THE X86 ASSEMBLY LANGUAGE

Assembly line

[label] mnemonic [operands] [; comment]

```
; This is a comment
   jmp label1 ; This is also a comment
   add eax, ebx
label1:
   sub edx, 32
```

Addressing

Register Addressing Mode

The operand is in a register.

```
    mov EAX, EBX ; move EBX to EAX
```

Immediate Addressing Mode

The operand is part of the instruction.

```
    mov EAX, 132 ; move 132 to EAX
```

Direct addressing mode

The operand is in memory, and the address is specified as an offset.

```
a_letter DB 'c'       ; Allocate one byte of memory,
initialize it to 'c'.
    mov AL, a_letter ; Move data at memory location
"a_letter" into AL.
                ; I.e. move 'c' to AL.
```

Addressing

Register Indirect Addressing

The operand is found at the memory location specified by the register.
The register is enclosed in square bracket.

```
    mov EAX, ESP ; Move stack pointer to EAX
    mov EBX, [ESP] ; Move value at top-of-stack to EBX
```

Indirect Addressing Mode

The offset of the data is in one of the eight general-purpose registers.

```
.DATA
array DD 20 DUP (0) ; Array of 20 integers initialized to zero
.CODE
    mov ECX, OFFSET array ; Move starting address of 'array' to ECX
```

Addressing

Based Addressing

One of the eight general-purpose registers acts like a base register in computing the effective address of an operand. The address is computed by adding a signed (8-bit or 32-bit) number to the base address.

```
mov ECX, 20[EBP] ; ECX = memory[EBP + 20]
```

Indexed Addressing

The effective address is computed by:

signed displacement + (Index * scale factor)

```
add AX, [DI + 20]   ; AX = AX + memory[DI + 20]
mov AX, table[ESI*4]; AX = memory[ OFFSET table + ESI * 4 ]
add AX, table[SI]   ; AX = AX + memory[ OFFSET table + SI * 1]
```

scale factor: 1, 2, 4 or 8

index register: EAX, EBX, ECX, EDX, ESI, EDI, EBP

Addressing

Based-Indexed Addressing

In this addressing mode, the effective address is computed as:

Base + (Index * Scale factor) + signed displacement.

```
mov EAX, [EBX+ESI] ; AX = memory[EBX + (ESI * 1) + 0]
mov EAX, [EBX+EPI*4+2] ; AX = memory[EBX + (EPP * 4) + 2]
```

base register: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP

index register: EAX, EBX, ECX, EDX, ESI, EDI, EBP

scale factor: 1, 2, 4 or 8

signed displacement: 8, 16 or 32-bit value

PTR Directive

- Resolves ambiguous operator size

```
mov [ESI], al ; Store a byte-size value in memory
          ; location pointed by ESI

mov [ESI], 5  ; Error: operand must have the size specified

mov [ESI], BYTE PTR 5  ; Store 8-bit value



mov     ax, WORD PTR [num] ; Load a word-size value from a DWORD
```

Practical example with NASM

- addressing.asm

```
_start:
        mov eax, 0xdeadbeaf             ; eax=0xdeadbeaf
        mov ebx, dword [0xdeadbeaf]     ; ebx <-- (dword)[0xffffffffdeadbeaf]   ... SIGSEGV
        mov eax, valore                 ; eax=0x6000ec -> 0xdeadbeaf
        mov ebx, [valore]               ; ebx=0xdeadbeaf
        mov eax, ebx                    ; eax=0xdeadbeaf
        mov eax, [ebx]                  ; ebx <-- (dword)[0xffffffffdeadbeaf]   ... SIGSEGV
        mov eax, [ebx+2]                ; ebx <-- (dword)[0xffffffffdeadbeaf + 2]   ... SIGSEGV
        mov eax, [ebx*4+0xdeadbeaf]     ; ebx <-- (dword)[0xffffffffdeadbeaf*4 + 0xdeadbeaf]   ...
                                        ; SIGSEGV
        mov eax, [edx + ebx*4 + 42]
```

Let's check

- Install gdb (gcc)
- Install Peda: https://github.com/longld/peda
  - Ubuntu (https://nuc13us.wordpress.com/2015/02/01/installing-gdb-peda-in-ubuntu/)
- Install NASM:
  - sudo apt-get clean
  - sudo apt-get remove nasm-rdoff
  - sudo apt-get install nasm

  If it gives you trouble, run this instead:
  - sudo apt-get clean
  - sudo dpkg -r nasm-rdoff
  - sudo apt-get install nasm

Exercise

- nasm -f elf64 -o addr.o addressing.asm
- ld -o addr addr.o
- gdb addr
- break _start
- si
  - Hit return
- set $pc= *addr*