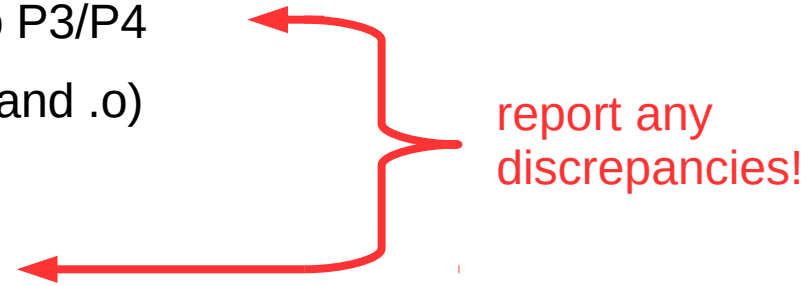
CS 261
Fall 2016

Mike Lam, Professor

Y86-64 Introduction

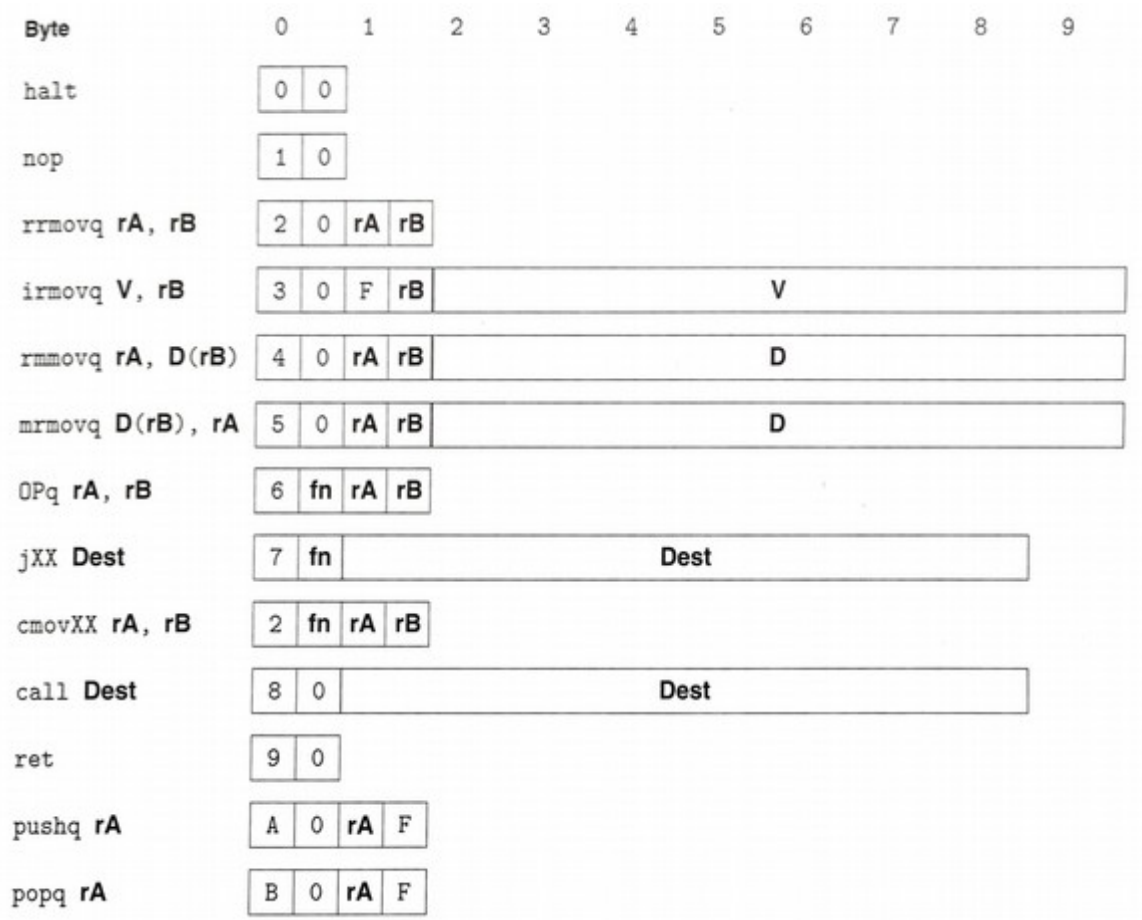
Projects 3 & 4: Support Utilities

- New folder on stu: `/cs/students/cs261/f16/src/`**`y86`**
  - **`isa.pdf`**: Y86-64 reference sheet
  - **`y86`**: compiled reference solution to P3/P4
  - **`yas`**: Y86-64 assembler (.ys → .yo and .o)
  - **`yis`**: Y86-64 simulator (takes .yo)
  - **`ssim`**: CPU simulator (takes .yo)
  - **`simple.ys`**: sample Y86-64 assembly program



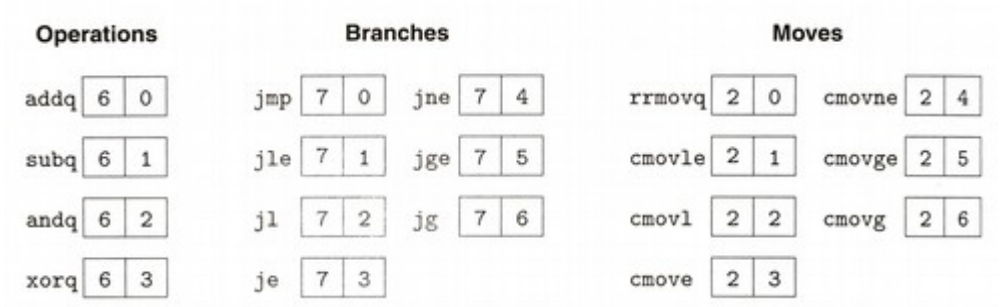
- These will help with P3/P4: learn to use them!
  - "`yas <yourfile.ys>`" to assemble code into object files
- Hint: make shortcuts in your working folder for easier access
  - "`ln -s /cs/students/cs261/f16/src/y86/y86 ref`"
  - "`ln -s /cs/students/cs261/f16/src/y86/yas yas`"
  - "`ln -s /cs/students/cs261/f16/src/y86/ssim ssim`"

Projects 3 & 4: Y86-64 ISA



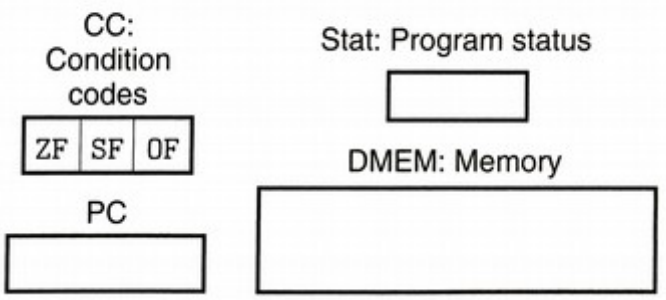
Number	Register name
0	%rax
1	%rcx
2	%rdx
3	%rbx
4	%rsp
5	%rbp
6	%rsi
7	%rdi

Value	Name	Meaning
1	AOK	Normal operation
2	HLT	halt instruction encountered
3	ADR	Invalid address encountered
4	INS	Invalid instruction encountered



RF: Program registers

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	



Differences from textbook

- No extended registers (`%r8-%r14`)
- Execution begins at "entry point" from MiniELF, not address zero
  - Question: is there a significance to "address zero"?
  - Use "`_start`" label to indicate entry point in assembly
  - Use a jump if you want to run the simulator
  - Example:

```
.pos 0 code
   jmp _start

.pos 0x100 code
_start:
   <code goes here>
```

Using the stack

- The stack must be initialized manually
  - Example:

```
.pos 0 code
   jmp _start

.pos 0x100 code
_start:
   irmovq _stack, %rsp
   <code goes here>

.pos 0xf00 stack
_stack:
```

Data segments

- Data should be stored in data segments
  - Retrieve address (i.e., create pointer) using labels and `irmovq`
  - No indexed addressing mode--must do pointer arithmetic yourself!
  - Example:

```
.pos 0x100 code
_start:
    irmovq array, %rbx              # rbx = &data
    mrmovq (%rbx), %rax             # rax = *rbx

    irmovq $2, %rdi
    addq %rbx, %rdi
    mrmovq (%rdi), %rcx             # rcx = rbx[2]

.pos 0x300 data
array:
    .quad 1
    .quad 2
    .quad 3
    .quad 4
```

Exercises

- Write Y86-64 code to add 3 and 5 (store result in `%rbx`)
- Write Y86-64 code to multiply 3 and 5 (store result in `%rcx`)
  - HINT: add 3 to itself 5 times, or vice versa
- Write a function that adds any two numbers
  - Use standard x86 calling conventions
    - (params in `%rdi` and `%rsi`, return in `%rax`)
  - Include driver code that calls the function
  - Don't forget to set up the stack!

Exercises

- Write Y86-64 code to add 3 and 5
- Write Y86-64 code to multiply 3 and 5
- Write a function that adds any two numbers

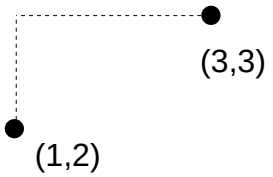
```
.pos 0x100 code
_start:
    irmovq $3, %rax
    irmovq $5, %rbx
    addq %rax, %rbx
    halt
```

```
.pos 0x100 code
_start:
    irmovq $3, %rax
    irmovq $5, %rdx
    irmovq $0, %rcx
    irmovq $1, %rdi
loop:
    addq %rax, %rcx
    subq %rdi, %rdx
    jne loop
    halt
```

```
.pos 0x100 code
_start:
    irmovq _stack, %rsp
    irmovq $4, %rdi
    irmovq $8, %rsi
    call addnums
    halt
addnums:
    # params: %rdi and %rsi
    addq %rsi, %rdi
    rrmovq %rdi, %rax
    ret
.pos 0xf00 stack
_stack:
```

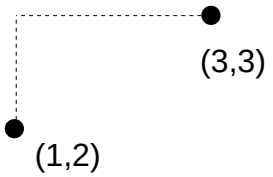
Example

```
                        %rdi      %rsi      %rdx      %rcx
long manhattan(long x1, long y1, long x2, long y2)
{
    long dx = x2 – x1;
    long dy = y2 – y1;
    return dx + dy;
}
```



Example

```
                    %rdi      %rsi      %rdx      %rcx
long manhattan(long x1, long y1, long x2, long y2)
{
    long dx = x2 – x1;
    long dy = y2 – y1;
    return dx + dy;
}
```




```
.pos 0 code
    jmp _start

.pos 0x100 code
_start:
    irmovq 0xf00, %rsp
    irmovq $1, %rdi
    irmovq $2, %rsi
    irmovq $3, %rdx
    irmovq $3, %rcx
    call manhattan
    halt

manhattan:
    subq %rdi, %rdx
    subq %rsi, %rcx
    addq %rdx, %rcx
    rrmovq %rcx, %rax
    ret
```

Example

```
long sum (long *start, long count)
{
    long sum = 0;
    for (int i = 0; i < count; i++) {
        sum += start[i];
    }
    return sum;
}
```

```
long sum (long *start, long count)
{
    long sum = 0;
    while (count) {
        sum += *start;
        start++;
        count--;
    }
    return sum;
}
```

Y86-64 code

```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     irmovq $8,%r8          Constant 8
3     irmovq $1,%r9          Constant 1
4     xorq %rax,%rax         sum = 0
5     andq %rsi,%rsi         Set CC
6     jmp     test           Goto test
7   loop:
8     mrmovq (%rdi),%r10     Get *start
9     addq %r10,%rax         Add to sum
10    addq %r8,%rdi          start++
11    subq %r9,%rsi          count--.  Set CC
12  test:
13    jne    loop            Stop when 0
14    ret                    Return
```

Template

```
.pos 0 code
    jmp _start

.pos 0x100 code
_start:
    irmovq _stack, %rsp
    [ YOUR CODE GOES HERE ]
    halt

.pos 0xf00 stack
_stack:
```