Chapter 3:

Assembly Language Programming II

Topics

- Condition Codes/Flags
  - CF, ZF, OF, SF
- Conditional Control Flow
  - Conditional Jumps & Branching
  - Conditional Move

Announcements

- Data Lab grading this week
  - Sign up for grading time slots with your TA
- Bomb Lab available on moodle, due Friday Oct 3
  - TA reviewed this in recitation on Monday
- Prof. Han traveling today
  - No office hours today, will hold Wed office hours 3-4 pm
- First midterm probably the week of Oct 6
- Essential that you read the textbook in detail & do the practice problems
  - Read Chapter 3.1-3.14, skip 3.12 for now

Recap…

- **X86 Assembly language**
  - Eight 32-bit CPU registers: %eax, %ebx, %ecx, %edx, %edi, %esi, %esp, %ebp
  - `movl` *Source*, *Dest*
    - Move between memory and CPU registers
    - Covered a swap example
    - Complex indexed addressing mode:
      - `movl 8(%eax,%edx,4), dst`
  - `addl` *Src*,*Dest*
    - Also subl, imull, sall, sarl, andl, etc.
  - `leal` *Src*,*Dest*
    - Helpful for array arithmetic
  - Covered an arithmetic assembly example

Another Example

```
int logical(int x, int y)
{
  int t1 = x^y;
  int t2 = t1 >> 17;
  int mask = (1<<13) - 7;
  int rval = t2 & mask;
  return rval;
}
```

```
logical:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    xorl 12(%ebp),%eax
    sarl $17,%eax
    andl $8185,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```

Set Up

Body

Finish

```
movl 8(%ebp),%eax     # eax = x
xorl 12(%ebp),%eax    # eax = x^y
sarl $17,%eax         # eax = t1>>17
andl $8185,%eax       # eax = t2 & 8185
```

Another Example

```
int logical(int x, int y)
{
  int t1 = x^y;
  int t2 = t1 >> 17;
  int mask = (1<<13) - 7;
  int rval = t2 & mask;
  return rval;
}
```

```
logical:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    xorl 12(%ebp),%eax
    sarl $17,%eax
    andl $8185,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```

Set Up

Body

Finish

```
movl 8(%ebp),%eax      # eax = x
xorl 12(%ebp),%eax     # eax = x^y   (t1)
sarl $17,%eax          # eax = t1>>17    (t2)
andl $8185,%eax        # eax = t2 & 8185
```

Another Example

```
int logical(int x, int y)
{
  int t1 = x^y;
  int t2 = t1 >> 17;
  int mask = (1<<13) - 7;
  int rval = t2 & mask;
  return rval;
}
```

```
logical:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    xorl 12(%ebp),%eax
    sarl $17,%eax
    andl $8185,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```

Set Up

Body

Finish

```
movl 8(%ebp),%eax     # eax = x
xorl 12(%ebp),%eax    # eax = x^y   (t1)
sarl $17,%eax         # eax = t1>>17    (t2)
andl $8185,%eax       # eax = t2 & 8185
```

Another Example

```
int logical(int x, int y)
{
  int t1 = x^y;
  int t2 = t1 >> 17;
  int mask = (1<<13) - 7;
  int rval = t2 & mask;
  return rval;
}
```

$2^{13} = 8192, 2^{13} - 7 = 8185$

```
logical:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    xorl 12(%ebp),%eax
    sarl $17,%eax
    andl $8185,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```

Set Up

Body

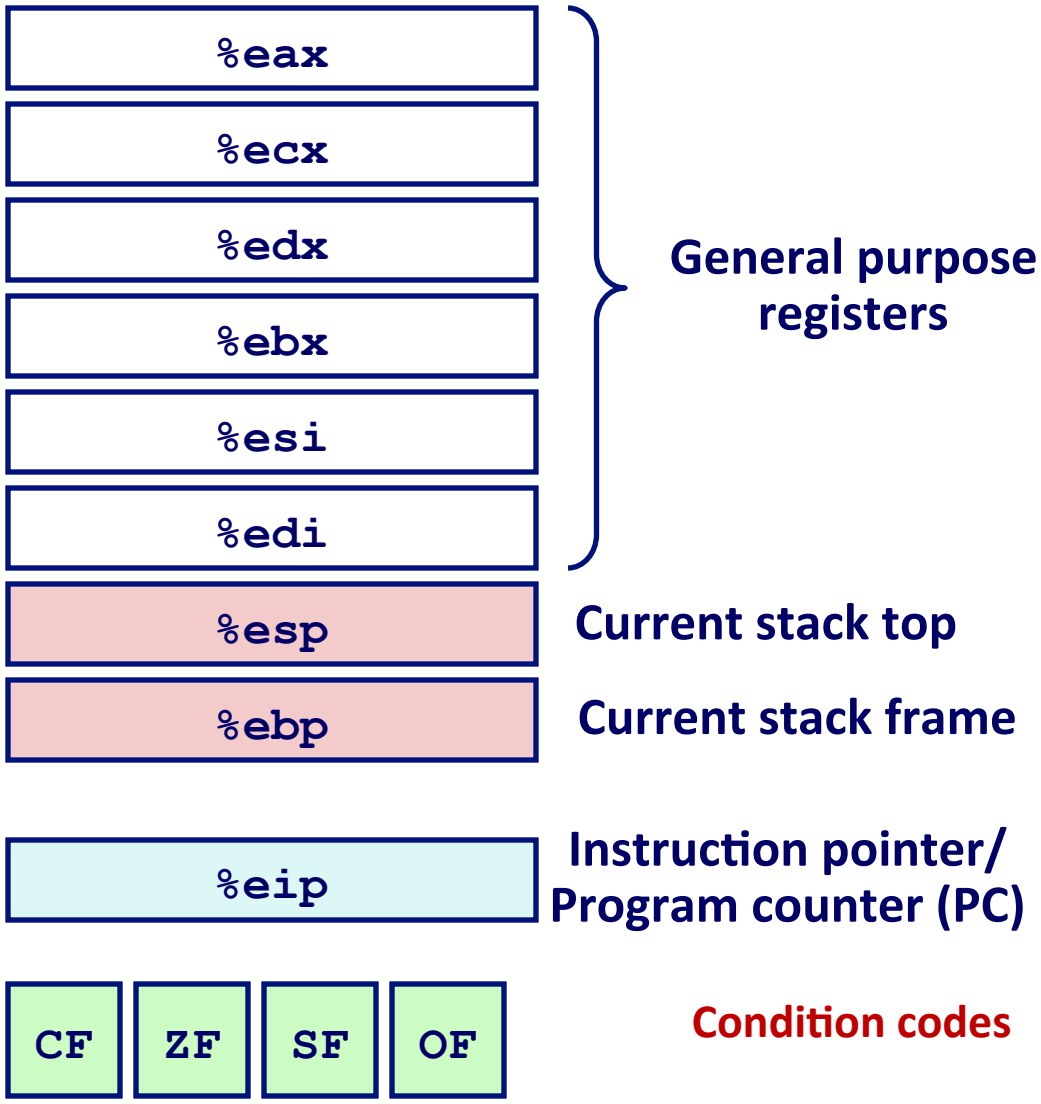
Finish

```
movl 8(%ebp),%eax      # eax = x
xorl 12(%ebp),%eax     # eax = x^y   (t1)
sarl $17,%eax          # eax = t1>>17    (t2)
andl $8185,%eax        # eax = t2 & 8185
```

Note how compiler combines 2 source code lines into 1

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**

  `addl` ***Src,Dest***

  C analog: `t = a+b`

  - **CF set if carry out from most significant bit**
    - **Used to detect 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)`

- **Codes set differently depending on instructions, and in some cases not set at all, e.g.** `lea` **instruction**

Example: Left Shifting <<

- `shl` **or** `sal`
- **How are condition flags set?**
  - **shl/sal sets carry flag to last shifted out bit.**
    - **For unsigned, if CF=1, then overflow occurred.**
  - **for overflow flag OF,**
    - **if shift by one,**
      - » **if CF & MSbit identical after shift, OF = 0**
      - » **else OF = 1 (MS bit changes from 1->0 or 0->1),**

      i.e. OF = CF ^ MSbit
    - **else if shift by > 1, OF undefined.**
    - **For signed, if OF=1, then overflow occurred.**

Condition Codes (Explicit Setting: Compare)

- **Explicit Setting by Compare Instruction**

  `cmpl` ***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

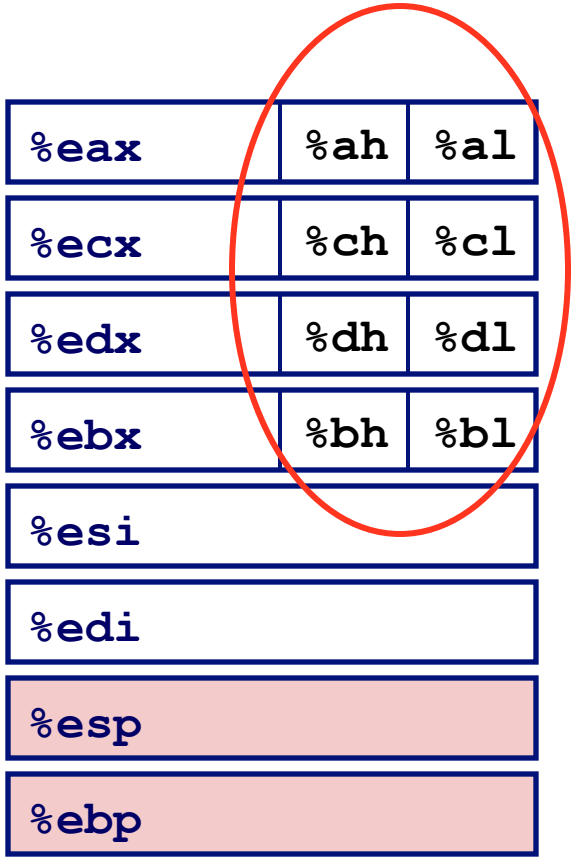
  - Sets condition codes based on value of *Src1* `&` *Src2*
  - Useful to have one of the operands be a mask

  - ZF set when `a&b == 0`
  - SF set when `a&b < 0`

Reading Condition Codes

- **SetX Instructions:**
  Set single byte based on 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

```
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!

Reading Condition Codes

- **setX dest // e.g.** `setl %al`
  - **Set single byte of dest to 0 or 1 based on combinations of condition codes (expressions below are only relevant if previous instruction was a compare** `cmp`**)**

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) – derivation in text
`setle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`seta`	`~CF&~ZF`	Above (unsigned)
`setb`	`CF`	Below (unsigned)

Jumping – Conditional or not

- **jX Instructions**
  - **Unconditional jump: `jmp` Label or `jmp` *%eax or `jmp` *(%eax)**
  - **Conditional jumps to different part of code depending on condition codes, e.g. `jle` Label**

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

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

Conditional Branch Example (Cont.)

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

```
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;
}
```

- 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;
}
```

- Rewritten absdiff with labels closely reflects assembly language

```
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
```

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

Conditional Move

- **cmovX src, dest**
  - Set dest=src only if condition X holds
  - More efficient than conditional branching for highly pipelined processors – easier to guess the next instruction to execute
  - But overhead: both branches are evaluated

cmovX	Condition	Description
`cmove`	`ZF`	Equal / Zero
`cmovne`	`~ZF`	Not Equal / Not Zero
`cmovs`	`SF`	Negative
`cmovns`	`~SF`	Nonnegative
`cmovg`	`~(SF^OF)&~ZF`	Greater (Signed)
`cmovge`	`~(SF^OF)`	Greater or Equal (Signed)
`cmovl`	`(SF^OF)`	Less (Signed)
`cmovle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`cmova`	`~CF&~ZF`	Above (unsigned)
`cmovb`	`CF`	Below (unsigned)

Conditional Move Example

- Rewrite the absdiff example using conditional moves instead of conditional branching

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

```
absdiff: # x in %edi, y in %esi
  movl   %edi, %eax  # eax = x
  movl   %esi, %edx  # edx = y
  subl   %esi, %eax  # eax = x-y
  subl   %edi, %edx  # edx = y-x
  cmpl   %esi, %edi  # x<y?
  cmovl  %edx, %eax  # eax=edx if <=
  ret
```

- Note how the control flow is much easier to predict than all the jumping around with labels, e.g. `jle`, in the conditionally branched version of absdiff

Conditional Move Example

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

```
int cmovdiff(int x, int
y)
{
  int rval = x-y;
  int tval = y-x;
  int test = x < y;
  if (test) rval = tval;
  return rval;
}
```

Conditional Move Example

```
int cmovdiff(int x, int
y)
{
  int rval = x-y;
  int tval = y-x;
  int test = x < y;
  if (test) rval = tval;
  return rval;
}
```

```
absdiff: # x in %edi, y in %esi
  movl   %edi, %eax  # eax = x
  movl   %esi, %edx  # edx = y
  subl   %esi, %eax  # eax = x-y
  subl   %edi, %edx  # edx = y-x
  cmpl   %esi, %edi  # x<y?
  cmovl  %edx, %eax  # eax=edx if <=
  ret
```

Conditional Move Example

```
int cmovdiff(int x, int
y)
{
  int rval = x-y;
  int tval = y-x;
  int test = x < y;
  if (test) rval = tval;
  return rval;
}
```

```
absdiff: # x in %edi, y in %esi
  movl   %edi, %eax  # eax = x
  movl   %esi, %edx  # edx = y
  subl   %esi, %eax  # eax = x-y
  subl   %edi, %edx  # edx = y-x
  cmpl   %esi, %edi  # x<y?
  cmovl  %edx, %eax  # eax=edx if <=
  ret
```

Conditional Move Example

```
int cmovdiff(int x, int
y)
{
  int rval = x-y;
  int tval = y-x;
  int test = x < y;
  if (test) rval = tval;
  return rval;
}
```

```
absdiff: # x in %edi, y in %esi
  movl   %edi, %eax  # eax = x
  movl   %esi, %edx  # edx = y
  subl   %esi, %eax  # eax = x-y
  subl   %edi, %edx  # edx = y-x
  cmpl   %esi, %edi  # x<y?
  cmovl  %edx, %eax  # eax=edx if <=
  ret
```

Conditional Move Example

```
int cmovdiff(int x, int
y)
{
  int rval = x-y;
  int tval = y-x;
  int test = x < y;
  if (test) rval = tval;
  return rval;
}
```

```
absdiff: # x in %edi, y in %esi
  movl   %edi, %eax  # eax = x
  movl   %esi, %edx  # edx = y
  subl   %esi, %eax  # eax = x-y
  subl   %edi, %edx  # edx = y-x
  cmpl   %esi, %edi  # x<y?
  cmovl  %edx, %eax  # eax=edx if <=
  ret
```

- Control flow is more predictable, but both branches must be evaluated

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, like dereferencing a null pointer or incrementing a global variable (then & else expressions always evaluated)
  - Then and else expressions are too expensive