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, 2026) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP
  - Review session in class next Wednesday

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

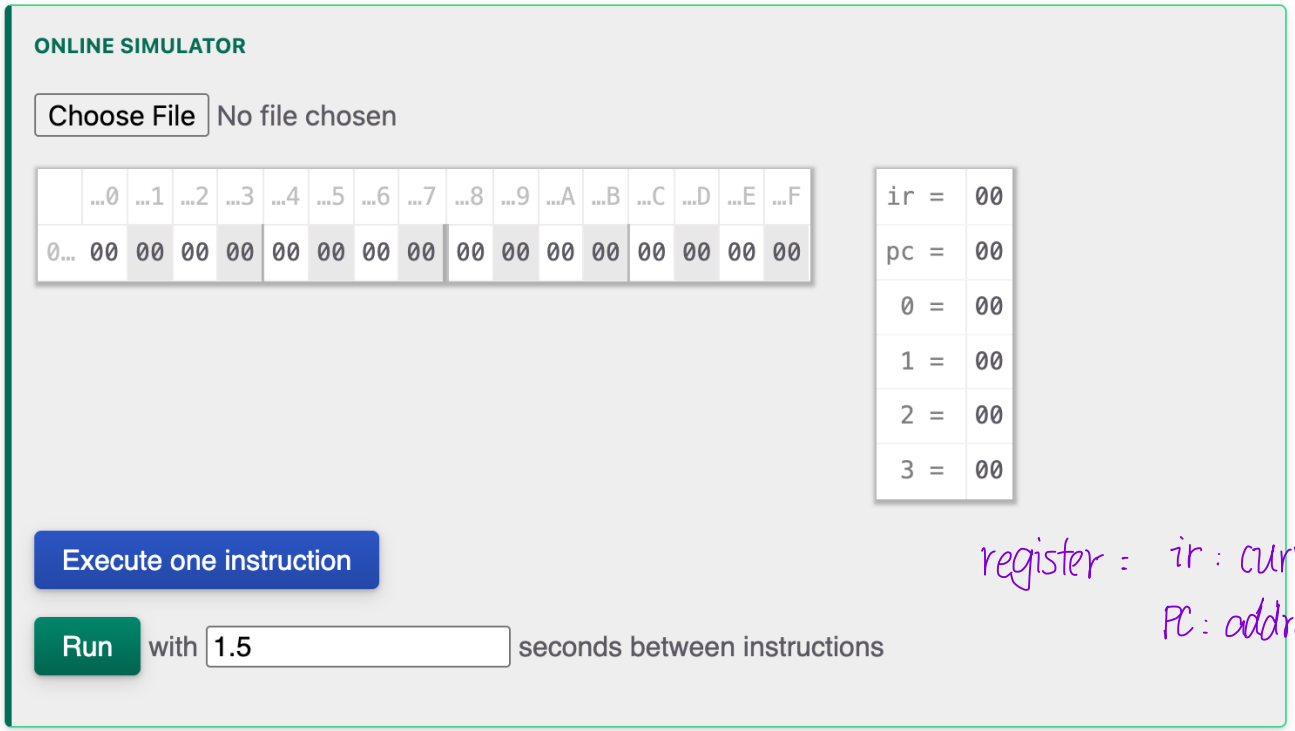
r1 = 0x42 — 0 110 01 00 — 42
6 4 42

r0 += F8 — 0 110 00 10 — F8
6 2 F8

if r0<=0, PC=r1 — 0 111 00 01
(R0) (R1)
7 1

b442b2F871

Online Simulator



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

https://uva-cs.github.io/cso1-s26/homework/hw3-product.html

Homework Hints

1. Write pseudocode that does the desired task.
2. Deal with control flow.
3. Split multi-operation lines into a series of single-operation lines.
   - x = y − z; becomes x = y; x -= z;
4. Convert operations to those in our instruction set.
   - x -= z; becomes w = z; w = -w; x += w;
5. Deal with loops.
6. Assign variables to our four registers.
   - Example: r0 = x, r1 = y, r2 = z, r3 = w
   - r0 = r1; r3 = r2; r3 = -r3; r0 += r3
7. Write those instructions into triples, then hex.

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 4: 0x17 * 3

0x17 * 3 ⇒ 0x17+0x17+0x17 ⇒

for(i=0;i<3;i++) x+=0x17;

```
x=0;
i=0;
while(i<3){
  x+=0x17;
  i+=1;
}
```

```
x=0;
i=-2
//HERE
x+=0x17
i+=1;
if(i<3) jump HERE
if(i<=2)
if(i-2<=0)
if(i<=0) jump HERE
```

icode 3-3: rA=PC (Save where to back)

0011 / 3 | 10 / r2 | 11 / 3

what values we need?

x → r0

i → r1

HERE → r2

y=0x17 → immediate

Dealing with Variables and Memory

What if we have many variables? Compute: x += y

We only have 4 registers! ⇒ save them in memory!

x	0x80
y	0x81
z	0x82
m	0x83
n	0x84
p	0x85
foo	0x86

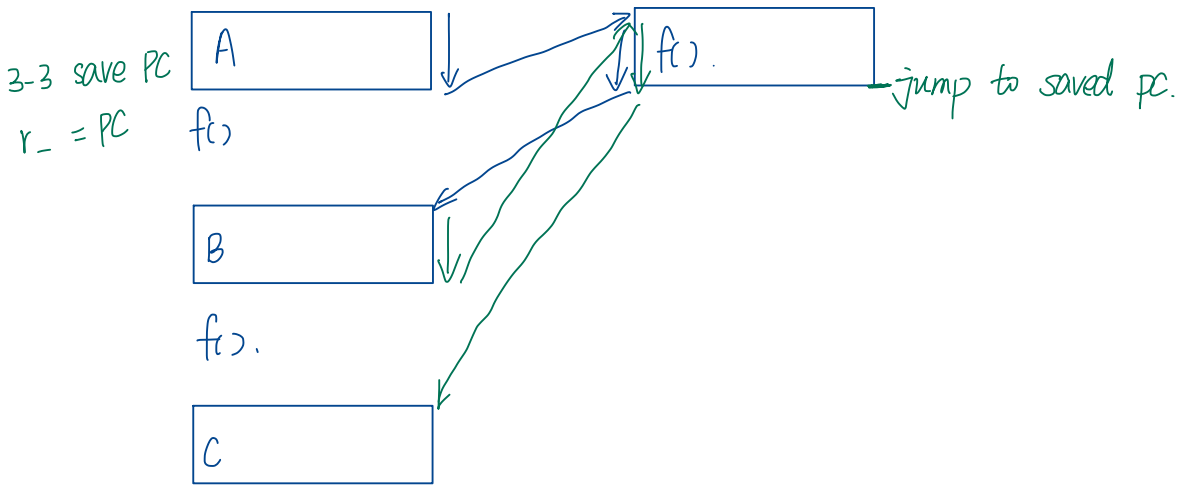
```
r0 = M[0x80]   // Now r0 has x
r1 = M[0x81]   // Now r1 has y
r0 += r1       // do the calculation
M[0x80] = r0   // store x back to memory
M[0x81] = r1   // store y back to memory (No need, optimization later).
```

x += z: r0 = M[0x80] // read x again, also not necessary, optimization later.

No matter how many variables I have, I only need 3 registers to do whole thing

2: operating on variables 1: address

Function Calls



Exercises

Q5.3 XOR

1 Point

Suppose we then shift it back and xor it with the original, like

```
((0xCA >> 3) << 3) ^ 0xCA.
```

For signed:

0 XXX XXXX

>>3. 0 000 XXXX

<<3: 0XXX X000

The result is:

- (●) the same for both signed and unsigned integers
- ( ) larger for signed than unsigned integers
- ( ) larger for unsigned than signed integers
- ( ) there is no way to know

0XXX X000

^ 0XXX XXXX

XXX

0^1=1

0^0=0

when you XOR any bit x with 0, the result is always x itself.

similar thing for negatives.

Exercises

Q8 Counter

1 Point

To build a 4-bit counter circuit, we could directly connect the output of the increment circuit back to the input.

No!
Oscillate unpredictably.

- ○ True
- ◉ False

Exercises

Q4 Floating Point

2 Points

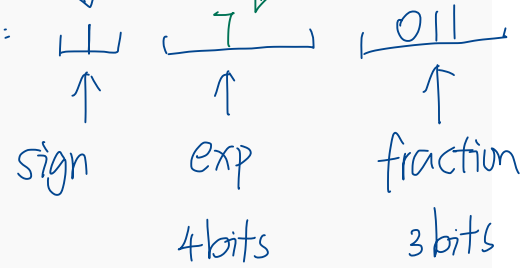
Assume we will use 8-bit floating-point numbers with **3 fraction bits**. How would we encode the binary number `-010110000` into this 8-bit floating point representation?

- ( ) `0 011 1110`
- ( ) `1 011 1110`
- ( ) `0 0111 011`
- ( ) `0 1110 011`
- ( ) `1 0111 011`
- (•) `1 1110 011`

step1: scientific expression:

-1.011×10^{7}

step2:



step3: what is 7 in biased?

$$\begin{array}{r} 0111 \\ +\,0111 \\ \hline 1110 \end{array}$$

0111 ← 2's complement

0111 ← bias

so we get:

1 1110 011

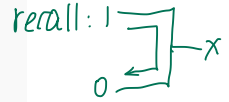
Exercises

Q3 Coding hardware

1 Point

When coding in a hardware description language (code that can be turned into circuits), there are no typical control constructs like `if`, `while`, and `for`; in addition, which of the following is **not** permitted?

- ○ Accessing the same variable twice, like `y = x + 1; z = x - 1;`
- ◉ Assigning to the same variable twice, like `y = x + 1; y = w - z;` → you can't assign 2 values to one variable.
- ○ Conditional operations, like `y = (x < 0) ? x : -x;` Multiplexer
- ○ Including several operators in a single expression, like `y = (x + y) ^ z;`

recall: 

Exercises

Q5 Cycles

1 Point

In class, we built a computer that we could program with 1-byte instructions, containing an icode, source, and destination registers. In each cycle, the logic circuits **only** calculated the operation specified by the given icode, which would be later written to a register.

- ○ True
- ◉ False

Do all the calculations at the same time, the icode will select from the results.

Building a Computer