Branching

CS 217

Condition Codes

- Processor State Register (PSR)

<table>
<thead>
<tr>
<th>31</th>
<th>23</th>
<th>20</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>iCC</td>
</tr>
</tbody>
</table>

- Integer condition codes (icc)
  - N set if the last ALU result was negative
  - Z set if the last ALU result was zero
  - V set if the last ALU result was overflowed
  - C set if the last ALU instruction that modified the iCC caused a carry out of, or a borrow into, bit 31
Condition Codes (cont)

- \( cc \) versions of the integer arithmetic instructions set all the codes
  
  \[
  \begin{align*}
  \text{addcc} & \text{ src1, src2, dst} \\
  \text{subcc} & \text{ src1, src2, dst}
  \end{align*}
  \]

- \( cc \) versions of the logical instructions set only \( N \) and \( Z \) bits
  
  \[
  \begin{align*}
  \text{andcc} & \text{ src1, src2, dst} \\
  \text{orc} & \text{ src1, src2, dst}
  \end{align*}
  \]

Compare and Test Instructions

- Synthetic instructions can set condition codes

<table>
<thead>
<tr>
<th>Synthetic</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{tst reg}</td>
<td>orcc \text{ reg},%g0,%g0</td>
</tr>
<tr>
<td>\text{cmp src1, src2}</td>
<td>subcc \text{ src1, src2, %g0}</td>
</tr>
<tr>
<td>\text{cmp src, value}</td>
<td>subcc \text{ src, value, %g0}</td>
</tr>
</tbody>
</table>

Using \( %g0 \) as the destination discards the result
**Carry and Overflow**

- If the carry bit is set
  - the last addition resulted in a carry, or
  - the last subtraction resulted in a borrow

- Used for multi-word addition
  
  - `addcc %g3, %g5, %g7`
  - `addxcc %g2, %g4, %g6`

(%g6, %g7) = (%g2, %g3) + (%g4, %g5)

- If the overflow bit is set
  - result of subtraction (or signed-addition) doesn't fit

**Branches**

- Tests on the condition codes implement conditional branches and loops

```cpp
If (a == 0)
    a = 1;
else
    a = 2;
```
Branch Instructions

- Transfer control based on \texttt{icc}

\[
\begin{array}{c}
\text{a} \\
\text{n} \\
\text{...} \\
\text{z}
\end{array}
\]

\(\text{b} \{,a\} \hspace{1cm} \text{label}\)

- Target is a PC-relative address: \(\text{PC} + 4 \times \text{disp22}\)
- Where PC is the address of the branch instruction

Branch Instructions (cont)

- Unconditional branches (and synonyms)
  - \texttt{ba} \quad \texttt{jmp} \quad \text{branch always}
  - \texttt{bn} \quad \texttt{nop} \quad \text{branch never}

- Raw condition-code branches
  - \texttt{bnz} \quad \text{!Z}
  - \texttt{bz} \quad \text{Z}
  - \texttt{bpos} \quad \text{!N}
  - \texttt{bneg} \quad \text{N}
  - \texttt{bcc} \quad \text{!C}
  - \texttt{bcs} \quad \text{C}
  - \texttt{bvc} \quad \text{!V}
  - \texttt{bvs} \quad \text{V}
Branching Instructions (cont)

• Comparisons

<table>
<thead>
<tr>
<th>instruction</th>
<th>signed</th>
<th>unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>be</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>bne</td>
<td>!Z</td>
<td>!Z</td>
</tr>
<tr>
<td>bg bgu</td>
<td>!(Z</td>
<td>(N^V))</td>
</tr>
<tr>
<td>ble bleu</td>
<td>Z</td>
<td>(N^V)</td>
</tr>
<tr>
<td>bge bgeu</td>
<td>!(N^V)</td>
<td>!C</td>
</tr>
<tr>
<td>bl blu</td>
<td>N^V</td>
<td>C</td>
</tr>
</tbody>
</table>

Control Transfer

• Instructions normally fetched and executed from sequential memory locations

• PC is the address of the current instruction, and nPC is the address of the next instruction (nPC = PC + 4)

• Branches and control transfer instructions change nPC to something else
Control Transfer (cont)

• Control transfer instructions
  - instruction  type  addressing mode
    - bicc  conditional branch  PC-relative
    - jmpl  jump and link  register indirect
    - rett  return from trap  register indirect
    - call  procedure call  PC-relative
    - ticc  traps  register indirect

PC-relative addressing is like register displacement addressing that uses the PC as the base register

Control Transfer (cont)

• Branch instructions

<table>
<thead>
<tr>
<th>op</th>
<th>a</th>
<th>cond</th>
<th>op2</th>
<th>disp22</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nPc = PC + signextend(disp22) << 2

• Calls

<table>
<thead>
<tr>
<th>op</th>
<th>disp30</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td></td>
</tr>
</tbody>
</table>

nPc = PC + signextend(disp30) << 2

position-independent code does not depend on where it's loaded; uses PC-relative addressing
Branching Examples

• if-then-else
  if (a > b)
    c = a;
  else
    c = b;

• Loops
  for (i=0; i<n; i++)
    . . .

Branching Examples (cont)

• Loops
  for (i=0; i<n; i++)
    . . .

  #define i %10
  #define n %11
  clr i
  L1: cmp i, n
      bge L2; nop
      . . .
      inc i
      ba L1; nop
  L2:  ...
### Branching Examples (cont)

- Alternative implementation

```assembly
for (i=0; i<n; i++)
  ...
```

```assembly
#define i %10
#define n %11
clr i
ba L2; nop
L1: ...
  inc i
L2: cmp i,n
  bl L1; nop
```