Lab4

- Strategy?

- lea: how does adding work again?
- What’s going on w/ data and control transfer?
- Look at a jump table example

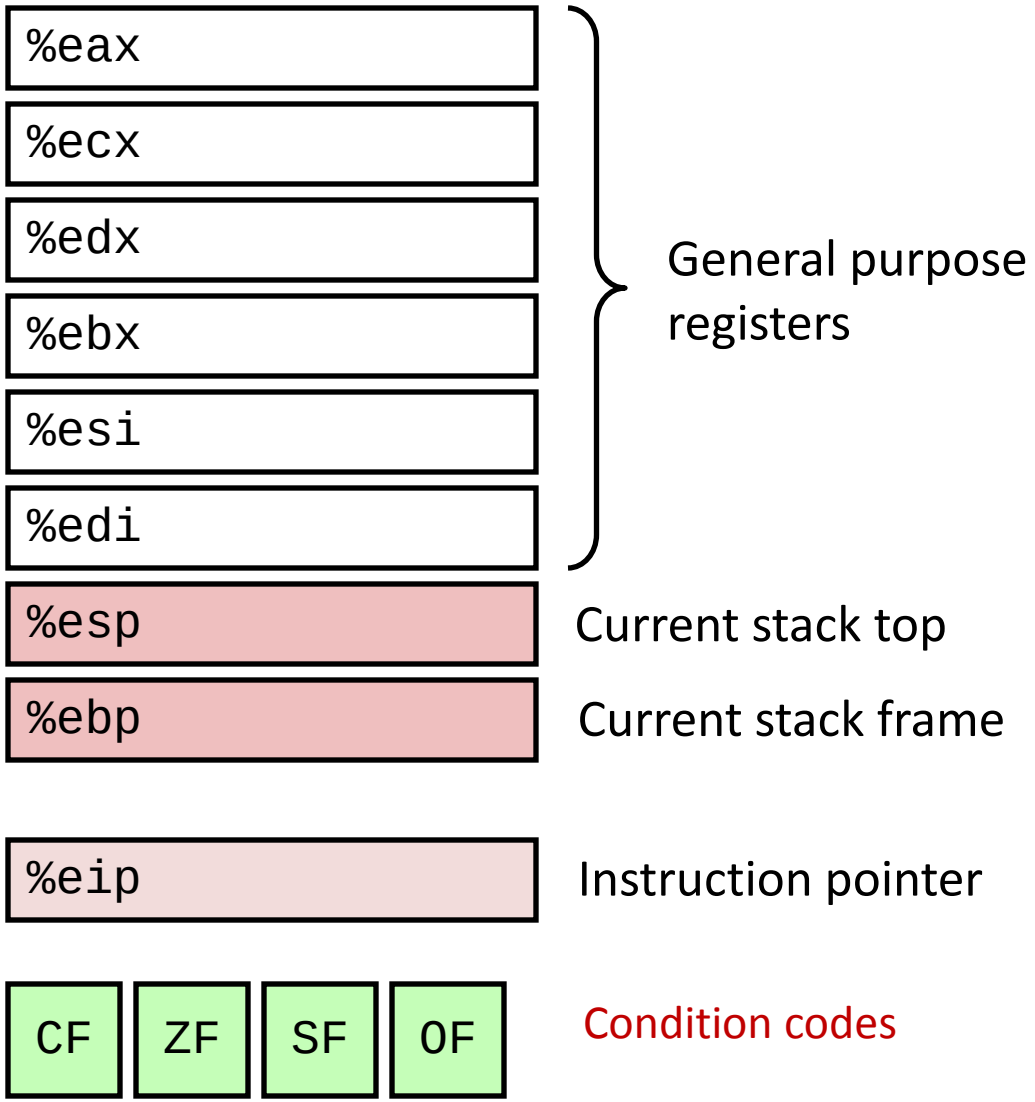
Simple leal example

```
int t1 = x+y;
t1: %ecx
x: %edx
y: %eax
```

```
leal (%edx,%eax),%ecx
```

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

Condition Codes (Explicit Setting: Compare)

- 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)

- 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 when **`a&b == 0`**

  – SF set when **`a&b < 0`**

Reading Condition Codes

- SetX Instructions
  - Set single byte 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 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

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

Reading Condition Codes (Cont.)

- SetX Instructions:
  Set single byte 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

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

Note inverted ordering!

Conditional Branch Example

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

```
absdiff:
    pushl   %ebp                    } Setup
    movl    %esp, %ebp
    movl    8(%ebp), %edx           }
    movl    12(%ebp), %eax          }
    cmpl    %eax, %edx              } Body1
    jle     .L7                     }
    subl    %eax, %edx              }
    movl    %edx, %eax              }
.L8:
    leave                           } Finish
    ret                             }
.L7:
    subl    %edx, %eax              } Body2
    jmp     .L8                     }
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
Exit:
  return result;
Else:
  result = y-x;
  goto Exit;
}
```

```
absdiff:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L7
    subl    %eax, %edx
    movl    %edx, %eax
.L8:
    leave
    ret
.L7:
    subl    %edx, %eax
    jmp     .L8
```

- C allows “goto” as means of transferring control
  - Closer to machine-level programming style
- Generally considered bad coding style

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
Exit:
  return result;
Else:
  result = y-x;
  goto Exit;
}
```

```
absdiff:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L7
    subl    %eax, %edx
    movl    %edx, %eax
.L8:
    leave
    ret
.L7:
    subl    %edx, %eax
    jmp     .L8
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
Exit:
  return result;
Else:
  result = y-x;
  goto Exit;
}
```

```
absdiff:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L7
    subl    %eax, %edx
    movl    %edx, %eax
.L8:
    leave
    ret
.L7:
    subl    %edx, %eax
    jmp     .L8
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
Exit:
  return result;
Else:
  result = y-x;
  goto Exit;
}
```

```
absdiff:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L7
    subl    %eax, %edx
    movl    %edx, %eax
.L8:
    leave
    ret
.L7:
    subl    %edx, %eax
    jmp     .L8
```

General Conditional Expression Translation

C Code

```
val = Test ? Then-Expr : Else-Expr;
```

```
val = x>y ? x-y : y-x;
```

Goto Version

```
  nt = !Test;
  if (nt) goto Else;
  val = Then-Expr;
Done:
  . . .
Else:
  val = Else-Expr;
  goto Done;
```

- *Test* is expression returning integer
  - = 0 interpreted as false
  - ≠0 interpreted as true
- Create separate code regions for then & else expressions
- Execute appropriate one

General Form with Conditional Move

C Code

```
val = Test ? Then-Expr : Else-Expr;
```

Conditional Move Version

```
val1  = Then-Expr;
val2  = Else-Expr;
val1  = val2 if !Test;
```

- Both values get computed
- Overwrite then-value with else-value if condition doesn't hold
- Don't use when:
  - Then or else expression have side effects
  - Then and else expression are to expensive