Conditionals and Control Flow

- **A conditional branch is sufficient to implement most control flow constructs offered in higher level languages**
  - if (condition) then {…} else {…}
  - while (condition) {…}
  - do {…} while (condition)
  - for (initialization; condition; iterative) {…}
- **Unconditional branches implement some related control flow constructs**
  - break, continue
- **In x86, we'll refer to branches as "jumps" (either conditional or unconditional)**

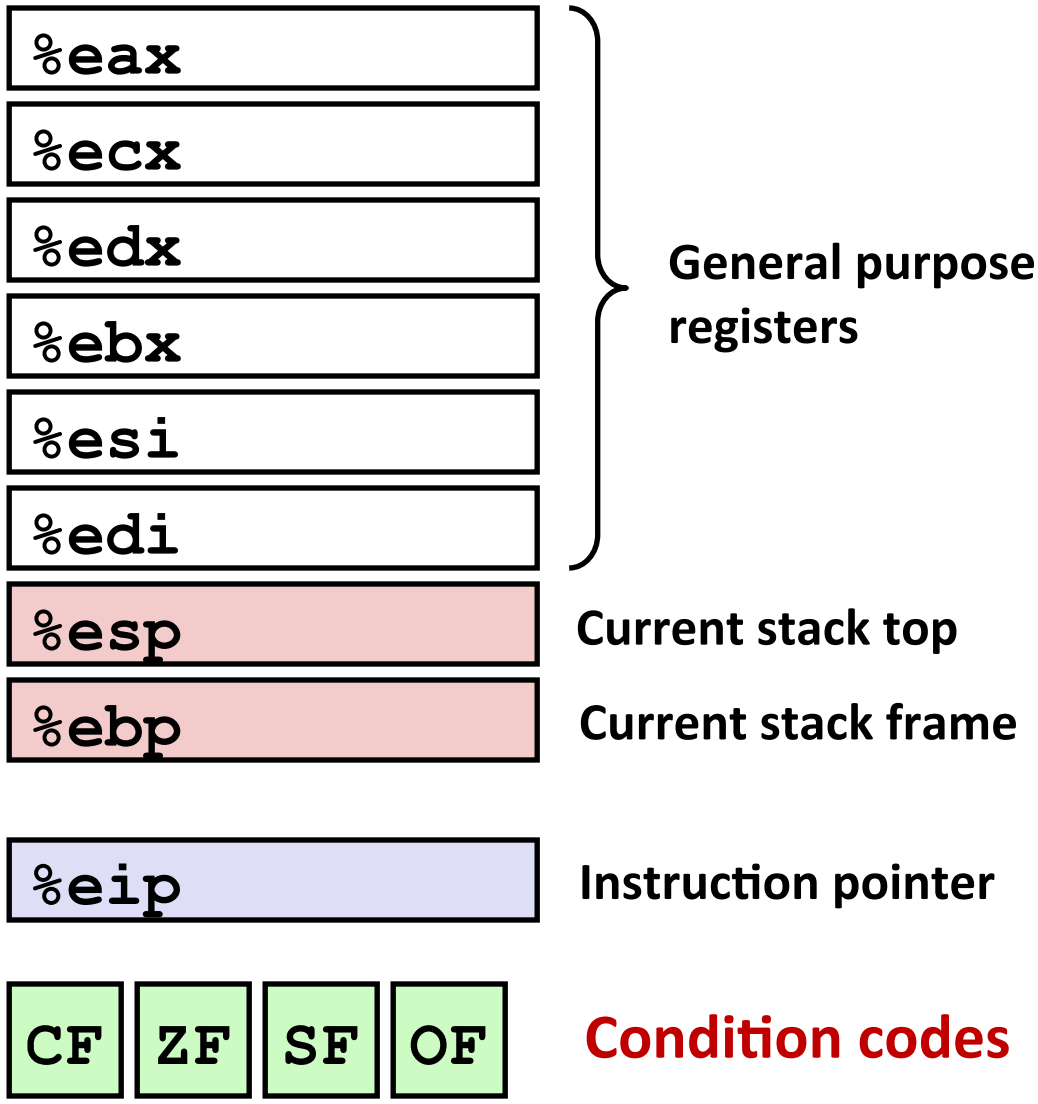
Jumping

- **jX Instructions**
  - Jump to different part of code depending on 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)

Processor State (IA32, Partial)

- **Information about currently executing program**
  - Temporary data (**%eax**, …)
  - Location of runtime stack (**%ebp**, **%esp**)
  - Location of current code control point (**%eip**)
  - 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 (think of it as side effect) by arithmetic operations**

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

  - **CF set** if carry 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)`

- ***Not*** **set by `lea` instruction (beware!)**
- **Full documentation (IA32): http://www.jegerlehner.ch/intel/IntelCodeTable.pdf**

Condition Codes (Explicit Setting: Compare)

- **Single-bit registers**

  `CF` Carry Flag (for unsigned) **`SF`** Sign Flag (for signed)

  `ZF` Zero Flag **`OF`** Overflow Flag (for signed)

- **Explicit Setting by Compare Instruction**

  `cmpl/cmpq` *Src2,Src1*

  `cmpl b,a` like computing **`a-b`** without setting destination

  - **CF set** if carry 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)

- **Single-bit registers**

  CF Carry Flag (for unsigned) **SF** Sign Flag (for signed)

  ZF Zero Flag **OF** Overflow Flag (for signed)

- **Explicit Setting by Test instruction**

  testl/**testq** *Src2,Src1*

  testl 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** if **a&b == 0**
  - **SF set** if **a&b < 0**

  - **testl %eax, %eax**
    - Sets SF and ZF, check if eax is +,0,-

Reading Condition Codes

- **SetX Instructions**
  - Set a single byte to 0 or 1 based on combinations of condition codes

SetX	Condition	Description
`sete`	`ZF`	Equal / Zero
`setne`	`~ZF`	Not Equal / Not Zero
`sets`	`SF`	Negative
`setns`	`~SF`	Nonnegative
`setg`	`~(SF^OF)&~ZF`	Greater (Signed)
`setge`	`~(SF^OF)`	Greater or Equal (Signed)
`setl`	`(SF^OF)`	Less (Signed)
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)

Reading Condition Codes (Cont.)

- **SetX Instructions:**
  Set single byte to 0 or 1 based on combination of condition codes
- **One of 8 addressable byte registers**
  - Does not alter remaining 3 bytes
  - Typically use **`movzbl`** to finish job

```
int gt (int x, int y)
{
  return x > y;
}
```

%eax	%ah	%al
%ecx	%ch	%cl
%edx	%dh	%dl
%ebx	%bh	%bl
%esi		
%edi		
%esp		
%ebp		

Body: y at 12(%ebp), x at 8(%ebp)

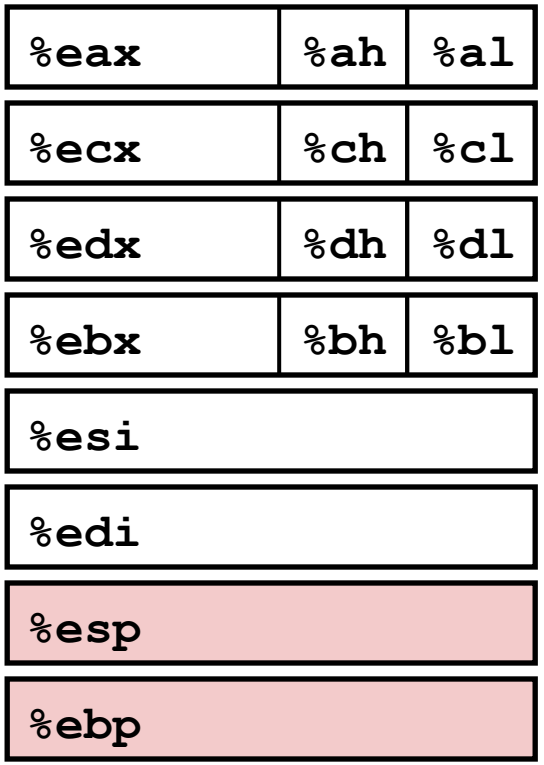
```
movl 12(%ebp),%eax
cmpl %eax,8(%ebp)
setg %al
movzbl %al,%eax
```

What does each of these instructions do?

Reading Condition Codes (Cont.)

- **SetX Instructions:**
  Set single byte to 0 or 1 based on combination of condition codes
- **One of 8 addressable byte registers**
  - Does not alter remaining 3 bytes
  - Typically use **`movzbl`** to finish job

```
int gt (int x, int y)
{
  return x > y;
}
```



Body: y at 12(%ebp), x at 8(%ebp)

```
movl 12(%ebp),%eax      # eax = y
cmpl %eax,8(%ebp)       # Compare x and y   <-- (x – y)
setg %al                # al = x > y
movzbl %al,%eax         # Zero rest of %eax
```