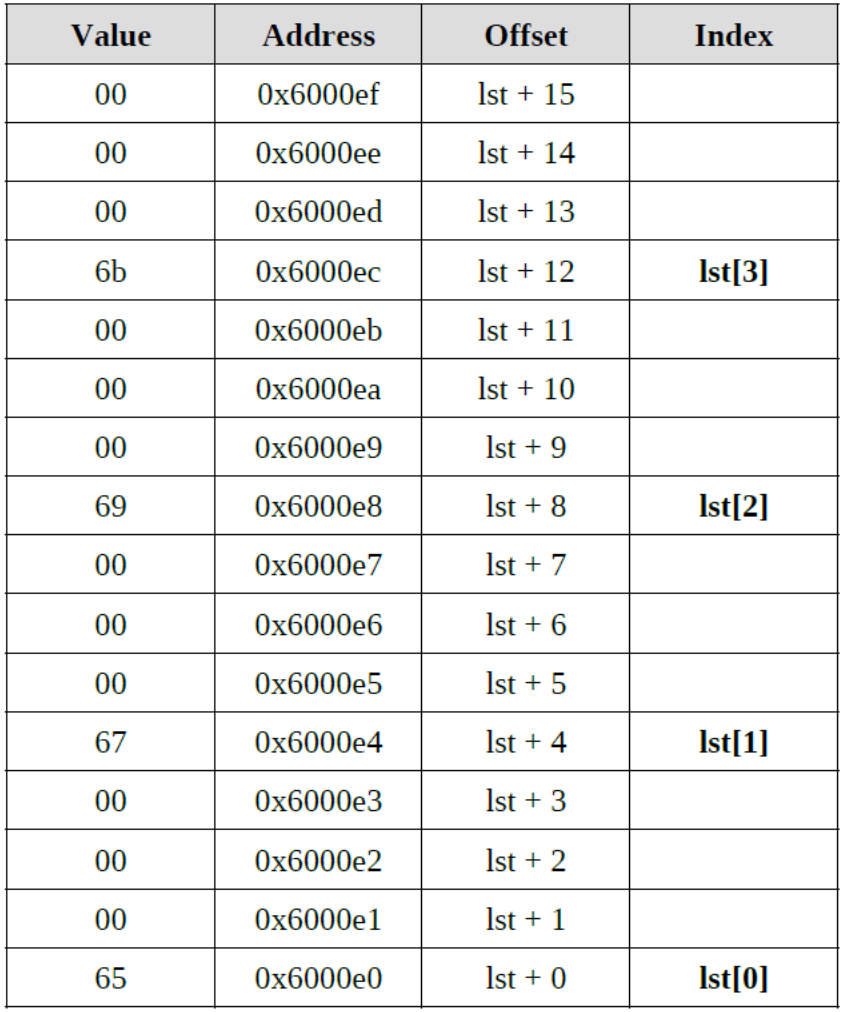
Examples:

```
    mov     eax, ebx                ; register addressing mode
    mov     eax, 123                ; immediate addressing mode
    mov     eax, dword [dNum]          ; memory addressing mode
```

Addressing Modes

Array

lst dd 101, 103, 105, 107



Value	Address	Offset	Index
00	0x6000ef	lst + 15	
00	0x6000ee	lst + 14	
00	0x6000ed	lst + 13	
6b	0x6000ec	lst + 12	**lst[3]**
00	0x6000eb	lst + 11	
00	0x6000ea	lst + 10	
00	0x6000e9	lst + 9	
69	0x6000e8	lst + 8	**lst[2]**
00	0x6000e7	lst + 7	
00	0x6000e6	lst + 6	
00	0x6000e5	lst + 5	
67	0x6000e4	lst + 4	**lst[1]**
00	0x6000e3	lst + 3	
00	0x6000e2	lst + 2	
00	0x6000e1	lst + 1	
65	0x6000e0	lst + 0	**lst[0]**

lst →

← **Base address**

```
; access lst[0]
mov eax, dword [lst]

; or
mov rbx, list
mov eax, dword [rbx]
```

Addressing Modes

Offset consideration

Separation between two consecutive elements in array depends on data size

Examples

```
lst     dd  101, 103, 105, 107            ;double word --> 4 byte
msg     db  ‘a’, ‘b’, ‘c’             ;byte --> 1 byte
table   dw  0xffab, 0x7fff, 0x17ff    ;word --> 2 byte
```

Addressing Modes

Accessing array using offset

Value	Address	Offset	Index
00	0x6000ef	lst + 15	
00	0x6000ee	lst + 14	
00	0x6000ed	lst + 13	
6b	0x6000ec	lst + 12	**lst[3]**
00	0x6000eb	lst + 11	
00	0x6000ea	lst + 10	
00	0x6000e9	lst + 9	
69	0x6000e8	lst + 8	**lst[2]**
00	0x6000e7	lst + 7	
00	0x6000e6	lst + 6	
00	0x6000e5	lst + 5	
67	0x6000e4	lst + 4	**lst[1]**
00	0x6000e3	lst + 3	
00	0x6000e2	lst + 2	
00	0x6000e1	lst + 1	
65	0x6000e0	lst + 0	**lst[0]**

lst

Base address

```
; load base address
mov rbx, lst
; access lst[1]
mov eax, dword [lst+4]
; access lst[2]
mov eax, dword [lst+8]
; access lst[3]
mov eax, dword [lst+12]
```

Addressing Modes

Accessing array using incremental offset

Value	Address	Offset	Index
00	0x6000ef	lst + 15	
00	0x6000ee	lst + 14	
00	0x6000ed	lst + 13	
6b	0x6000ec	lst + 12	**lst[3]**
00	0x6000eb	lst + 11	
00	0x6000ea	lst + 10	
00	0x6000e9	lst + 9	
69	0x6000e8	lst + 8	**lst[2]**
00	0x6000e7	lst + 7	
00	0x6000e6	lst + 6	
00	0x6000e5	lst + 5	
67	0x6000e4	lst + 4	**lst[1]**
00	0x6000e3	lst + 3	
00	0x6000e2	lst + 2	
00	0x6000e1	lst + 1	
65	0x6000e0	lst + 0	**lst[0]**

lst →

← **Base address**

```
; load base address
mov rbx, lst
; load offset
mov rsi, 4
; access lst[1]
mov eax, dword [lst+rsi]
; access lst[2]
add rsi, 4
mov eax, dword [lst+rsi]
; access lst[3]
add rsi, 4
mov eax, dword [lst+rsi]
```

Addressing Modes

The general format of memory addressing is as follows:

[baseAddr + (indexReg * scaleValue) + displacement]

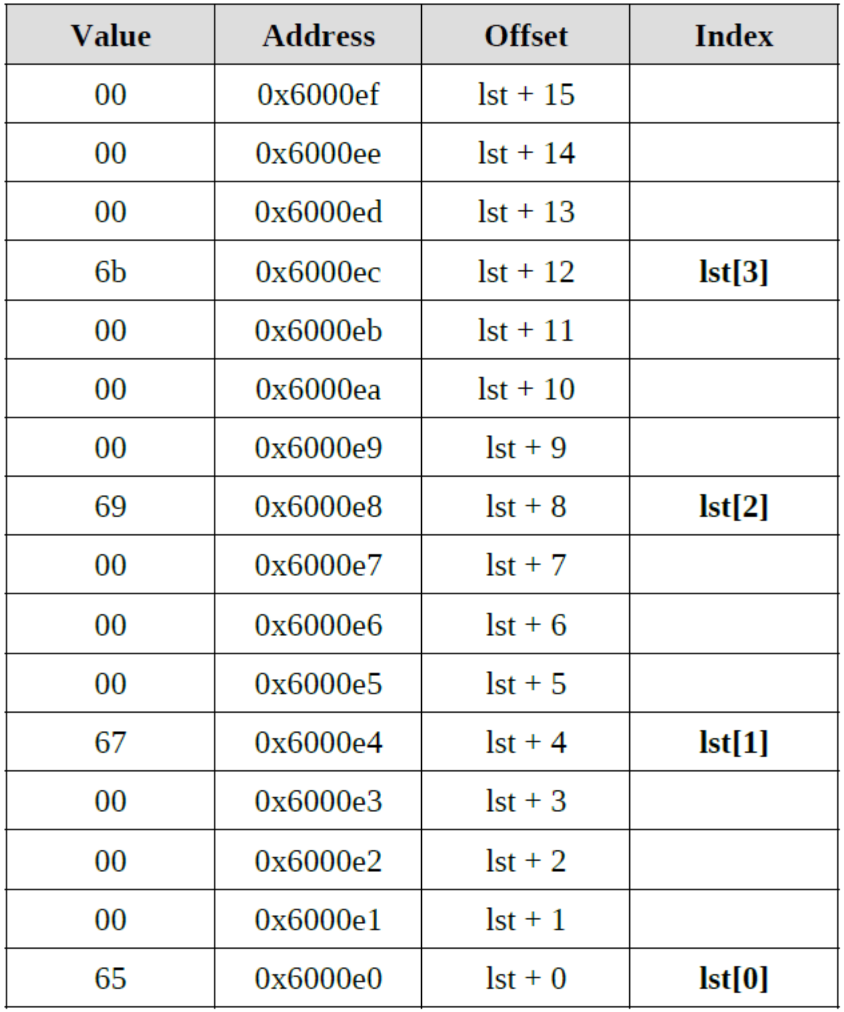
BaseAddr --\> a register or variable name
IndexReg --\> register used for indexing
ScaleValue --\> immediate value
Displacement --\> immediate value

Examples:

```
mov eax, dword [var1]
mov rax, qword [rbx+rsi]
mov ax, word [lst+4]
mov bx, word [lst+rdx+2]
mov rcx, qword [lst+(rsi*8)]
mov al, byte [buff-1+rcx]
mov eax, dword [rbx+(rsi*4)+16]
```

Addressing Modes

Accessing array (*revisited*)



Value	Address	Offset	Index
00	0x6000ef	lst + 15	
00	0x6000ee	lst + 14	
00	0x6000ed	lst + 13	
6b	0x6000ec	lst + 12	**lst[3]**
00	0x6000eb	lst + 11	
00	0x6000ea	lst + 10	
00	0x6000e9	lst + 9	
69	0x6000e8	lst + 8	**lst[2]**
00	0x6000e7	lst + 7	
00	0x6000e6	lst + 6	
00	0x6000e5	lst + 5	
67	0x6000e4	lst + 4	**lst[1]**
00	0x6000e3	lst + 3	
00	0x6000e2	lst + 2	
00	0x6000e1	lst + 1	
65	0x6000e0	lst + 0	**lst[0]**

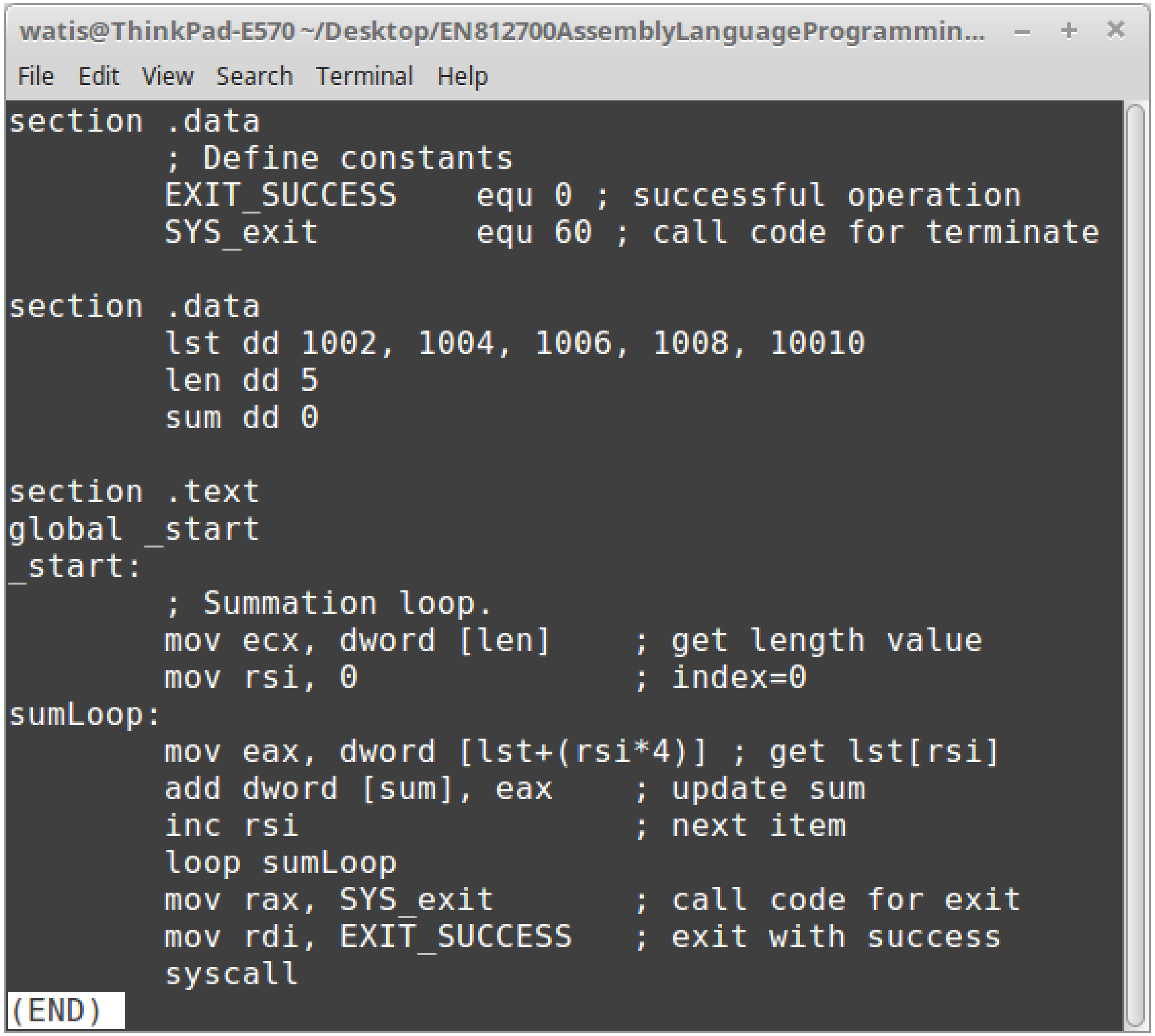
lst →

← **Base address**

```
; load base address
mov rbx, lst
; load offset
mov rsi, 1
; access lst[1]
mov eax, dword [lst+4*rsi]
; access lst[2]
inc rsi
mov eax, dword [lst+4*rsi]
; access lst[3]
inc rsi
mov eax, dword [lst+4*rsi]
```

Addressing Modes

List summation


watis@ThinkPad-E570 ~/Desktop/EN812700AssemblyLanguageProgrammin...
File Edit View Search Terminal Help

```
section .data
        ; Define constants
        EXIT_SUCCESS    equ 0 ; successful operation
        SYS_exit        equ 60 ; call code for terminate

section .data
        lst dd 1002, 1004, 1006, 1008, 10010
        len dd 5
        sum dd 0

section .text
global _start
_start:
        ; Summation loop.
        mov ecx, dword [len]    ; get length value
        mov rsi, 0              ; index=0
sumLoop:
        mov eax, dword [lst+(rsi*4)] ; get lst[rsi]
        add dword [sum], eax    ; update sum
        inc rsi                 ; next item
        loop sumLoop
        mov rax, SYS_exit       ; call code for exit
        mov rdi, EXIT_SUCCESS   ; exit with success
        syscall
(END)
```

Assignment #2

1. Implement a 1D array, then find min, max and average values

2. Design a scheme to represent 2D array in memory

- **how array elements are stored in memory**
- **how to access array element by specifying (row, col)**