Systems Programming

Machine-Level Programming II: Control

Byoungyoung Lee

Seoul National University

byoungyoung@snu.ac.kr

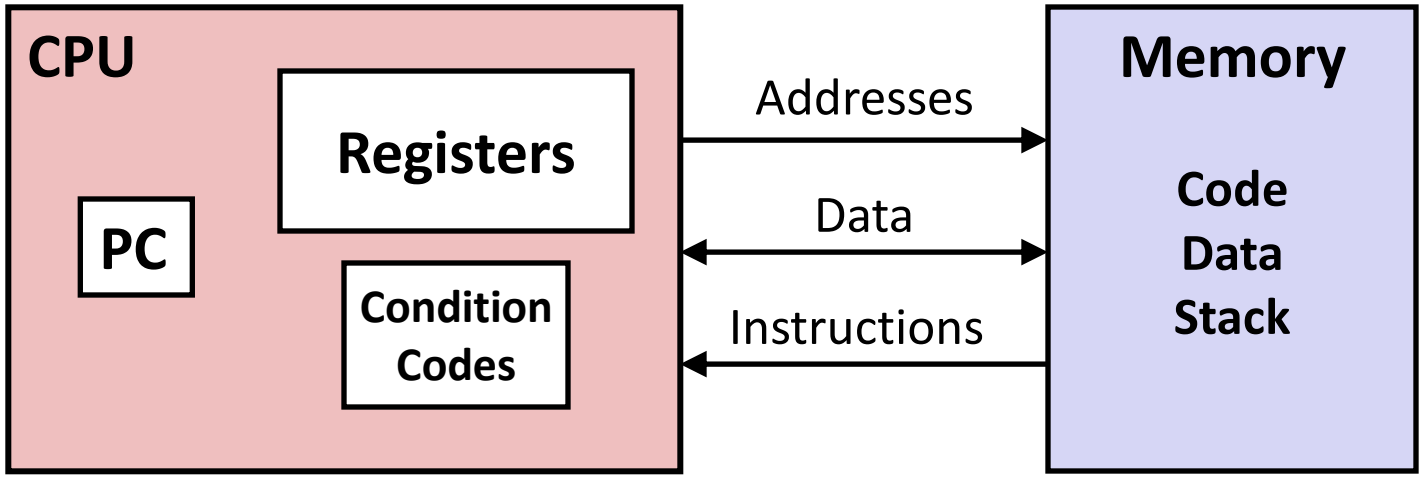
https://lifeasageek.github.io

Lecture slides are prepared based on materials provided by CSAPP authors.

Today

- From last week: Turning C into machine code
- Basics of control flow
- Condition codes
- Conditional branches
- Loops

Recall: ISA = Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

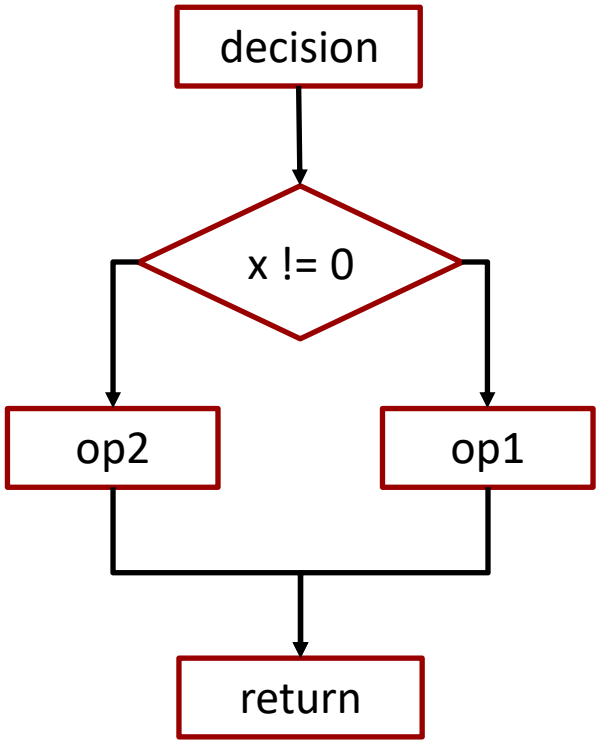
- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures


CONTROL... CONTROL...
YOU MUST LEARN CONTROL

Control flow

```
extern void op1(void);
extern void op2(void);

void decision(int x) {
    if (x) {
        op1();
    } else {
        op2();
    }
}
```



Control flow in assembly language

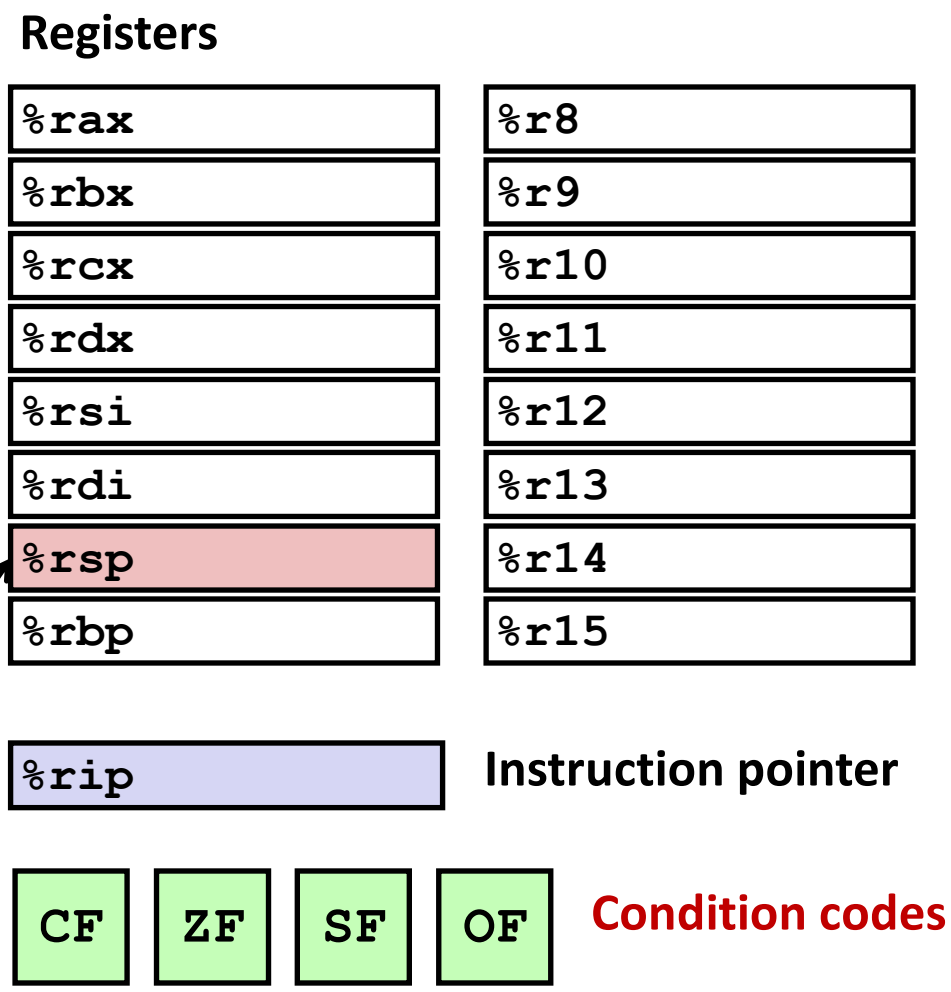
```
extern void op1(void);
extern void op2(void);

void decision(int x) {
    if (x) {
        op1();
    } else {
        op2();
    }
}
```

```
decision:
        subq    $8, %rsp
        testl   %edi, %edi
        je      .L2
        call    op1
        jmp     .L1
.L2:
        call    op2
.L1:
        addq    $8, %rsp
        ret
```

Processor State (x86-64)

- **Information about currently executing program**
  - Temporary data (**%rax**, …)
  - Location of runtime stack (**%rsp**)
  - Location of current code control point (**%rip**, …)
  - Status of recent tests (**CF, ZF, SF, OF**)



Condition Codes (Implicit Setting)

- **Single bit registers**
  - **CF** Carry Flag (for unsigned) **SF** Sign Flag (for signed)
  - **ZF** Zero Flag **OF** Overflow Flag (for signed)

- **Implicitly set (as side effect) after arithmetic operations**

  Example: **`addq`** *Src*,*Dest* ↔ **`t = a+b`**

  CF set if carry/borrow out from most significant bit (unsigned overflow)

  ZF set if **`t == 0`**

  SF set if **`t < 0`** (as signed)

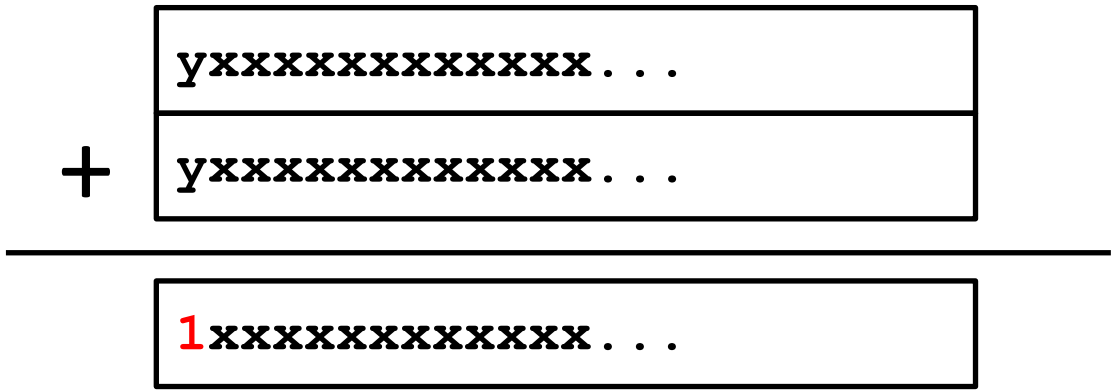
  OF set if two's-complement (signed) overflow

```
(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)
```

ZF set when

000000000000…00000000000

SF set when

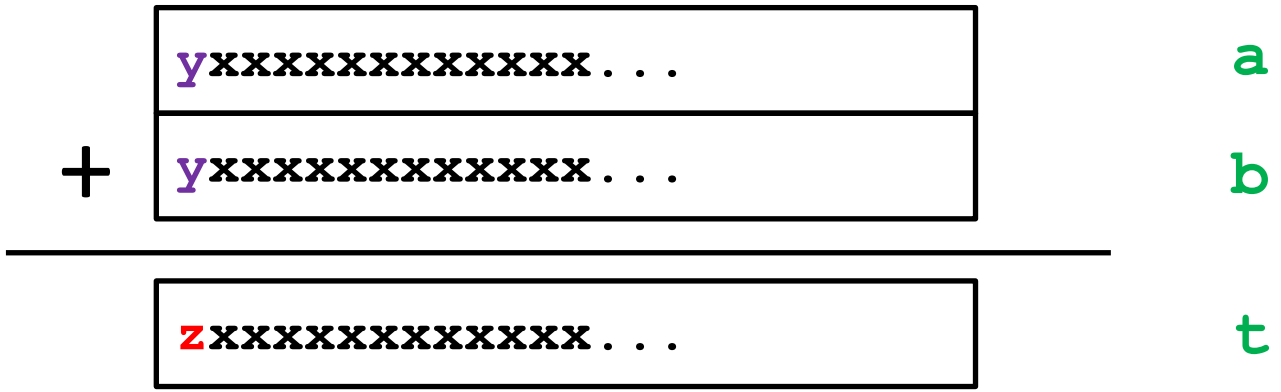


For signed arithmetic, this reports when result is a negative number

CF set when

For unsigned arithmetic, this reports overflow

OF set when



$z = \sim y$

```
(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)
```

For signed arithmetic, this reports overflow

Condition Codes (Explicit Setting: Compare)

- **Explicit Setting by Compare Instruction**
  - **`cmpq`** *Src2*, *Src1*
  - **`cmpq b,a`** like computing **`a-b`** without setting destination

  - **CF set** if carry/borrow out from most significant bit (used for unsigned comparisons)
  - **ZF set** if **`a == b`**
  - **SF set** if **`(a-b) < 0`** (as signed)
  - **OF set** if two's-complement (signed) overflow

```
(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)
```

Condition Codes (Explicit Setting: Test)

- **Explicit Setting by Test instruction**
  - **`testq`** *Src2*, *Src1*
    - **`testq b,a`** like computing **`a&b`** without setting destination
  - Sets condition codes based on value of *Src1* & *Src2*
  - Useful to have one of the operands be a mask
  - **ZF set** when **`a&b == 0`**
  - **SF set** when **`a&b < 0`**

Very often:

```
testq  %rax,%rax
```

Today

- Control: Condition codes
- **Conditional branches**
- Loops

Jumping

- **jX Instructions**
  - Jump to different part of code depending on condition codes
  - Implicit reading of condition codes

jX	Condition	Description
`jmp`	`1`	Unconditional
`je`	`ZF`	Equal / Zero
`jne`	`~ZF`	Not Equal / Not Zero
`js`	`SF`	Negative
`jns`	`~SF`	Nonnegative
`jg`	`~(SF^OF)&~ZF`	Greater (signed)
`jge`	`~(SF^OF)`	Greater or Equal (signed)
`jl`	`SF^OF`	Less (signed)
`jle`	`(SF^OF)|ZF`	Less or Equal (signed)
`ja`	`~CF&~ZF`	Above (unsigned)
`jb`	`CF`	Below (unsigned)

Conditional Branch Example

- **Generation**

```
$ gcc -Og -S -fno-if-conversion control.c
```

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

```
absdiff:
   cmpq     %rsi, %rdi   # x-y
   jle      .L4
   movq     %rdi, %rax
   subq     %rsi, %rax   # x-y
   ret
.L4:                     # x <= y
   movq     %rsi, %rax
   subq     %rdi, %rax   # y-x
   ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

Today

- Control: Condition codes
- Conditional branches
- **Loops**

“Do-While” Loop Example

C Code

```
long pcount_do(ulong x) {
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

Goto Version

```
long pcount_goto(ulong x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

- Count number of 1’s in argument `x`
- Use conditional branch to either continue looping or to exit loop

General “Do-While” Translation

C Code

```
do
   Body
   while (Test);
```

Goto Version

```
loop:
   Body
   if (Test)
     goto loop
```

“Do-While” Loop Compilation

```
long pcount_goto(ulong x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

Register	Use(s)
%rdi	Argument x
%rax	result

```
   xorq    %rax,%rax   #   result = 0
.L2:                   # loop:
   movq    %rdi, %rdx
   andl    $1, %edx    #   t = x & 0x1
   addq    %rdx, %rax  #   result += t
   shrq    %rdi        #   x >>= 1
   jne     .L2         #   if(x) goto loop
   ret
```

General “While” Translation

- “Jump-to-middle” translation
- Used with –Og

While version

```
while (Test)
  Body
```

Goto Version

```
  goto test;
loop:
  Body
test:
  if (Test)
    goto loop;
done:
```

While Loop Example

C Code

```
long pcount_while(ulong x)
{
  long result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Jump to Middle

```
long pcount_goto_jtm(ulong x)
{
  long result = 0;
  goto test;
 loop:
  result += x & 0x1;
  x >>= 1;
 test:
  if(x) goto loop;
  return result;
}
```

- Compare to do-while version of function
- Initial goto starts loop at test

“For” Loop Form

General Form

```
for (Init; Test; Update)
    Body
```

```
#define WSIZE 8*sizeof(int)
long pcount_for(ulong x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{
  unsigned bit =
      (x >> i) & 0x1;
  result += bit;
}
```

“For” Loop → While Loop

For Version

```
for (Init; Test; Update)
        Body
```

While Version

```
Init;
while (Test) {
    Body
    Update;
}
```

Summarizing

- **C Control**
  - if-then-else
  - do-while
  - while, for
  - Switch (didn’t cover here)

- **Assembler Control**
  - Conditional jump
  - Conditional move
  - Indirect jump (via jump tables)
  - Compiler generates code sequence to implement more complex control

Summary

- **Today**
  - Control: Condition codes
  - Conditional branches & conditional moves
  - Loops
  - Switch statements
- **Next Time**
  - Stack
  - Call / return
  - Procedure call discipline