Assembly: Data movement and arithmetic operations.

Yipeng Huang

Rutgers University

March 2, 2021

Table of contents

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.
6. Programming Assignment 3 on bits, bytes, integers, floats out. Due Monday March 22.

Programming Assignment 3: Quickstart

Table of contents

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 );`

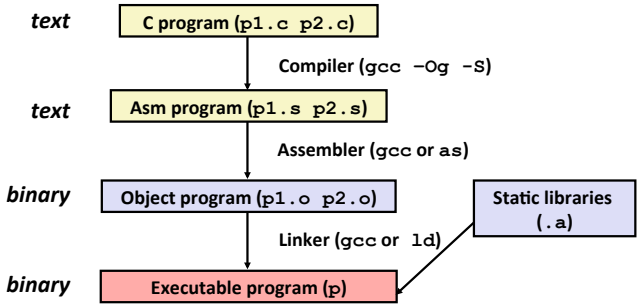
Swapping different data sizes

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 -0g -S swap.c
- objdump -d swap

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

Table of contents

Data size and x86 / x86-64 registers

Assembly syntax

Instruction Source, Dest

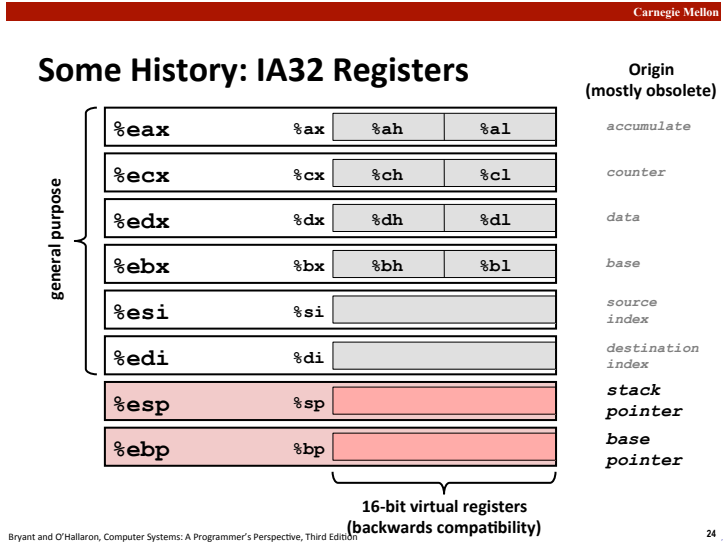
```
swap_l:
 movq (%rsi), %rax
 movq (%rdi), %rdx
 movq %rdx, (%rsi)
 movq %rax, (%rdi)
 ret
```

swap	data type	mov operation	registers
swap_uc	unsigned char	movb (move byte)	%al, %dl
swap_sc	signed char	movb (move byte)	%al, %dl
swap_c	char	movb (move byte)	%al, %dl
swap_s	short	movw (move word)	%ax, %dx
swap_i	int	movl	%eax, %edx
swap_l	long	movq	%rax, %rdx

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

data type	registers
char	%al, %dl
short	%ax, %dx
int	%eax, %edx
long	%rax, %rdx

Note the backward compatibility.



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

data type	registers
char	%al, %dl
short	%ax, %dx
int	%eax, %edx
long	%rax, %rdx

Note the backward compatibility.

x86-64 Integer Registers

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

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

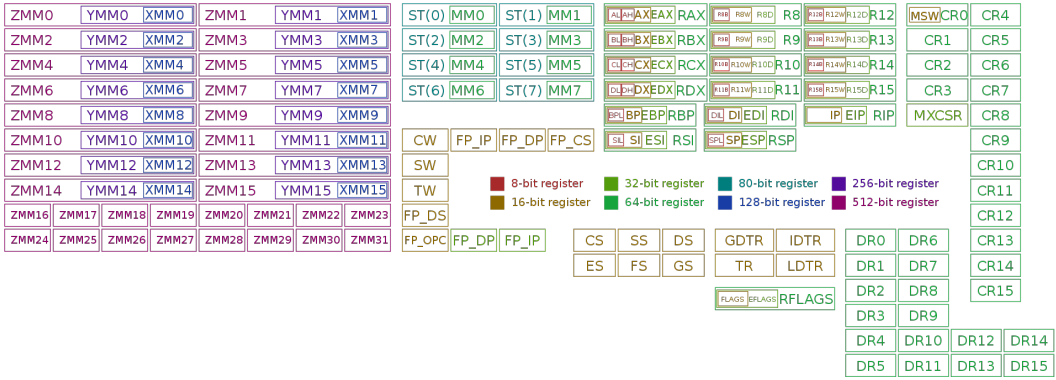


Figure: x86-64 with SIMD extensions registers. Image credit: `https://commons.wikimedia.org/wiki/File:Table_of_x86_Registers_svg.svg`

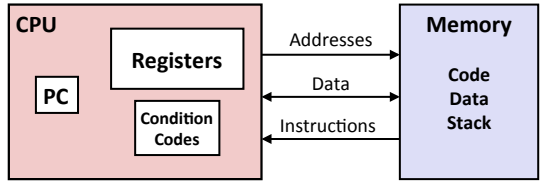
Table of contents

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

	Source	Dest	Src,Dest	C Analog
movq	Imm	Reg	`movq $0x4,%rax`	`temp = 0x4;`
		Mem	`movq $-147,(%rax)`	`*p = -147;`
	Reg	Reg	`movq %rax,%rdx`	`temp2 = temp1;`
		Mem	`movq %rax,(%rdx)`	`*p = temp;`
	Mem	Reg	`movq (%rax),%rdx`	`temp = *p;`

Cannot do memory-memory transfer with a single instruction

- 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.

addressing_modes.c: Imm→Mem

C code

```
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

addressing_modes.c: Imm→Mem (with displacement)

C code

```
void displacement_1 ( long * ptr ) {
  ptr[1] = 0xFFFFFFFFFFFFFFFF;
}
```

Assembly code

```
displacement_1:
    movq $-1, 8(%rdi)
    ret
```

- 8(%rdi) indicates memory location at address stored in %rdi register + 8

addressing_modes.c: Imm→Mem (with displacement)

function signature	assembly code
void displacement_c (char * ptr);	movb $-1, 1(%rdi)
void displacement_s (short * ptr);	movw $-1, 2(%rdi)
void displacement_i (int * ptr);	movl $-1, 4(%rdi)
void displacement_l (long * ptr);	movq $-1, 8(%rdi)

Table of contents

addressing_modes.c: Imm→Mem (with index)

C code

```
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→Mem (with index)

function signature	assembly code
void index_c (char * ptr, long index);	movb $-1, (%rdi,%rsi)
void index_s (short * ptr, long index);	movw $-1, (%rdi,%rsi,2)
void index_i (int * ptr, long index);	movl $-1, (%rdi,%rsi,4)
void index_l (long * ptr, long index);	movq $-1, (%rdi,%rsi,8)

addressing_modes.c: Imm→Mem (with displacement and index)

C code

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

- 8(%rdi,%rsi,8) indicates memory location at address stored in %rdi register + 8 × value stored in %rsi register + 8

Assembly code

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

Table of contents

Sign extension due to unsigned and signed data types

Converting to a data type with more bits:

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

function signature	assembly code
unsigned short uc_to_us (unsigned char input);	movzbl %dil, %eax
signed short uc_to_ss (unsigned char input);	movzbl %dil, %eax
unsigned short sc_to_us (signed char input);	movsbw %dil, %ax
signed short sc_to_ss (signed char input);	movsbw %dil, %ax

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