### Control Flow

The flow of control is
- the sequence of instruction executions performed by a program
- every program execution can be described by such a linear sequence
- Controlling flow in languages like Java

### Conditional branches
- goto <address> if <condition>

### Options for evaluating condition
- unconditional
- conditional based on value of a register (==0, >0 etc.)

### Conditional jump
- goto <address> if <condition>
  - conditional check result of last executed ALU instruction
  - goto <address> if <condition>

### Specifying target address
- absolute 32-bit address
  - this requires a 6 byte instruction, which means jumps have high overhead
  - is this a serious problem? how would you decide?

### ISA requirement (apparently)
- at least one PC-relative jump
  - specifies jump target using full 32-bit address
  - use when the jump distance is too large to fit in a byte

### ISA for Static Control Flow (part 1)
- at least one PC-relative jump
  - specifies relative distance using real distance / 2 — why?
  - at least one absolute jump
  - conditional jumps (at least == and > 0)

- Make these PC-relative — why?

### New instructions (so far)
- jump assembly uses label, not direct hex number
- PC-relative count starts from next instruction, after fetch increments PC

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Name</th>
<th>Semantics</th>
<th>Assembly</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch pc ← (a=pc+oo*2)</td>
<td>jump immediate</td>
<td>pc ← a (specified as label)</td>
<td>a ← pc+oo</td>
<td>laba</td>
</tr>
<tr>
<td>branch if &lt;condition&gt; pc ← (a=pc+oo*2)</td>
<td>branch if equal</td>
<td>pc ← a (pc=pc+oo*2) if &lt;condition&gt;</td>
<td>a ← pc+oo</td>
<td>laba</td>
</tr>
<tr>
<td>branch if &lt;condition&gt; pc ← (a=pc+oo*2)</td>
<td>branch if greater</td>
<td>pc ← a (pc=pc+oo*2) if &lt;condition&gt;</td>
<td>a ← pc+oo</td>
<td>laba</td>
</tr>
</tbody>
</table>

### ISA for Static Control Flow (part 2)
- at least one absolute jump (0x0)
- unconditional
  - unconditional jumps are two bytes
- conditional jumps (at least == and > 0)
  - conditional jumps are two bytes

### Code Examples

#### Java
- a method is a sub-routine with a name, arguments and local scope
- method invocation causes the sub-routine to run with values bound to arguments and with a possible result bound to the invocation

#### C
- a procedure is
- a procedure call is
Diagramming a Procedure Call

- Caller
  - goto ping
  - j ping

- Callee
  - do whatever ping does
  - goto foo just after call to ping()

- continue executing

Questions
- How is RETURN implemented?
- It's a jump, but is the address a static property or a dynamic one?

Implementing Procedure Return

- return address is
  - the address the procedure jumps to when it completes
  - the address of the instruction following the call that caused it to run
  - a dynamic property of the program

- questions
  - how does procedure know the return address?
  - how does it jump to a dynamic address?

ISA for Static Control Flow (part 2)

- New requirements
  + read the value of the PC
  + jump to a dynamically determined target address

Complete new set of instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Semantics</th>
<th>Assembly</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch</td>
<td>pc ← (a==pc+pp*2)</td>
<td>br a</td>
<td>8-pp</td>
</tr>
<tr>
<td>branch if equal</td>
<td>pc ← (a==pc+pp*2) if r[c]==0</td>
<td>beq a</td>
<td>9cpp</td>
</tr>
<tr>
<td>branch if greater</td>
<td>pc ← (a==pc+pp*2) if r[c]&gt;0</td>
<td>bgt a</td>
<td>acpp</td>
</tr>
<tr>
<td>jump immediate</td>
<td>pc ← a (a specified as label)</td>
<td>j a</td>
<td>--------</td>
</tr>
<tr>
<td>get pc</td>
<td>r[d] ← pc + (o==p*2)</td>
<td>gpc $o,rd</td>
<td>6fpd</td>
</tr>
<tr>
<td>jump base+offset</td>
<td>pc ← r[t] + (o==pp*2)</td>
<td>j o(rt)</td>
<td>ctpp</td>
</tr>
</tbody>
</table>

Compiling Procedure Call / Return

- void foo () {
  ping ();
}
- void ping () {}