ASM Language	**ใบงานที่ 2** Data Addressing Modes **สัปดาห์ที่** **ระยะเวลา** ในชั้นเรียน **01418331 แอสแซมบลีและสถาปัตยกรรมคอมพิวเตอร์** ชื่อ ....บุญฑภานต์ ศิริกมลทิพย์.......... รหัสนิสิต ....6430200418..........

เวลา : 2 ชั่วโมง

จุดประสงค์การเรียนรู้

1. ทำการเรียนรู้การเขียนโปรแกรมภาษา Assembly
2. ทำการเรียนรู้การกำหนดชนิดข้อมูลและ Symbolic

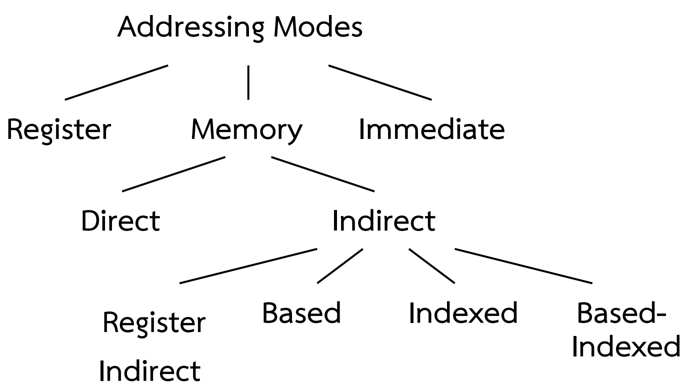
เนื้อหาการเรียนรู้

Data Addressing Modes

เนื้อหา

1. Data Addressing Modes

คำสั่ง assembly language จำเป็นต้องทำความเข้าใจเกี่ยวกับสถานที่เก็บของข้อมูลว่าเป็นตัวถูกกระทำ (operand) ว่าเป็นต้นทาง (source operands) หรือ ปลายทาง (destination operands) ซึ่งการกระทำในลักษณะนี้ถูกเรียกว่า **data addressing mode** และสามารถจัดลำดับการแสดงเป็นไดอะแกรมตามผังนี้:



Register addressing คือ เมื่อ register จะถูกใช้เป็น source หรือ destination ของ operand ในส่วนนี้มีผลกับ addressing mode มากเพราะ registers เป็นการดำเนินการภายใน processor และการเข้าถึงสามารถเข้าถึงได้อย่างรวดเร็ว

Immediate addressing คือ เมื่อค่า immediate (ค่าคงที่) ที่ถูกใช้สำหรับเป็น source operand ได้เท่านั้นไม่สามารถถูกกำหนดเป็น destination operand ได้

Memory addressing คือ ถูกใช้กำหนดตำแหน่ง (address) ของ source และ destination operands ภายใน memory. เป็นโหมด (mode) ที่ต้องกำหนดรายละเอียดหรือตำแหน่งของค่า operand ที่จะถูกนำมาหรือนำไปจัดเก็บ ซึ่งสามารถถูกกำหนดเป็นแบบ **direct และ indirect** memory addressing

Direct memory addressing คือ เมื่อตำแหน่งของ memory operand ถูกกำหนดการเข้าถึงโดยตรงจากชื่อ (name) ตัวอย่าง:

```
mov sum, eax          ; sum is a variable in memory
```

Direct memory addressing เป็นการเข้าถึง/เปลี่ยนแปลงค่าใน memory ได้โดยตรง แต่สำสำหรับการเข้าถึง arrays หรือข้อมูลแบบมีโครงสร้าง (data structures) โดย address ของ elements ของ array จำเป็นต้องใช้ register เป็นเหมือน pointer เพื่อชี้ array elements ซึ่งในส่วนนี้จะถูกเรียกว่า **indirect memory addressing ที่**สามารถกำหนดประเภทเป็น **register indirect**, **based**, **indexed**, และ **based-indexed**, ขึ้นกับว่า address ของ memory operandได้ถูกกำหนดไว้อย่างไร ซึ่งโดยทั่วไป general memory address สามารถกำหนดได้ตามนี้:

Address = [BaseReg + IndexReg * Scale + Disp]

ในที่นี้การเข้าถึงแบบ general indirect memory addressing ถูกเรียกว่า **based-indexed**, เพราะได้รวม **base register** กับ **index register** และ **displacement**.

indirect memory addressing modes ส่วนอื่นมี: **register indirect**, **based**, และ **indexed** ดูรูปแบบได้จากตัวอย่าง

Register Indirect Addressing:	**Address = [Reg]**
Based Addressing:	**Address = [BaseReg + Disp]**
Indexed Addressing:	**Address = [IndexReg * Scale + Disp]**

IA-32 processor architecture สำหรับ 32-bit addressing และใช้ได้เหมือน 16-bit addressing ได้ด้วย สำหรับ 16- bit addressing modes ถูกใช้ครั้งแรกกับ 8086 processor ที่ซึ่งใช้ได้เฉพาะ 16-bit registers เท่านั้น, 16-bit addressing modes จะใช้ BX และ BP สำหรับ base register และ SI และ DI เพื่อทำ index register ไม่มี scale factor และ displacements ถูกจำกัดที่ 16 bits.

ข้อดีของ IA-32 processor architecture, registers ถูกขยายจาก 16 bits ถึง 32 bits ซึ่ง 32-bit addressing modes เป็นการเพิ่มอีก 16-bit addressing มี 32-bit general-purpose register (EAX, EBX, ECX, EDX, ESI, EDI, EBP, or ESP) ที่สามารถถูกใช้เป็นเหมือน base register ได้ และ general-purpose register ESP ยังสามารถถูกใช้เป็น index registerได้ในบางครั้ง **scale factor** ที่ใช้มี 1, 2, 4, หรือ 8 ถูกเพิ่มให้กับ indexed-addressing กับ displacements เป็นการขยายให้ถึง 32-bits.

เปรียบเทียบระหว่าง 16-bit และ 32-bit addressing modes สรุปได้ตามตารางด้านล่างนี้:

	16-bit Addressing	32-bit Addressing
Base Register	BX, BP	EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
Index Register	SI, DI	EAX, EBX, ECX, EDX, ESI, EDI, EBP
Scale Factor	None	1, 2, 4, 8
Displacement	0, 8, 16 bits	0, 8, 32 bits

1.1 Examples on 32-bit Addressing Modes

โปรแกรมต่อไปนี้แสดงให้เห็นถึง 32-bit memory addressing modes และคำสั่ง LEA (Load Effective Address) ซึ่งเป็นคำสั่งนำตำแหน่ง **address** ที่เก็บค่าของ source operand ไปเก็บไว้ใน destination.

LEA destination, source Load Effective Address ของ *source* ไปเก็บที่ *destination*

ตัวอย่างการนำค่าตำแหน่งเก็บไว้ที่ Register ที่เหมือนกับ คือ OFFSET ของ variable มาเก็บที่ Register โดยใช้คำสั่ง mov ดูได้จากด้านล่าง

lea eax, variable => mov eax, OFFSET variable

ให้เขียนโปรแกรม addressing.asm ด้านล่างนี้ เพื่อนำไปประกอบกับการฝึกปฏิบัติในหัวข้อทัดไป

```
section .data
arrayB    db      "ASM 02418331",10,0
arrayW    dw      100h,200h,300h, 400h
arrayD    dd 01234567h,89ABCDEFh

section .text
        global _start
_start:
      mov al, byte [arrayB]         ; same as [arrayB]
      mov ah, byte [arrayB+5]       ; same as [arrayB+5]
      mov bx, word [arrayW+2]       ; same as [arrayW+2]
      mov ecx,dword [arrayD]        ; same as arrayD
      mov edx,dword [arrayD+2]      ; same as arrayD[2]

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

; Based Addressing
      mov edx, 4
      mov al, byte [arrayB+edx]
```

```
mov bx, word [arrayW+esi+edx]
mov ecx,dword [arrayD+edx]

mov esi, 1
mov byte [arrayB+(esi*2)], 'S'
mov word [arrayW+(esi*2)], 102h
mov dword [arrayD+(esi*4)], 0

move ax. 1
int 0x80
```

3.2 ให้ทำการ Assemble and Link 'addressing.asm'

3.3 ให้ทำการ Debugger เพื่อหาค่าที่ถูกประมวลผลจาก './addressing'

ทำการบันทึกค่าของ registers และ memory variables ในโปรแกรม *addressing.asm*. ลงในกรอบสี่เหลี่ยมหลังจากการประมวลผลแล้วตอบเป็นเลข hexadecimal

โดยใช้การทำจากโปรแกรม Windows Debugger ให้เปิด source file *addressing.asm* จาก **File** menu หากโปรแกรมไม่กำลังเปิดในขณะนี้ หากต้องการ registers และ memory โดยเลือกได้จาก **View** menu ใน Memory window เขียนชื่อตัวแปรแรกของโปรแกรม *arrayB* ลงใน Virtual address

Direct Memory Addressing

1) al = 0x41	2) ah = 0x32	3) bx = 0x200
4) ecx = 0x1234567	5) edx = 0xcdef0123	

Register Indirect Addressing

6) ecx = 0x402003	7) edx = 0x40200f
8) bx = 0x3020	9) al = 0x1

Based Addressing

11) al = 0x30	12) bx = 0x300	13) ecx = 0x89abcdef