CPSC 213

Introduction to Computer Systems

Unit 1d Static Control Flow

Readings for Next 2 Lectures

Textbook

Condition Codes - Loops

• 3.6.1-3.6.5

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

3

Loops (S5-loop)
∙ In Java	<pre>public class Foo { static int s = 0; static int i; static int a[] = new int[10]; static void foo () { for (i=0; i<10; i++) s += a[i]; } }</pre>
∙ In C	<pre>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]; }</pre>

Implement loops in machine

```
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];
}
```

• Can we implement *this* loop with the existing ISA?

5

Loop unrolling

• Using array syntax

```
int s=0;
int i;
int a[10] = {2,4,6,8,10,12,14,16,18,20};
void foo () {
    i = 0;
    s += a[i];
    i++;
    s += a[i];
    i++;
    ...
    s += a[i];
    i++;
}
```

- Using pointer-arithmetic syntax for access to a?
- Will this technique generalize
 - will it work for all loops? why or why not?

Control-Flow ISA Extensions

Conditional branches

- goto <address> if <condition>
- Options for evaluating condition
 - unconditional
- conditional based on value of a register (==0, >0 etc.)
 - goto <address> if <register> <condition> 0
- conditional check result of last executed ALU instruction
 - goto <address> if last ALU result <condition> 0

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?
- are jumps for for/while/if etc. different from jumps for procedure call?

7

PC Relative Addressing

Motivation

- jumps are common and so we want to make them as fast as possible
- small instructions are faster than large ones, so make some jumps be two bytes
- Observation
- some jumps such as for/while/if etc. normally jump to a nearby instruction
- ${\ensuremath{\bullet}}$ so the jump distance can be described by a small number that could fit in a byte
- PC Relative Addressing
 - specifies jump target as a delta from address of current instruction (actually next)
 - \bullet in the execute stage $\ensuremath{\textit{pc register}}$ stores the address of next sequential instruction
 - the pc-relative jump delta is applied to the value of the pc register
 - jumping with a delta of 0 jumps to the next instruction
 - \bullet jump instructions that use pc-relative addressing are called $\ensuremath{\textit{branches}}$
- Absolute Addressing
 - specifies jump target using full 32-bit address
 - $\ensuremath{\bullet}$ use when the jump distance too large to fit in a byte

ISA for Static Control Flow (part 1)

ISA requirement (apparently)

- at least one PC-relative jump
 - specify relative distance using real distance / 2 why?
- at least one absolute jump
- some conditional jumps (at least = and > 0)
 - make these PC-relative why?
- New instructions (so far)

Name	Semantics	Assembly	Machine
branch	pc ← (a ==pc+ oo * 2)	br a	8–00
branch if equal	pc ← (a==pc+oo*2) if r[c]==0	beg r c, a	9coo
branch if greater	pc ← (a ==pc+ oo *2) if r[c]>0	bgt r c, a	acoo
jump	pc ← a	ја	b aaaaaaaa

9

This example

• pseudo code template

loop:	i=0 goto end_loop if not (i<10) s+=a[i] i++ goto loop
end_loop:	

• ISA suggest two transformations

- only conditional branches we have compared to 0, not 10
- no need to store i and s in memory in each loop iteration, so use temp_ to indicate this

loop:	<pre>temp_i=0 temp_s=0 temp_t=temp_i-10 goto end_loop if temp_t==0 temp_s+=a[temp_i]</pre>
end_loop:	temp_i++ goto loop s=temp_s i=temp_i

11

loop:	tem tem got tem	p_i=0 p_s=0 p_t=temp_i-10 o end_loop if te p_s+=a[temp_i]	mp_t==0	
end_loop:	got s=t	p_i++ o loop emp_s emp_i		
sembly cod	е	Assume that all v	ariables are	e global variables
	1d	\$0x0, r0 \$a, r1	# r1 =	address of a[0]
loop:	vom add	r5, end_loop (r1, r0, 4), r3	# r5 = # r5 = # if te # r3 =	temp_i temp_i=10

Implementing if-then-else (S6-if)

if (a>b) max = a; else max = b;

• General form

• in Java and C

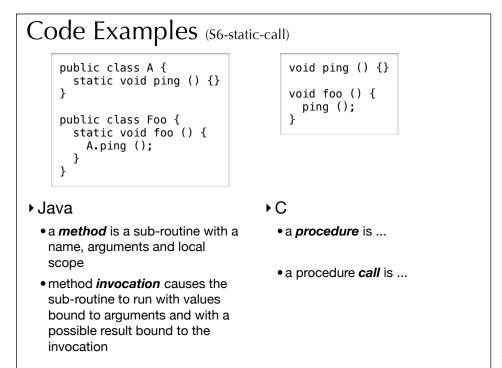
- if <condition> <then-statements> else <else-statements>

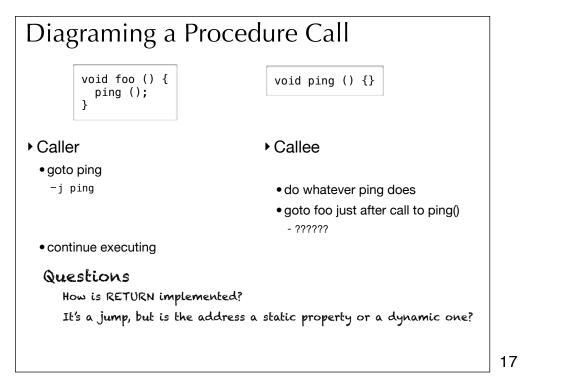
• pseudo-code template

else: then: end_if:	<pre>temp_c = not <condition> goto then if (temp_c==0) <else-statements> goto end_if <then-statements></then-statements></else-statements></condition></pre>
end_if:	

This exam A This exam A S A S A S A S A S A S A S A S A S A S A	ple	
 pseudo-co 	de template	
else: then: end_if:	<pre>temp_a=a temp_b=b temp_c=temp_a-temp_b goto then if (temp_c>0) temp_max=temp_b goto end_if temp_max=temp_a max=temp_max</pre>	
 assembly c 	code	
else: then: end_if:	mov r1, r2 not r2 inc r2 add r0, r2 bgt r2, then mov r1, r3 br end_if mov r0, r3	<pre># r0 = &a # r0 = a # r1 = &b # r1 = b # r2 = b # temp_c = ! b # temp_c = a-b # if (a>b) goto +2 # temp_max = b # goto +1 # temp_max = a # r0 = &max # max = temp_max</pre>

Static Procedure Calls





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
 - caller will save the return address in ${\bf r6}$
 - $\ensuremath{\cdot}$ there is a bit of a problem here if the callee makes a procedure call, more later \ldots
- 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

19

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

Name	Semantics	Assembly	Machine
branch	pc ← (a ==pc+ oo *2)	br a	8–00
branch if equal	pc ← (a ==pc+ oo *2) if r[c]==0	beg a	9coo
branch if greater	pc ← (a ==pc+ oo *2) if r[c]>0	bgt a	acoo
jump	pc ← a	j a	b aaaaaaaa
get pc	r[d] ← pc	gpc r d	6f-d
indirect jump	pc ← r[t] + (o == pp *2)	j o (r t)	ctpp