Arithmetic Example

Register	Use(s)
%rdi	1st argument (x)
%rsi	2nd argument (y)
%rax	return value

```
long simple_arith(long x, long y)
{
  long t1 = x + y;
  long t2 = t1 * 3;
  return t2;
}
```

```
y += x;
y *= 3;
long r = y;
return r;
```

```
simple_arith:
  addq    %rdi, %rsi
  imulq     $3, %rsi
  movq    %rsi, %rax
  ret
```


Understanding swap()

Registers

%rdi	0x120
%rsi	0x100
%rax	
%rdx	

Memory

	Word Address
123	0x120
	0x118
	0x110
	0x108
456	0x100

```
swap:
   movq   (%rdi), %rax  #  t0 = *xp
   movq   (%rsi), %rdx  #  t1 = *yp
   movq   %rdx, (%rdi)  # *xp =  t1
   movq   %rax, (%rsi)  # *yp =  t0
   ret
```


Address Computation Examples

%rdx	0xf000
%rcx	0x0100

D(Rb,Ri,S) → Mem[Reg[Rb]+Reg[Ri]*S+D]

ignore the memory access for now

Expression	Address Computation	Address (8 bytes wide)
0x8(%rdx)		
(%rdx,%rcx)		
(%rdx,%rcx,4)		
0x80(,%rdx,2)		

Review Questions

- Which of the following x86-64 instructions correctly calculates %rax=9*%rdi?
  - A. `leaq (,%rdi,9), %rax`
  - B. `movq (,%rdi,9), %rax`
  - C. `leaq (%rdi,%rdi,8), %rax`
  - D. `movq (%rdi,%rdi,8), %rax`
- If %rsi is 0x B0BACAFE 1EE7 F0 0D, what is its value after executing **`movswl %si, %esi`**?

Arithmetic Example

Register	Use(s)
%rdi	1st argument (x)
%rsi	2nd argument (y)
%rdx	3rd argument (z)

```
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;
}
```

```
arith:
  leaq    (%rdi,%rsi), %rax
  addq    %rdx, %rax
  leaq    (%rsi,%rsi,2), %rdx
  salq    $4, %rdx
  leaq    4(%rdi,%rdx), %rcx
  imulq   %rcx, %rax
  ret
```

- Interesting Instructions
  - `leaq`: "address" computation
  - `salq`: shift
  - `imulq`: multiplication
    - Only used once!

Example Condition Code Setting

- Assuming that `%al` = `0x80` and `%bl` = `0x81`, which flags (CF, ZF, SF, OF) are set when we execute **`cmpb %al, %bl`**?

Practice Question 1

Register	Use(s)
%rdi	1st argument (x)
%rsi	2nd argument (y)
%rax	return value

```
long absdiff(long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
    ________________
    ________________          # x > y:
    movq    %rdi, %rax
    subq    %rsi, %rax
    ret
.L4:                          # x <= y:
    movq    %rsi, %rax
    subq    %rdi, %rax
    ret
```

A. `cmpq %rsi, %rdi` / `jle .L4`
B. `cmpq %rsi, %rdi` / `jg .L4`
C. `testq %rsi, %rdi` / `jle .L4`
D. `testq %rsi, %rdi` / `jg .L4`
E. We're lost…