

Access

Young W. Lim

2017-01-30 Mon

1 Introduction

- References
- IA32 Operand Forms
- Data Movement Instructions

"Self-service Linux: Mastering the Art of Problem Determination", Mark Wilding
"Computer Architecture: A Programmer's Perspective", Bryant & O'Hallaron

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

Operand Types

- 1 Immediate Operand Type
- 2 Register Operand Type
- 3 Memory Reference Type

1) Immediate Operand Type

- constant values
- \$ followed by integer number
- only one or two bytes of 4 bytes integer

2) Register Operand Type

- denote the content of a register
 - 8 32-bit registers for double word operations
 - 8 8-bit registers for a byte operation
 - Ea : an arbitrary register a
 - $R[Ea]$ the value of an Ea register
 - view the set of registers as an array R
 - indexed by register identifiers

3) Memory Reference Type

- access some memory location
 - according to the computed address
 - effective address
- view the memory as a large array of bytes
- $Mb[Addr]$: the by b-byte value stored in memory starting at Addr
- addressing modes : allowing different forms of memory references
 - Imm : immediate offset
 - Eb : a base register
 - Ei : an index register
 - s : a scale factor (1, 2, 4, 8)

Addressing Modes

Imm	$M[\text{Imm}]$	Absolute
(Ea)	$M[R[\text{Ea}]]$	Indirect
Imm (Eb)	$M[\text{Imm} + R[\text{Eb}]]$	Base + displace
(Eb, Ei)	$M[R[\text{Eb}] + R[\text{Ei}]]$	Indexed
Imm (Eb, Ei)	$M[\text{Imm} + R[\text{Eb}] + R[\text{Ei}]]$	Indexed
(, Ei, s)	$M[R[\text{Ei}] * s]$	Scaled Indexed
Imm (, Ei, s)	$M[\text{Imm} + R[\text{Ei}] * s]$	Scaled Indexed
(Eb, Ei, s)	$M[R[\text{Eb}] + R[\text{Ei}] * s]$	Scaled Indexed
Imm (Eb, Ei, s)	$M[\text{Imm} + R[\text{Eb}] + R[\text{Ei}] * s]$	Scaled Indexed

IA32 Integer Registers (1)

(a, c, d, b, si, di)

a: %eax(32), %ax(16), %ah(8), %al(8)

c: %ecx(32), %cx(16), %ch(8), %cl(8)

d: %edx(32), %dx(16), %dh(8), %dl(8)

b: %ebx(32), %bx(16), %bh(8), %bl(8)

si: %esi(32), %si(16)

di: %edi(32), %di(16)

sp: %esp(32), %sp(16) (stack pointer)

bp: %ebp(32), %bp(16) (frame pointer)

IA32 Integer Registers (2)

32-bit Registers

Caller Save Registers

`%eax, %ecx, %edx`

Callee Save Registers

`%ebx, %esi, %edi`

Stack Frame Registers

`%esp, %ebp`

16-bit Registers

`%ax, %cx, %dx, %bx, %si, %di, %sp, %bp`

8-bit Registers

`%ah, %al, %ch, %cl, %dh, %dl, %bh, %bl`

Data Movement Instructions

```
movl    S, D    ; S -> D    ; move double word
movw    S, D    ; S -> D    ; move word
movb    S, D    ; S -> D    ; move byte
movsbl  S, D    ; SignExt(S) -> D ; move sign-extended byte
movzbl  S, D    ; ZeroExt(S) -> D ; move zero-extended byte
pushl   S        ; R[%esp]-4 -> R[%esp]; S -> M[R[%esp]] ; push
popl    D        ; M[R[%esp]] -> D; R[%esp]+4 -> R[%esp] ; pop
```

movl examples

```
movl  $0x4050, %eax      ; immediate -> register
movl  %ebp, %esp        ; register   -> register
movl  (%edi, %ecx), %eax ; memory    -> register
movl  $-17,m (%esp)     ; immediate -> memory
movl  %eax, -12(%ebp)   ; register  -> memory
```

stack examples (1)

```
pushl %ebp
```

```
subl $4, %esp
```

```
movl %ebp, (%esp)
```

```
push %eax
```

```
movl (%esp), %eax
```

```
addl $4, %esp
```

stack examples (2)

initially

%eax: 0x123

%edx: 0

%esp: 0x108

push %eax

%eax: 0x123

%edx: 0

%esp: 0x104 <- 0x108

pop %edx

%eax: 0x123

%edx: 0x123 <- 0

%esp: 0x108 <- 0x014

Comparing byte movements

```
movb %dh, %al  
movsbl %dh, %eax  
movzbl %dh, %eax
```

pointer examples (1)

```
int exchange(int *xp, int y) {  
    int x = *xp;  
    *xp = y;  
    return x;  
}
```

```
movl 8(%ebp), %eax    ; get xp  
movl 12(%ebp), %edx   ; get y  
movl (%eax), %ecx    ; get x at *xp  
movl %edx, (%eax)    ; store y at *p  
movl %ecx, %eax      ; set x as return address
```


pointer examples (2)

```
movl 8(%ebp), %eax
movl 12(%ebp), %edx
movl (%eax), %ecx
movl %edx, (%eax)
movl %ecx, %eax
```

- xp parameter at offset 8
- y parameter at offset 12
- relative the address in %ebp
- xp to %eax
- y to %edx
- (%eax) dereferences xp : *xp
- any function returning an integer or pointer value by placing the value in register %3ax

pointer examples (3)

- pointers are simply addresses
- dereferencing a pointer
 - store that pointer in a register
 - using this register, perform indirect memory reference
- local variables are kept in registers rather than stored in memory location
- Register access is much faster

function prototype

```
movl 8(%ebp), %edi
movl 12(%ebp), %ebx
movl 16(%ebp), %esi
movl (%edi), %eax
movl (%ebx), %edx
movl (%esi), %ecx
movl %eax, (%ebx)
movl %edx, (%esi)
movl %ecx, (%edi)
```