CS 31: Intro to Systems
IA32

Martin Gagné
Swarthmore College
February 9, 2017
Types of IA32 Instructions

• Data movement
  • Move values between registers and memory

• Arithmetic
  • Uses ALU to compute a value

• Control
  • Change PC based on ALU condition code state

• Stack / Function call (We’ll cover these in detail later)
  • Shortcut instructions for common operations
Recall Memory Operands

- displacement(\%reg)
  - e.g., addl \%eax, -8(\%ebp)

- IA32 allows a memory operand as the source or destination, but NOT BOTH
  - One of the operands must be a register

- This would **not** be allowed:
  - addl -4(\%ebp), -8(\%ebp)
  - If you wanted this, movl one value into a register first
From last time...

```assembly
movl %ebp, %ecx
subl $16, %ecx
movl (%ecx), %eax
orl %eax, -8(%ebp)
negl %eax
movl %eax, 4(%ecx)
```
How would you do this in IA32?

x is 2 at %ebp-8, y is 3 at %ebp-12, z is 2 at %ebp-16

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x1270</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0x1260</th>
<th>2</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1264</td>
<td>3</td>
<td>y</td>
</tr>
<tr>
<td>0x1268</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>0x126c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1270</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C code: \( z = x ^ y \)
How would you do this in IA32?

x is 2 at %ebp-8, y is 3 at %ebp-12, z is 2 at %ebp-16

C code: \( z = x \ ^ \ y \)

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x1270</td>
</tr>
</tbody>
</table>

A: 
- movl -8(%ebp), %eax
- movl -12(%ebp), %edx
- xorl %eax, %edx
- movl %eax, -16(%ebp)

B: 
- movl -8(%ebp), %eax
- movl -12(%ebp), %edx
- xorl %edx, %eax
- movl %eax, -16(%ebp)

C: 
- movl -8(%ebp), %eax
- movl -12(%ebp), %edx
- xorl %eax, %edx
- movl %eax, -8(%ebp)

D: 
- movl -16(%ebp), %eax
- movl -12(%ebp), %edx
- xorl %edx, %eax
- movl %eax, -8(%ebp)

E: none of these implements \( z = x \ ^ \ y \)
How would you do this in IA32?

x is 2 at %ebp-8, y is 3 at %ebp-12, z is 2 at %ebp-16

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x1270</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1260</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x1264</td>
<td>3</td>
<td>y</td>
</tr>
<tr>
<td>0x1268</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>0x126c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1270</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x = y >> 3 | x * 8
How would you do this in IA32?

*x is 2 at %ebp-8, y is 3 at %ebp-12, z is 2 at %ebp-16*

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td></td>
</tr>
<tr>
<td>%edx</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x1270</td>
</tr>
</tbody>
</table>

\[
x = y \gg 3 \mid x \times 8
\]

\[
\begin{align*}
\text{movl} & \ -8(\%ebp), \ %eax & \# \ R[\%eax] & \leftarrow x \\
\text{imull} & \ $8, \ %eax & \# \ R[\%eax] & \leftarrow x \times 8 \\
\text{movl} & \ -12(\%ebp), \ %edx & \# \ R[\%edx] & \leftarrow y \\
\text{rshl} & \ $3, \ %edx & \# \ R[\%edx] & \leftarrow y \gg 3 \\
\text{orl} & \ %eax, \ %edx & \# \ R[\%edx] & \leftarrow y \gg 3 \mid x \times 8 \\
\text{movl} & \ %edx, \ -8(\%ebp) & \# \ M[R[\%ebp-8]] & \leftarrow \text{result}
\end{align*}
\]
Control Flow

• Previous examples focused on:
  – data movement (movl)
  – arithmetic (addl, subl, orl, negl, sall, etc.)

• Up next: Jumping!

(Changing which instruction we execute next.)
Unconditional Jumping / Goto

```c
int main() {
    int a = 10;
    int b = 20;

    goto label1;
    a = a + b;

    label1:
    return 0;
}
```

A label is a place you **might** jump to.
Labels are ignored except for goto/jumps.
(Skipped over if encountered)

```c
int x = 20;
L1:
    int y = x + 30;
L2:
    printf("%d, %d\n", x, y);
```
Unconditional Jumping / Goto

```c
int main() {
    int a = 10;
    int b = 20;
    goto label1;
    a = a + b;

    label1:
    return;
}
```

```asm
push %ebp
mov %esp, %ebp
sub $16, %esp
movl $10, -8(%ebp)
movl $20, -4(%ebp)
jmp label1
movl -4(%ebp), $eax
addl $eax, -8(%ebp)
movl -8(%ebp), %eax
label1:
leave
```
Unconditional Jumping / Goto

- Usage besides GOTO?
Unconditional Jumping / Goto

• Usage besides GOTO?
  – if-then-else (end of “then” clause)
  – infinite loop;
  – break;
  – continue;
  – functions (handled differently)
jmp isn’t very useful by itself...

We’d like to use branch instructions for:
• if/else
• switch
• for loops
• while loops

But if jmp were our only branch instruction, the closest we could get would be an infinite loop.

We need conditional jumps.
Condition Codes (or Flags)

• Set in two ways:
  1. As “side effects” produced by ALU
  2. In response to explicit comparison instructions

• IA-32, condition codes tell you:
  • If the result is zero (ZF)
  • If the result’s first bit is set (negative if signed) (SF)
  • If the result overflowed (assuming unsigned) (CF)
  • If the result overflowed (assuming signed) (OF)
Processor State in Registers

• Temporary data
  \%eax - \%edi

• Location of runtime stack
  \%ebp, \%esp

• Location next instruction \%eip

• Status of recent tests
  \%EFLAGS:
  CF, ZF, SF, OF

- General purpose registers
- Current stack top
- Current stack frame
- Instruction pointer (PC)
- Condition codes
Instructions that set condition codes

1. Arithmetic/logic side effects (addl, subl, orl, etc.)

2. **CMP and TEST:**

   - **cmp l b, a** like computing \( a - b \) without storing result
     - Sets \( OF \) if overflow, Sets \( CF \) if carry-out,
     - Sets \( ZF \) if result zero, Sets \( SF \) if results is negative

   - **test l b, a** like computing \( a \& b \) without storing result
     - Sets \( ZF \) if result is zero, sets \( SF \) if \( a \& b < 0 \)
     - \( OF \) and \( CF \) flags are zero (no overflow with \( \& \) )
Which flags would this `subl` set?

- Suppose `%eax` holds 5, `%ecx` holds 7

```
subl $5, %eax
```

If the result is zero (ZF)
If the result’s first bit is set (negative if signed) (SF)
If the result overflowed (assuming unsigned) (CF)
If the result overflowed (assuming signed) (OF)

A. ZF
B. SF
C. CF and ZF
D. CF and SF
E. CF, SF, and OF
Which flags would this `cmp l` set?

- Suppose `%eax` holds 5, `%ecx` holds 7

```c
cmp l %ecx, %eax
```

If the result is zero (ZF)
If the result’s first bit is set (negative if signed) (SF)
If the result overflowed (assuming unsigned) (CF)
If the result overflowed (assuming signed) (OF)

A. ZF  
B. SF  
C. CF and ZF  
D. CF and SF  
E. CF, SF, and OF
Conditional Jumping

- Jump based on which condition codes are set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>1</td>
<td>Unconditional</td>
</tr>
<tr>
<td>je</td>
<td>ZF</td>
<td>Equal / Zero</td>
</tr>
<tr>
<td>jne</td>
<td>~ZF</td>
<td>Not Equal / Not Zero</td>
</tr>
<tr>
<td>js</td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>jns</td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg</td>
<td>~(SF^OF) &amp; ~ZF</td>
<td>Greater (Signed)</td>
</tr>
<tr>
<td>jge</td>
<td>~(SF^OF)</td>
<td>Greater or Equal (Signed)</td>
</tr>
<tr>
<td>jl</td>
<td>(SF^OF)</td>
<td>Less (Signed)</td>
</tr>
<tr>
<td>jle</td>
<td>(SF^OF)</td>
<td>ZF</td>
</tr>
<tr>
<td>ja</td>
<td>~CF &amp; ~ZF</td>
<td>Above (unsigned jg)</td>
</tr>
<tr>
<td>jb</td>
<td>CF</td>
<td>Below (unsigned)</td>
</tr>
</tbody>
</table>

Jump Instructions: (fig. 3.12)
You do not need to memorize these.
int userval;
scanf("%d", &userval);

if (userval == 42) {
    userval += 5;
} else {
    userval -= 10;
}
...

• Suppose user gives us a value via scanf

• We want to check to see if it equals 42
  • If so, add 5
  • If not, subtract 10
How would we use jumps/CCs for this?

```c
int userval;
scanf("%d", &userval);

if (userval == 42) {
    userval += 5;
} else {
    userval -= 10;
}
```

Assume userval is stored in %eax at this point.
How would we use jumps/CCs for this?

int userval;
scanf("%d", &userval);

if (userval == 42) {
    userval += 5;
} else {
    userval -= 10;
}

... (A) cmpl $42, %eax
    je L2
L1:
    subl $10, %eax
    jmp DONE
L2:
    addl $5, %eax
DONE:
    ...

(B) cmpl $42, %eax
    jne L2
L1:
    subl $10, %eax
    jmp DONE
L2:
    addl $5, %eax
DONE:
    ...

(C) cmpl $42, %eax
    jne L2
L1:
    addl $5, %eax
    jmp DONE
L2:
    subl $10, %eax
DONE:
    ...

Assume userval is stored in %eax at this point.
Loops via \texttt{goto}

Goal: translate for loops and while loops to IA32.

- We know how to translate a for loop to a while loop, so let’s focus on while loops.

- Intermediate step: translate c code with a while loop into c code with \texttt{goto} statements.
Translate `while` → `goto`

```c
int i=1, j=100, k=0;
while(i < j){
    i *= 2;
    j -= i;
}
k = j + i;
```
Translate `goto` $\rightarrow$ IA32

```c
int i=1, j=100, k=0;

L1:
    if(i >= j) goto L2;
    i *= 2;
    j -= i;
    goto L1;

L2:
    k = j + i;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8B00</td>
<td>0</td>
</tr>
<tr>
<td>0x8B04</td>
<td>100</td>
</tr>
<tr>
<td>0x8B08</td>
<td>1</td>
</tr>
<tr>
<td>0x8B0C</td>
<td></td>
</tr>
<tr>
<td>0x8B10</td>
<td></td>
</tr>
</tbody>
</table>

**Hint:**

- `cmp l j`  
- `jge`  
- `jmp`
Translate \texttt{goto} \rightarrow \texttt{IA32}

\begin{verbatim}
int i=1, j=100, k=0;

L1:
    if(i >= j) goto L2;
    i *= 2;
    j -= i;
    goto L1;

L2:
    k = j + i;
\end{verbatim}

\begin{tabular}{|c|c|}
\hline
0x8B00 & 0 & k \\
0x8B04 & 100 & j \\
0x8B08 & 1 & i \\
0x8B0c & \\
0x8B10 & (%ebp) \\
\hline
\end{tabular}

Solution on last slide

\textbf{Hint:}
\begin{verbatim}
cmpl
jge
jmp
\end{verbatim}
Exercise!

Translate the following code in IA32
(Assume x is stored in %eax)

```c
for(i=0; i<10; i++) {
    if (x is even)  // find a clever way to do this!
        x = x / 2;
    else
        x = 3 * x + 1;
}
```
Assignment!

In Written Homeworks section of course web page
Due one week from today
Print pdf and write your answers directly on it
Up Next

Pointers and Memory
Translate \texttt{goto} \rightarrow \texttt{IA32}

```
int i=1, j=100, k=0;

L1:
    if(i >= j) goto L2;
    i *= 2;
    j -= i;
    goto L1;

L2:
    k = j + i;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8B00</td>
<td>0</td>
</tr>
<tr>
<td>0x8B04</td>
<td>100</td>
</tr>
<tr>
<td>0x8B08</td>
<td>1</td>
</tr>
<tr>
<td>0x8B0C</td>
<td></td>
</tr>
<tr>
<td>0x8B10</td>
<td></td>
</tr>
</tbody>
</table>

\textbf{Hint:}
\begin{align*}
\text{cmpl} & \quad j \geq i \\
\text{jge} & \quad \text{jmp (ebp)}
\end{align*}
Translate \texttt{goto} \rightarrow \texttt{IA32}

\begin{verbatim}
movl  -8(%ebp),  %eax
movl  -12(%ebp), %ecx
L1:
cmp   %ecx, %eax
jge   L2
imull $2, %eax
movl  %eax, -8(%ebp)
subl  %eax, %ecx
movl  %ecx, -12(%ebp)
jmp   L1
L2:
addl  %eax, %ecx
movl  %ecx, -16(%ebp)
\end{verbatim}

Possible optimization:
move the value of the variables to memory only after the loop has completed