CPSC 213
Introduction to Computer Systems
Unit 1d
Static Control Flow

1. General form
- in Java and C
  - if <condition> <then-statements> else <else-statements>
- pseudo-code... goto then if (temp_c==0)
else: <else-statements>
then: <then-statements>
end_if:

ISA for Static Control Flow
(part 1)

- ISA requirement (apparently)
  - at least one PC-relative jump
    - specify jump target as a delta from address of current instruction
    - unconditional jumps for for/while/if etc. normally jump to a nearby instruction
    - some jumps such as for/while/if etc. different from jumps for procedure call?

Implement loops in machine

- Can we implement this loop with the existing ISA?
  - int  s=0;
  - int  i;
  - int a[] = {2,4,6,8,10,12,14,16,18,20};
  - void foo () {
    for (i=0; i<10; i++) 
      s += a[i];
  }

PC Relative Addressing

Will this technique generalize

Absolute Addressing

New instructions (so far)

Condition Codes - Loops

Observation

Loop unrolling
- Using array syntax
- Using pointer-arithmetic syntax for access to a?
- Will this technique generalize

Using array syntax
- int  s=0;
- int  i;
- int a[] = {2,4,6,8,10,12,14,16,18,20};
- void foo () {
  for (i=0; i<10; i++)
    s += a[i];
}

Control-Flow ISA Extensions

IC: instruction counter, R: register

<table>
<thead>
<tr>
<th>Name</th>
<th>Semantics</th>
<th>Assembly Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch pc ← (a==pc+oo*2) br a 8-oo</td>
<td>branch if equal pc ← (a==pc+oo*2) if r[c]==0 beq rc, a 9coo</td>
<td>branch if greater pc ← (a==pc+oo*2) if r[c]&gt;0 bgt rc, a acoo</td>
</tr>
<tr>
<td>jump pc ← a j a b--- aaaaaaaa</td>
<td>jump pc ← a j a b--- aaaaaaaa</td>
<td>jump pc ← a j a b--- aaaaaaaa</td>
</tr>
</tbody>
</table>
Diagramming a Procedure Call

Caller
• goto ping
  - j ping
• continue executing

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

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?

saving the return address
• only the caller knows the address
• so the caller must save it before it makes the call
• we need a new instruction to read the PC
  • we’ll call it gpc

jumping back to return address
• we need new instruction to jump to an address stored in a register
  • callee can assume return address is in r6

Compiling Procedure Call / Return

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

foo:   ld   $ping, r0 # r0 = address of ping ()
gpc  r6    # r6 = pc of next instruction
inca r6    # r6 = pc + 4
j    (r0) # goto ping ()

ping:  j    (r6) # return

ISA for Static Control Flow

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+oo*2)</td>
<td>br a</td>
<td>br-oo</td>
</tr>
<tr>
<td>branch if equal</td>
<td>pc += (a==pc+oo*2) if (r[c]==0)</td>
<td>beq a</td>
<td>b-eoo</td>
</tr>
<tr>
<td>branch if greater</td>
<td>pc += (a==pc+oo*2) if (r[c]&gt;0)</td>
<td>bgt a</td>
<td>b-goo</td>
</tr>
<tr>
<td>jump</td>
<td>pc += a</td>
<td>j a</td>
<td>j-oo</td>
</tr>
<tr>
<td>get pc</td>
<td>r[d] ← pc</td>
<td>gpc rd</td>
<td>gpc-oo</td>
</tr>
<tr>
<td>indirect jump</td>
<td>pc ← (r[t] + (o == pp*2))</td>
<td>j o(rt)</td>
<td>j o-pp</td>
</tr>
</tbody>
</table>

Name Semantics Assembly Machine
branch pc ← (a==pc+oo*2) br a br-oo
branch if equal pc ← (a==pc+oo*2) if (r[c]==0) beq a b-eoo
branch if greater pc ← (a==pc+oo*2) if (r[c]>0) bgt a b-goo
jump pc ← a j a j-oo
get pc r[d] ← pc gpc rd gpc-oo
indirect jump pc ← (r[t] + (o == pp*2)) j o(rt) j o-pp