More Pentium Instructions

October 20

CSC201 Section 002
Fall, 2000
Branching

• For the Pentium, there are both signed and unsigned branch instructions
  - we will ignore (and the book ignores) unsigned branch instructions
# Jump (Branch) Instructions

<table>
<thead>
<tr>
<th>SASM</th>
<th>Pentium</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>br</td>
<td>jmp</td>
<td>(unconditional)</td>
</tr>
<tr>
<td>bg</td>
<td>jg</td>
<td>If greater than</td>
</tr>
<tr>
<td>bgez</td>
<td>jge</td>
<td>If greater than or equal</td>
</tr>
<tr>
<td>bl</td>
<td>jl</td>
<td>If less than</td>
</tr>
<tr>
<td>blez</td>
<td>jle</td>
<td>If less than or equal</td>
</tr>
<tr>
<td>bez</td>
<td>je (or jz)</td>
<td>If equal (or if ZF = 1)</td>
</tr>
<tr>
<td>bnz</td>
<td>jne (or jnz)</td>
<td>If not equal (or if ZF = 0)</td>
</tr>
<tr>
<td></td>
<td>jo, jno</td>
<td>If OF = 1, 0</td>
</tr>
<tr>
<td></td>
<td>jc, jnc</td>
<td>If CF = 1, 0</td>
</tr>
<tr>
<td></td>
<td>js, jns</td>
<td>If SF = 1, 0</td>
</tr>
</tbody>
</table>
I/O Macros

• There are no "real" I/O instructions
  - only calls to the operating system for I/O services

• I/O capabilities provided by Windows + pmacros.inc
  - “put_ch reg/mem” = output least significant byte of operand
  - “get_ch” = input byte into AL register
  - “put_str mem” = output null-terminated string starting at location mem

• No formatting or automatic conversions!
Pentium Debugging

• Use the “View → Debugging → Registers” command in Visual Studio to see the register values

• Use the “Watch” window to change the register values
; For documentation purposes only.

Start the data declaration part of the program.

mystring db "now is the time", 0

.code

main:

; This label defines where program execution will start.

finit
fld op1
frndint
fstsw AX
fstsw AX
fcom ST(1)
fld op3
fld op4
fld op5
fld op6
fld op7

; CPU state information
Name | Value
---|---
this | CXX0017: Error: s
eax | 0x000300e00

; Debug Windows:
- Watch (Alt+3)
- Call Stack (Alt+7)
- Memory (Alt+6)
- Variables (Alt+4)
- Registers (Alt+5)
- Disassembly (Alt+8)
Floating Point

• Warning: I'm omitting a lot of details!
  - including a lot of legal instruction formats

• The floating point unit is logically separate from the rest of the CPU
  - first implemented as a coprocessor

• FPU Status Word
  - bit 14 = C3, bit 10 = C2, bit 8 = C0
Pentium Floating Point Registers

• 8 80-bit registers (double-precision)

• Organized (more or less) as a stack
  - numbered ST(0) (or ST, top of stack) to ST(7) (bottom of stack)
Moving Data To/From FPU

- **fld**: push value onto top of FP stack
  - operand = mem32 / mem64 / ST(i) (i calculated before pushing)
  - also, "fldz" means push zero onto stack

- **fstp**: pop value from top of FP stack
  - operand = mem32 / mem64 / ST(i) (i calculated before pushing)
FP Add/Subtract

- `faddp`: pop top two values, add, push result back onto stack
  - operands = ST(1), ST

- `fsubp`: pop top two values, subtract ST from ST(1), push result back onto stack
  - operands = ST(1), ST

- `fmulp`: pop top two values, multiply, push result back onto stack
  - operands = ST(1), ST

- `fdivp`: pop top two values, divide ST(i) by ST, push result back onto stack
  - operands = ST(1), ST
Other FPU Instructions

- **fchs**: change sign of top of stack
  - no operands

- **frndint**: round value of top of stack to nearest integer
  - no operands
  - fairly easy to convert result into integer representation
Flow of Control

• fcom: compare ST with operand and set C3--C0
  - operands: mem32 / mem64 / ST(i)
• Flag interpretations
  - ST = operand: C3 = 1, C2 = C0 = 0
  - ST > operand: C3 = C2 = C0 = 0
  - ST < operand: C0 = 1, C3 = C2 = 0
• fstsw: store status word into AX register
  - operand: AX required (why?)
  - Then can do bit testing on AX register and jump based on conditions
# Stack-Based Expression Evaluation

**Example:** \( v = w + x \times y - z \)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Top of stack ST</th>
<th>ST(1)</th>
<th>ST(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>finit</em></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><em>fld w</em></td>
<td></td>
<td>w</td>
<td>-</td>
</tr>
<tr>
<td><em>fld x</em></td>
<td></td>
<td>x</td>
<td>w</td>
</tr>
<tr>
<td><em>fld y</em></td>
<td></td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td><em>fmulp</em> ST(1), ST</td>
<td></td>
<td>X\times y</td>
<td>w</td>
</tr>
<tr>
<td><em>faddp</em> ST(1), ST</td>
<td></td>
<td>W+(x\times y)</td>
<td>-</td>
</tr>
<tr>
<td><em>fld z</em></td>
<td></td>
<td>z</td>
<td>W+(x\times y)</td>
</tr>
<tr>
<td><em>fsubp</em> ST(1), ST</td>
<td></td>
<td>W+(x\times y)-z</td>
<td>-</td>
</tr>
<tr>
<td><em>fstp v</em></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
More Expression Evaluation

• Another example

\[ a = b \times (c - d) / (e - b) + 10 \]

\[
\begin{align*}
&\text{fld} \ b \\
&\text{fld} \ c \\
&\text{fld} \ d \\
&\text{fsubp} \ ST(1), \ ST \\
&\text{fmulp} \ ST(1), \ ST \\
&\text{fld} \ e \\
&\text{fld} \ b \\
&\text{fsubp} \ ST(1), \ ST \\
&\text{fdivp} \ ST(1), \ ST \\
&\text{fld} \ 10 \\
&\text{faddp} \ ST(1), \ ST \\
&\text{fstp} \ a
\end{align*}
\]