Toy Instruction Set Architecture

CS 2130: Computer Systems and Organization 1

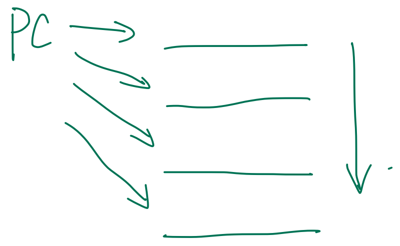
Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 3 due Monday at 11:59pm on Gradescope
- Midterm 1 next Friday (February 20) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP

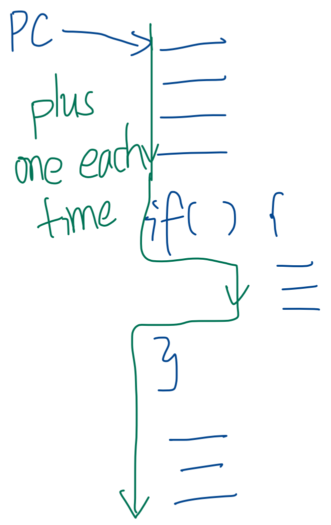
Jumps (control constructs)

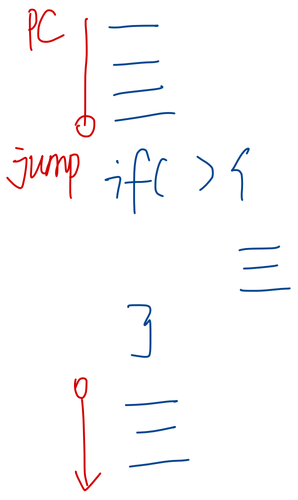


- Moves and math are large portion of our code
- We also need **control constructs**
  - Change what we are going to do next
  - if, while, for, functions, … in terms of machine code, these codes called jumps
- Jumps provide mechanism to perform these control constructs
- We jump by assigning a new value to the program counter PC

Jumps

- For example, consider an if





when we got "if", we have a choice

① Continue our code, line by line

② don't want to do the if body, magic teleport, teleports me down to the end of the if statement.

(if the first line after if has index 25, instead of PC+1, I'll say, PC=25)

Jumps

we make all our registers 8 bits, each register will hold one byte.

icode	meaning
7	Compare `rA` as 8-bit 2's-complement to 0 if `rA <= 0` set pc `= rB` else increment pc as normal

Instruction icode 7 provides a **conditional** jump

jumps if some condition is true. Specifically, we read the value in rA.

- Real code will also provide an **unconditional** jump, but a conditional jump is sufficient

just set rA to 0.

Writing Code

We can now write any* program! We are basically being what we called a "compiler"

- When you run code, it is being turned into instructions like ours
  $\hookrightarrow$ just some binaries.
- Modern computers use a larger pool of instructions than we have (we will get there)
  We have 14 instructions, modern computers can have thousands of instructions.

*we do have some limitations, since we can only represent 8-bit values and some operations may be tedious.

Our code to this machine code

How do we turn our control constructs into jump statements?

$$\begin{tikzcd}
\boxed{\text{if}} \arrow[r, bend right] & \boxed{\text{if-else}} & \boxed{\text{for( )}} \arrow[rr, bend right] & \boxed{\begin{array}{c}\text{for(x in y)}\\ \text{interation version}\end{array}} \arrow[l, bend left] & \boxed{\text{while}} & \boxed{\text{do-while}} \arrow[l, bend left]
\end{tikzcd}$$

how to convert for to while?

```
for (int i=0; i<25; i++){
        ≡
      }
```

$\Rightarrow$

```
int i=0
while(i<25) {
        ≡
     i++; }
```

In C/Java, we have if/else. But from the CPU perspective, there is no such thing as if/else — only sequential execution and jumps. Think of the CPU as walking forward on a road, Default behavior: keep moving forward. Only when we do NOT want to continue forward, we need to jump somewhere else. So the compiler naturally ask "When should I jump away?" instead of "When should I continue?"

For human: if D is true → A, otherwise → B. But for machine: 1. The CPU will continue downward by default. 2. If D is false, where should we go?

it looks like we're "thinking backwards", but actually we are just following the CPU's natural rule: continue unless force to jump.

if/else to jump

```
if ( D ) {
    [ A ]
} else {
    [ B ]
}
[ C. ]
```

if condition D is true, I will do A, skip B, continue to do C

```
if( !D ), jump to B
[ A ]
jump to C (unconditional jump)
[ B ]

[ C ]
```

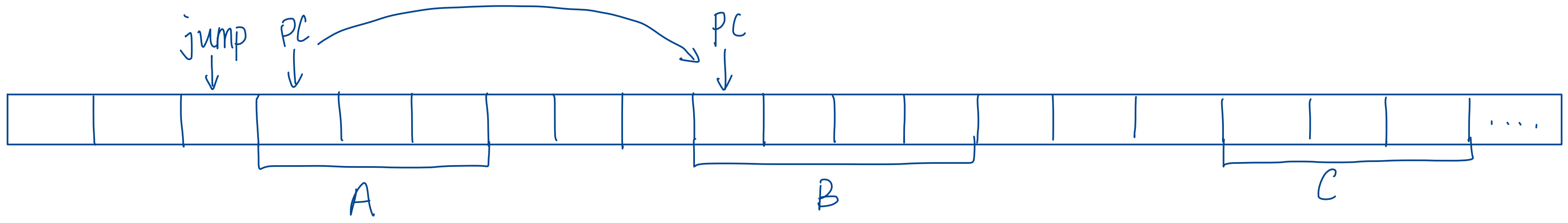
2 situations:
①. if D is true: don't jump, continue A, then C.
②. if D is false: jump to B, then C.

jumps happens at 2 places where?

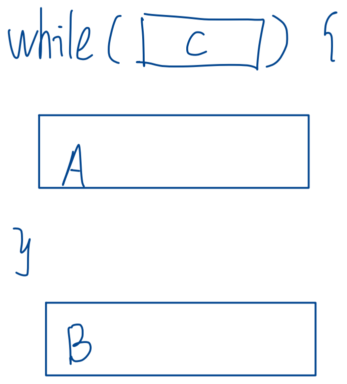
Why do compilers generate code this way?
①. "Fewer jumps" ⇒ faster execution
②. "Sequential flow" ⇒ better branch prediction
③. "This pattern appears in all architectures".

if/else to jump How does it work in memory?

my code is going to be in memory, one byte for each. (If this is true, then jump to index xxx).

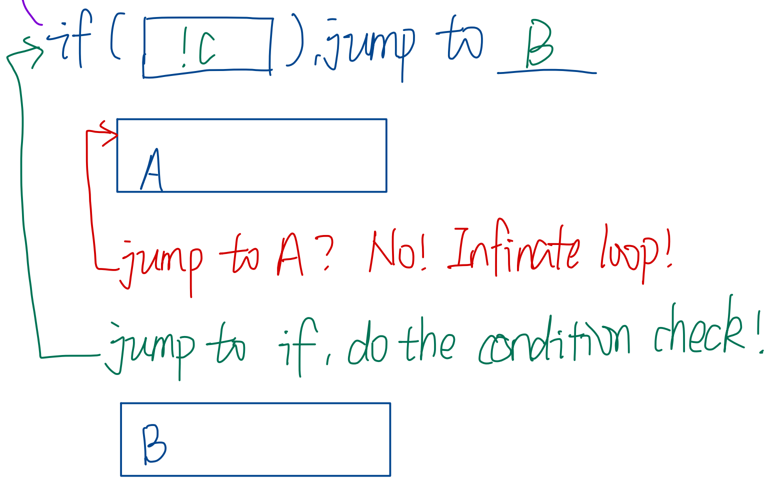


while to jump



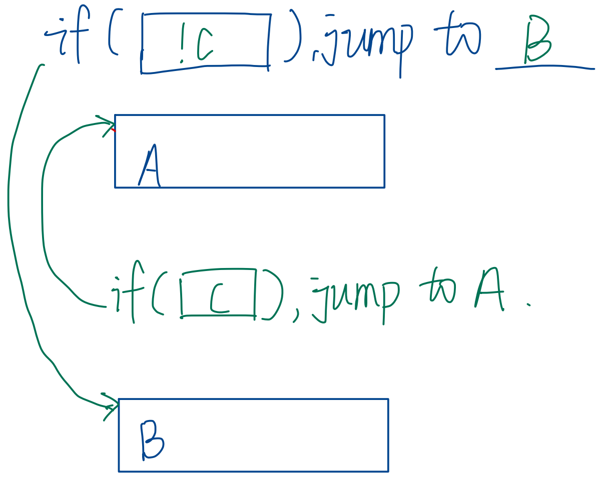
```
while ( C ) {
    A
}
B
```

How do I know where to jump?
① hard code if you know the address
② icode 3-3



```
if ( !C ), jump to B
    A
    jump to A?  No! Infinite loop!
    jump to if, do the condition check!
B
```

each loop iteration has 2 jump checks.



```
if ( !C ), jump to B
    A
    if( C ), jump to A.
B
```

each loop iteration only has 1 jump check.

It's not the same loop

It's a do-while loop!

UNIVERSITY of VIRGINIA

Encoding Instructions

icode	b	meaning
0		`rA = rB`
1		`rA &= rB`
2		`rA += rB`
3	0	`rA = ~rA`
	1	`rA = !rA`
	2	`rA = -rA`
	3	`rA = pc`
4		`rA` = read from memory at address `rB`
5		write `rA` to memory at address `rB`
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` &= read from memory at `pc + 1`
	2	`rA` += read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`
		For icode 6, increase pc by 2 at end of instruction
7		Compare `rA` as 8-bit 2's-complement to 0
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

Example 3: if r0 < 9 jump to 0x42

I don't have an instruction say r0<9.
I need "r0<=0" for icode 7, what should I do?

$r0<9 \Leftrightarrow r0<=8 \Leftrightarrow (r0-8)<=0$

$\Leftrightarrow$ r0+=-8 (0xF8)
r0<=0

r1 = 0x42 $\quad \underbrace{0\,110}_{6}\ \underbrace{01\,00}_{4}\ \underbrace{42}_{42}$

r0 += F8 $\quad \underbrace{0\,110}_{6}\ \underbrace{00\,10}_{2}\ \underbrace{F8}_{F8}$

if r0<=0, PC=r1 $\quad \underbrace{0\,111}_{7}\ \underbrace{\overset{(R0)}{00}\ \overset{(R1)}{01}}_{1}$

64 42 62 F8 71

register : ir : current instruction.
PC : address of the next instruction