Access

Young W. Lim

2020-04-09 Thr

Outline

1. Access
    - Based on
    - IA32 Operand Forms
    - Data Movement Instructions
    - Data Movement Examples

Based on

1. "Self-service Linux: Mastering the Art of Problem Determination",

Mark Wilding

1. "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.

Compling 32-bit program on 64-bit gcc

- `gcc -v`
- `gcc -m32 t.c`
- `sudo apt-get install gcc-multilib`
- `sudo apt-get install g++-multilib`
- `gcc-multilib`
- `g++-multilib`
- `gcc -m32`
- `objdump -m i386`

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
    `%eax, %ecx, %edx, %ebx, %esi, %edi, %esp, %ebp`
  - 8 8-bit registers for a byte operation
    `%ah, %al, %ch, %cl, %dh, %dl, %bh, %bl`
  - `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 (1)

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

Addressing Modes (2)

Imm		M[Imm]	Absolute
Imm	(Eb)	M[Imm + R[Eb]]	Base + displace
Imm	(Eb, Ei)	M[Imm + R[Eb] + R[Ei]]	Indexed
Imm	(, Ei, s)	M[Imm + R[Ei]*s]	Scaled Indexed
Imm	(Eb, Ei, s)	M[Imm + R[Eb] + R[Ei]*s]	Scaled Indexed
	(Ea)	M[R[Ea]]	Indirect
	(Eb, Ei)	M[R[Eb] + R[Ei]]	Indexed
	(, Ei, s)	M[R[Ei]*s]	Scaled Indexed
	(Eb, Ei, s)	M[R[Eb] + R[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 32-bit double word (`l`)
`movw S, D`	`S` → `D`	move 16-bit word (`w`)
`movb S, D`	`S` → `D`	move 8-bit byte (`b`)
`movsbl S, D`	`SignExt(S)` → `D`	move sign-extended byte (`sbl`)
`movzbl S, D`	`ZeroExt(S)` → `D`	move zero-extended byte (`zbl`)

`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 eamples

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

Comparing byte movements

```
movb     %dh,   %al    ; move 8-bit byte (b)
movsbl   %dh,   %eax   ’ move sign-extended byte (sbl)
movzbl   %dh,   %eax   ; move zero-extended byte (zbl)
```

stack eamples (1)

`push %ebp`	`subl $4, %esp`
	`movl %ebp, (%esp)`
`pop %eax`	`movl (%esp), %eax`
	`addl $4, %esp`

stack eamples (2)

	initially	push %eax	pop %edx
%eax	0x123	0x123	0x123
%edx	0	0	0x123
%esp	0x108	0x104	0x108

pointer eamples (1)

```
int a = 4;
int  = exchange(&a, 3);
printf("a=%d b=%d\n", a, b);
```

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

pointer eamples (2)

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

```
movl   8(%ebp), %eax     ; M[%ebp+8]  -> %eax    get xp
movl   12(%ebp), %edx    ; M[%ebp+12] -> %edx    get y
movl   (%eax), %ecx      ; M[%eax]    -> %ecx    get x at *xp
movl   %edx, (%eax)      ; %edx       -> M[%eax] store y at *xp
movl   %ecx, %eax        ; %ecx       -> %eax    set x as return value
```

Stack frame structure

%ebp+12	2nd argument
%ebp+8	1st argument
%ebp+4	return address
%ebp	saved %ebp

pointer eamples (3)

```
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 eamples (4)

- pointers are simply addresses
- derefencing 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)
```

```
void func(
          int *xp,
          int *yp,
          int *zp)
```