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
- Quiz 4 available today, due Sunday at 11:59pm
- Midterm 1 next Friday (October 3, 2025) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP

High-level Instructions

In general, 3 kinds of instructions

- **moves** - move values around without doing “work”
- **math** - broadly doing “work”
- **jumps** - jump to a new place in the code

Encoding Instructions

idea: ①. I have a value in memory at address hex 82
②. I want to add whatever in R3 to that value.

Example 2: M[0x82] += r3
Read memory at address 0x82, add r3, write back to memory at same address

One point: No instructions allow us to pass an immediate value as the address.
So let's save ourselves some time: just put 82 in a register.
Then we use icode 4 to read it out, icode 5 to write it back.

r2 = 0x82	0 110 (6)	10 00 (8)	82
r1 = M[r2]	0 100 (4)	01 10 (6)	
r1 += r3	0 010 (2)	01 11 (7)	
M[r2] = r1	0 101 (5)	01 10 (6)	

our machine reads 0 and 1.
But, for us → easier to read → pairs of hex
68 82 46 27 56

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 +`
		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 ⟺ r0<=8 ⟺ (r0−8)<=0
⟺ r0+=−8 (0xF8)
r0<=0

r1 = 0x42 — 0 110 01 00 → 6 4; 42 → 42

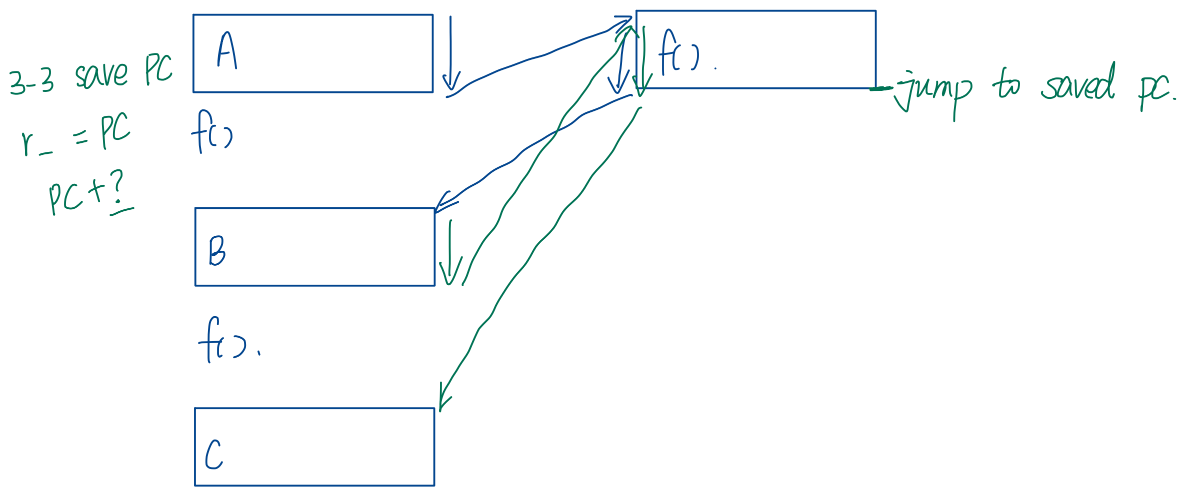
r0 += F8 — 0 110 00 10 → 6 2; F8 → F8

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

64 42 62 F8 71

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

Function Calls



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]=r1     // store x back to memory
M[0x81]=r0     // 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

Arrays

Array: a sequence of values (collection of variables)

In Java, arrays have the following properties:

- Fixed number of values
- Not resizable
- All values are the same type

Arrays

Array: a sequence of values (collection of variables)

In Java, arrays have the following properties:

- Fixed number of values
- Not resizable
- All values are the same type

How do we store them in memory?

Arrays

Storing Arrays

In memory, store array sequentially

- Pick address to store array
- Subsequent elements stored at following addresses
- Access elements with math

Example: Store array $\boldsymbol{arr}$ at **0x90**

- Access $\boldsymbol{arr}$[3] as **0x90 + 3** assuming 1-byte values

Quiz Questions – Quiz 1

Q5.3 XOR

1 Point

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

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

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

For signed:
0 XXX XXXX
>>3. 0 000 XXXX
<<3: 0XXX X000

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.

Quiz Questions – Quiz 2

Q4 Floating Point

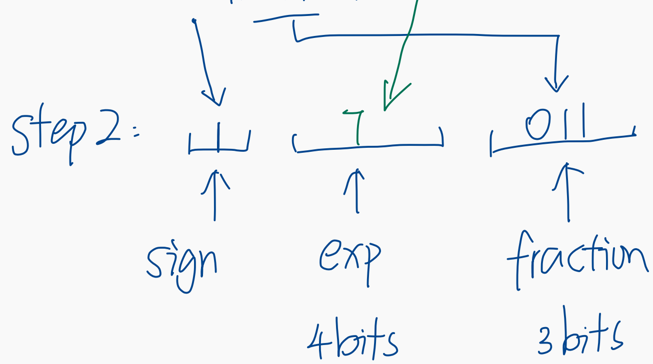
2 Points

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}



step3: what is 7 in biased?

```
  0111  ← 2's complement
+ 0111  ← bias
------
  1110
```

so we get:

1 1110 011

Quiz Questions – Quiz 3

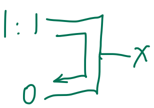
Q3 Coding hardware

1 Point

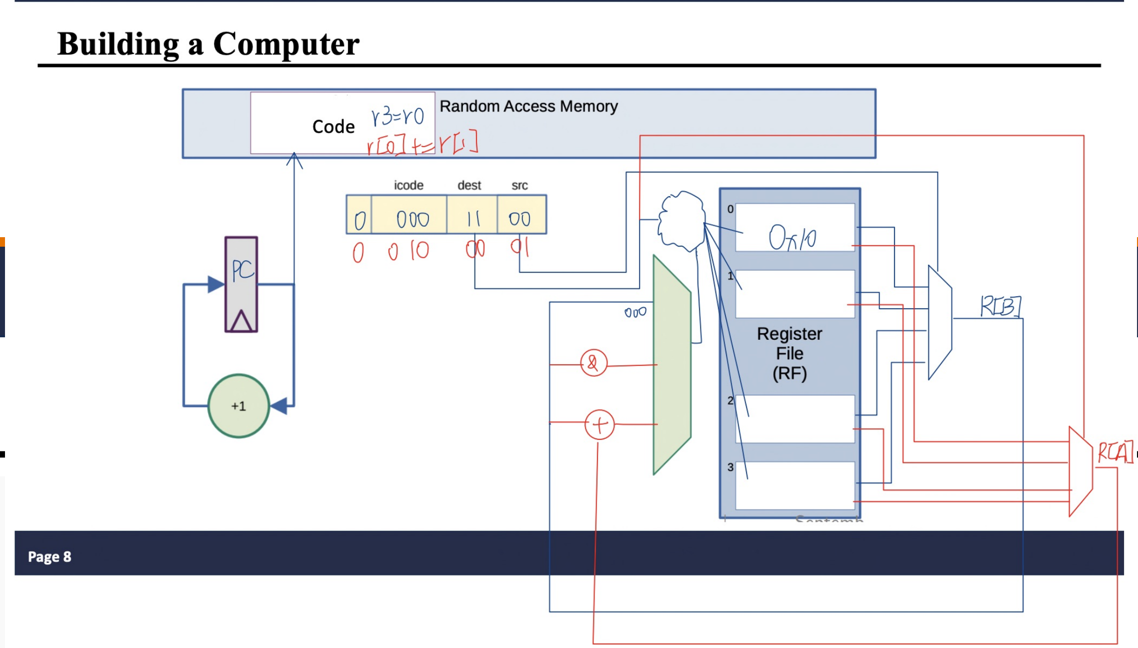
When coding in a hardware description language (code that can be turned into circuits), there are no typical control constructs like `if`, `while`, and `for`; in addition, which of the following is **not** permitted?

- ○ 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:



Quiz Questions – Quiz 3



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. (See

Quiz Questions – Quiz 3

Q6 Coding ToyISA

1 Point

In our example instruction set (Toy ISA) from class, encoding an operation may mean writing one or more instructions that collectively have exactly the same result as the operation we want. For example, to encode the operation `x = ~y`, we may encode it as `x = y; x = ~x;` (icode 0 then icode 3.0). However, while `y = ~y; x = y;` (icode 3.0 then icode 0) has the same effect, it would also modify y as well as x.

The source code operation `x = y + z;` could be implemented (efficiently using our instruction set as:

x=y

x+=z

- ○ One instruction
- ◉ Two instructions
- ○ Three or more instructions
- ○ It cannot be implemented with the Toy ISA instructions

Quiz Questions – Quiz 3

Q7 ToyISA Encoding

2 Points

In our example instruction set (Toy ISA) from class, which of the following programs will compute `r0 = r0 - r2`?

- ○ 3a 02
- ◉ 06 36 21
- ○ 3a 12 00
- ○ 06 34 21
- ○ None of the above

idea 1: find all the instruction sets and encode them

for example:

r2 =-r2
r0+=r2

or

r1 =r2
r1=-r1
r0 +=R1

you just find all possible solutions and encode them to find the answer.

idea 2: check each option to see what this binary do.

0000 0110	0011 0110	0010 0001
R1 =R2	R1 =-R1	R0+=R1

Quiz Questions – Quiz 3

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