Lecture 3

<2016-04-04 Mon>

Contents

1 Machine Level Programming

- Architecture: ISA (Instruction Set Architecture)

1.1 Assembly / Machine Code

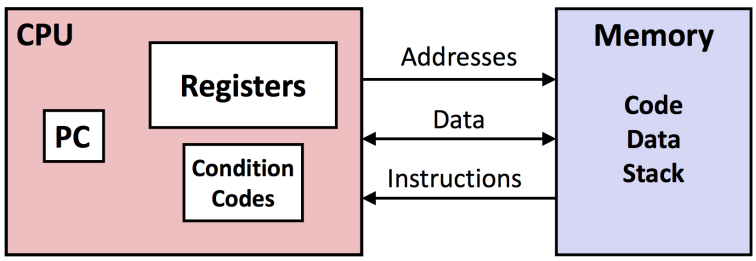

Figure 1: architecture

- PC (program counter): address of next instruction
- register files

- condition codes
- memory

compile C code into assembly code: `gcc -Og -S`
disassemble object code: `objdump -d`
disassemble in `gdb`: `disassemble <function_name>`

1.1.1 x86-64 Integer Registers

in x86-64, each register is of 8 bytes

Table 1: example of registers

8 bytes	lower 4 bytes	note
`%rax`	`%eax`	
`%rsp`	`%esp`	**stack pointer**, reserved for special use

1.1.2 Moving Data

- Operand type
  - immediate: constant integer data
    - e.g. `$0x400, $-533`
  - register: integer register
    - e.g. `%rax`
    - `%rsp`, stack pointer, reserved for special use
  - memory: 8 consecutive bytes of memory at address given by register
    - e.g. `(%rax)`

```
int temp, temp1, temp2;
int *p;
```

1. example: memory-memory transfer

   memory-memory transfer connot be done with a single instruction

   ```
   long temp1, temp2;
   temp1 = temp2;
   ```

Table 2: movq operand combinations

source	destination	assembly code	C analog (not exactly transliterated)
immediate	register	`movq $0x4,%rax`	`temp = 0x4`
	memory	`movq $0x4,(%rax)`	`*p = 0x4`
register	register	`movq %rax,%rdx`	`temp2 = temp1`
	memory	`movq %rax,(%rdx)`	`*p = temp`
memory	register	`movq (&rax),%rdx`	`temp = *p`

```
        ;; %rax stores the address of temp1
        ;; %rbx stores the address of temp2
movq    (%rbx), %rcx            ;load temp2 into register %rcx
movq    %rcx,  (%rax)           ;write temp2 into temp1
```

1.1.3 Memory Addressing Modes

- simple memory addressing modes
  - normal:
    - `(R)` : register
    - `MEM [ REG[R] ]` : access memory located at the address stored by register R
      - Register R specifies memory address
      - pointer dereference in C
      - e.g. `movq (%rax), %rdx`
  - displacement:
    - `D(R)` : R register, D displacement/offset
    - `MEM [ REG[R] + D ]` : access memory located at the address stored by register R and offset by D
      - D: constant displacement
      - e.g. `movq 8(%rax), %rdx`
- complete memory addressing modes
  - most general form
    - `D(Rb, Ri, S)`
      - D: constant displacement
      - Rb: base register
      - Ri: index register

        - S: scale 1, 2, 4, 8
      - `MEM [ REG[Rb] + S * REG[Ri] + D ]`
  - special case
    - `(Rb, Ri)` is equivalent to `0(Rb, Ri, 1)`
    - `D(Rb, Ri)` is equivalent to `D(Rb, Ri, 1)`
    - `(Rb, Ri, S)` is equivalent to `0(Rb, Ri, S)`

assembly	explanation	example
`(R)`	`MEM [ REG[R] ]`	`(%rax)`
`D(R)`	`MEM [ REG[R]+D ]`	`8(%rax)`
`D(Rb, Ri, S)`	`MEM [ REG[Rb] + S * REG[Ri] + D ]`	`8(%rdx,%rcx,4)`

Table 3: address computation example

`%rdx` `0xf000`
`%rcx` `0x0100`

memory addressing	complete memory addressing	computation	address
`0x8(%rdx)`	`0x8(0xf000,0,0)`	`0xf000 + 0x8`	`0xf008`
`(%rdx,%rcx)`	`0x0(0xf000,0x0100,1)`	`0xf000 + 0x0100`	`0xf100`
`(%rdx,%rcx,4)`	`0x0(0xf000,0x0100,4)`	`0xf000 + 4 * 0x0100`	`0xf400`
`0x80(,%rdx,2)`	`0x0(0,0xf000,2)`	`2 * 0xf000 + 0x80`	`0x1e080`

1. example: C code translated into assembly

```
void swap(long *xp, long *yp) {
  long x = *xp;
  long y = *yp;
  *xp = y;
  *yp = x;
}
```

register	value	type
`%rdi`	xp	address
`%rsi`	yp	address
`%rax`	x	long
`%rdx`	y	long

```
swap:
      movq      (%rdi), %rax        ;x = *xp
      movq      (%rsi), %rdx        ;y = *yp
      movq      %rdx,  (%rdi)       ;*xp = y
      movq      %rax,  (%rsi)       ;*yp = x
```

1.1.4 Address Computation Instruction

- `leaq src, dest`
  - `src` is address mode expression
  - set `dest` to address denoted by expression
  - **do not access memory**

1. Arithmetic Operations

format	computation	note
`addq src,dest`	`dest = dest + src`	
`subq src,dest`	`dest = dest - src`	
`imulq src,dest`	`dest = dest * src`	
`salq src,dest`	`dest = dest « src`	also called shlq
`sarq src,dest`	`dest = dest » src`	arithmetic
`shrq src,dest`	`dest = dest » src`	logical
`xorq src,dest`	`dest = dest ^ src`	
`andq src,dest`	`dest = dest & src`	
`orq src,dest`	`dest = dest | src`	
`incq dest`	`dest = dest + 1`	
`decq dest`	`dest = dest - 1`	
`negq dest`	`dest = -dest`	
`notq dest`	`dest = ~dest`	

instructions that ends with (instruction suffix)

- `b` : operate on lower-order 1 byte
- `w` : operate on lower-order 2 bytes
- `l` : operate on lower-order 4 bytes
- `q` : operate on lower-order 8 bytes

2. example1

```
long mul12(long x) {
  return x * 12;
}
```

converted to assembly

```
        ;; %rdi stores value of x
leaq    (%rdi,%rdi,2), %rax     ;%rax = x * 3
salq    $2, %rax                ;%rax = %rax << 2
```

3. example2

```
long arith(long x, long y, long z) {
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y*48;
  long t5 = t3+t4;
  long rval = t2*t5;
  return rval;
}
```

converted to assembly

register	value
%rdi	x
%rsi	y
%rdx	z, t4
%rax	t1, t2, rval
%rcx	t5

```
leaq    (%rdi,%rsi), %rax       ;t1 = x + y
addq    %rdx, %rax              ;t2 = t1 + z
leaq    (%rsi,%rsi,2), %rdx     ;t4 = y * 3
salq    $4, %rdx                ;t4 = t4 << 4
leaq    4(%rdi,%rdx), %rcx      ;t5 = t4 + x + 4
imulq   %rcx, %rax              ;rval = t2 * t5
```