Assembly: Data movement and arithmetic operations.

Yipeng Huang

Rutgers University

March 2, 2021
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
Looking ahead

Class plan

1. Quiz 6 is now an ungraded anonymous survey about PA2 and code review.
2. Today, Tuesday, 3/2: Data movement and arithmetic.
3. Thursday, 3/4: Assembly control flow.
4. Code review session for PA2 is the week of 3/8 - 3/12. TAs will take attendance to assign participation points.
5. Reading assignment for next three weeks: CS:APP Chapter 3.
Programming Assignment 3: Quickstart
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
Data movement instructions

Does unsigned / signed matter?

1. `void swap_uc ( unsigned char*a, unsigned char*b );`
2. `void swap_sc ( signed char*a, signed char*b );`

Swapping different data sizes

1. `void swap_c ( char*a, char*b );`
2. `void swap_s ( short*a, short*b );`
3. `void swap_i ( int*a, int*b );`
4. `void swap_l ( long*a, long*b );`
Unraveling the compilation chain

Turning C into Object Code

- Code in files `p1.c` `p2.c`
- Compile with command: `gcc -Og p1.c p2.c -o p`
  - Use basic optimizations (`-Og`) [New to recent versions of GCC]
  - Put resulting binary in file `p`

```
gcc -Og -S swap.c
```

```
objdump -d swap
```

Let’s go to CS:APP textbook lecture slides (05-machine-basics.pdf) slide 28
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
## Data size and x86 / x86-64 registers

### Assembly syntax

**Instruction Source, Dest**

```assembly
swap_l:
  movq (%rsi), %rax
  movq (%rdi), %rdx
  movq %rdx, (%rsi)
  movq %rax, (%rdi)
  ret
```

### Data type and mov operation

<table>
<thead>
<tr>
<th>swap</th>
<th>data type</th>
<th>mov operation</th>
<th>registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>swap_uc</td>
<td>unsigned char</td>
<td>movb (move byte)</td>
<td>%al, %dl</td>
</tr>
<tr>
<td>swap_sc</td>
<td>signed char</td>
<td>movb (move byte)</td>
<td>%al, %dl</td>
</tr>
<tr>
<td>swap_c</td>
<td>char</td>
<td>movb (move byte)</td>
<td>%al, %dl</td>
</tr>
<tr>
<td>swap_s</td>
<td>short</td>
<td>movw (move word)</td>
<td>%ax, %dx</td>
</tr>
<tr>
<td>swap_i</td>
<td>int</td>
<td>movl</td>
<td>%eax, %edx</td>
</tr>
<tr>
<td>swap_l</td>
<td>long</td>
<td>movq</td>
<td>%rax, %rdx</td>
</tr>
</tbody>
</table>
Data size and IA32, x86, and x86-64 registers

<table>
<thead>
<tr>
<th>data type</th>
<th>registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>%al, %dl</td>
</tr>
<tr>
<td>short</td>
<td>%ax, %dx</td>
</tr>
<tr>
<td>int</td>
<td>%eax, %edx</td>
</tr>
<tr>
<td>long</td>
<td>%rax, %rdx</td>
</tr>
</tbody>
</table>

Note the backward compatibility.

Some History: IA32 Registers

- **%eax**, %ax, %ah, %al
- **%ecx**, %cx, %ch, %cl
- **%edx**, %dx, %dh, %dl
- **%ebx**, %bx, %bh, %bl
- **%esi**, %si
- **%edi**, %di
- **%esp**, %sp
- **%ebp**, %bp

16-bit virtual registers (backwards compatibility)

Origin (mostly obsolete)
- accumulate
- counter
- data
- base
- source
- index
- destination
- stack
- pointer
- base
- pointer

Origin

Some History: IA32 Registers
Data size and IA32, x86, and x86-64 registers

### x86-64 Integer Registers

<table>
<thead>
<tr>
<th>data type</th>
<th>registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>%al, %dl</td>
</tr>
<tr>
<td>short</td>
<td>%ax, %dx</td>
</tr>
<tr>
<td>int</td>
<td>%eax, %edx</td>
</tr>
<tr>
<td>long</td>
<td>%rax, %rdx</td>
</tr>
</tbody>
</table>

Note the backward compatibility.

- Can reference low-order 4 bytes (also low-order 1 & 2 bytes)
Data size and IA32, x86, and x86-64 registers

Figure: x86-64 with SIMD extensions registers. Image credit: https://commons.wikimedia.org/wiki/File:Table_of_x86_Registers_svg.svg
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
Moving data between memory and the CPU registers

Assembly language and machine code is about:

- Moving data between memory and registers
- Processing data in the CPU

Assembly/Machine Code View

Programmer-Visible State
- PC: Program counter
  - Address of next instruction
  - Called “RIP” (x86-64)
- Register file
  - Heavily used program data
- Condition codes
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

Memory
- Byte addressable array
- Code and user data
- Stack to support procedures
### Addressing modes

#### movq Operand Combinations

<table>
<thead>
<tr>
<th>Source</th>
<th>Dest</th>
<th>Src,Dest</th>
<th>C Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imm</td>
<td>Reg</td>
<td>movq $0x4,%rax</td>
<td>temp = 0x4;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movq $-147,(%rax)</td>
<td>*p = -147;</td>
</tr>
<tr>
<td>Mem</td>
<td>Mem</td>
<td>movq %rax,%rdx</td>
<td>temp2 = temp1;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg</td>
<td>movq (%rax),%rdx</td>
<td>*p = temp;</td>
</tr>
</tbody>
</table>

- ☑ Already seen reg→mem and mem→reg in swap.c
- ☑ Cannot have immediate as destination.
- ☑ Cannot have mem→mem.
- ☑ See addressing_modes.c to see example of immediate.

**Cannot do memory-memory transfer with a single instruction**
addressing_modes.c: Imm→Mem

C code

```c
void immediate ( long * ptr ) {
    *ptr = 0xFFFFFFFFFFFFFFFF;
}
```

Assembly code

```
immediate:
    movq $-1, (%rdi)
    ret
```

- $ indicates the immediate value; corresponds to literals in C
- (%rdi) indicates memory location at address stored in %rdi register
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
addressing_modes.c: Imm→Mem (with displacement)

C code

```c
void displacement_l ( long * ptr ) {
    ptr[1] = 0xFFFFFFFFFFFFFFFF;
}
```

Assembly code

```assembly
displacement_l:
    movq $-1, 8(%rdi)
    ret
```

- 8(%rdi) indicates memory location at address stored in %rdi register + 8
addressing_modes.c: Imm→Mem (with displacement)

<table>
<thead>
<tr>
<th>function signature</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void displacement_c (char * ptr);</td>
<td>movb $-1, 1(%rdi)</td>
</tr>
<tr>
<td>void displacement_s (short * ptr);</td>
<td>movw $-1, 2(%rdi)</td>
</tr>
<tr>
<td>void displacement_i (int * ptr);</td>
<td>movl $-1, 4(%rdi)</td>
</tr>
<tr>
<td>void displacement_l (long * ptr);</td>
<td>movq $-1, 8(%rdi)</td>
</tr>
</tbody>
</table>
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
addressing_modes.c: Imm→Mem (with index)

C code

```c
void index_l ( long * ptr, long index ) {
  ptr[index] = 0xFFFFFFFFFFFFFFFF;
}
```

Assembly code

```
index_l:
  movq $-1, (%rdi,%rsi,8)
  ret
```

▶ (%rdi,%rsi,8) indicates memory location at address stored in %rdi register + 8 × value stored in %rsi register
### addressing_modes.c: Imm\(\rightarrow\)Mem (with index)

<table>
<thead>
<tr>
<th>function signature</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void index_c ( char * ptr, long index );</td>
<td>movb $-1, (%rdi,%rsi)</td>
</tr>
<tr>
<td>void index_s ( short * ptr, long index );</td>
<td>movw $-1, (%rdi,%rsi,2)</td>
</tr>
<tr>
<td>void index_i ( int * ptr, long index );</td>
<td>movl $-1, (%rdi,%rsi,4)</td>
</tr>
<tr>
<td>void index_l ( long * ptr, long index );</td>
<td>movq $-1, (%rdi,%rsi,8)</td>
</tr>
</tbody>
</table>
addressing_modes.c: Imm→Mem (with displacement and index)

C code

```c
void displacement_and_index ( long * ptr, long index ) {
    ptr[index+1] = 0xFFFFFFFFFFFFFFFF;
}
```

Assembly code

```
displacement_and_index:
    movq $-1, 8(%rdi,%rsi,8)
    ret
```

▶ 8(%rdi,%rsi,8) indicates memory location at address stored in %rdi register + 8 × value stored in %rsi register + 8
Table of contents

Announcements

Assembly code of swap.c

Data size and IA32, x86, and x86-64 registers

Assembly code of addressing_modes.c

Displacement memory addressing mode

Index memory addressing mode

Sign extension
Sign extension due to unsigned and signed data types

Converting to a data type with more bits:

```c
unsigned short uc_to_us ( unsigned char input ) {
    return input;
}
```

<table>
<thead>
<tr>
<th>function signature</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short uc_to_us ( unsigned char input );</td>
<td>movzbl %dil, %eax</td>
</tr>
<tr>
<td>signed short uc_to_ss ( unsigned char input );</td>
<td>movzbl %dil, %eax</td>
</tr>
<tr>
<td>unsigned short sc_to_us ( signed char input );</td>
<td>movsbw %dil, %ax</td>
</tr>
<tr>
<td>signed short sc_to_ss ( signed char input );</td>
<td>movsbw %dil, %ax</td>
</tr>
</tbody>
</table>

- movz: zero extension in the MSBs
- movs: signed extension in the MSBs