Midterm 1 Review

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

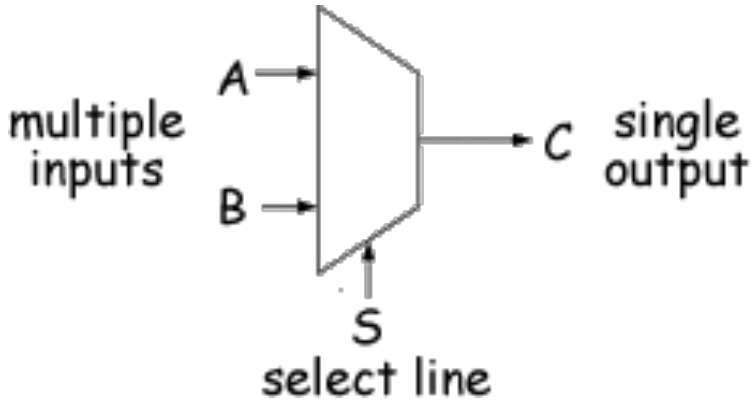
Announcements

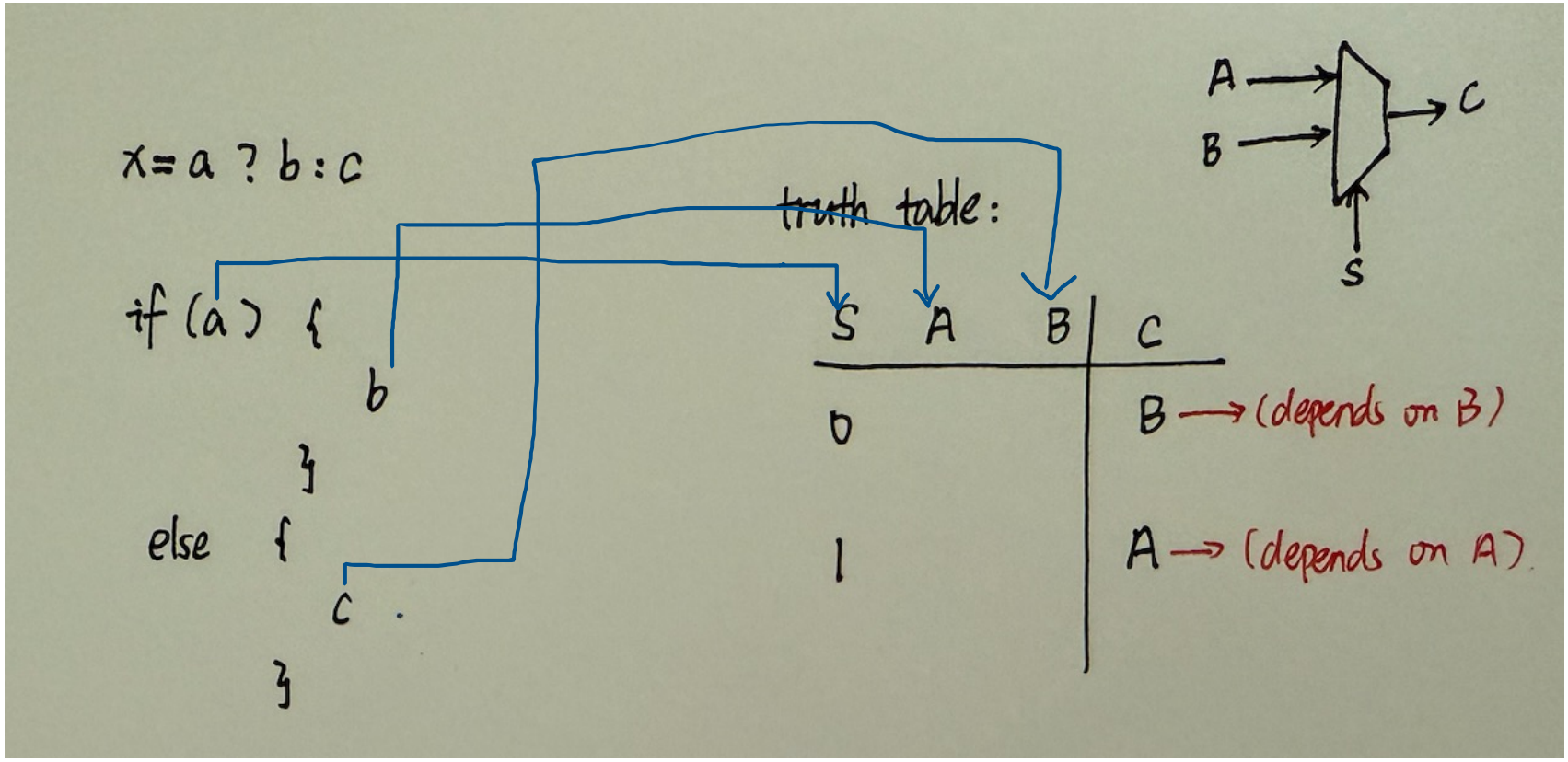
- Homework 3 due tonight at 11:59pm on Gradescope
- Midterm 1 Friday (October 3, 2025) in class
  - Written, closed notes
  - If you have SDAC, please schedule ASAP
- No Quiz this Friday!

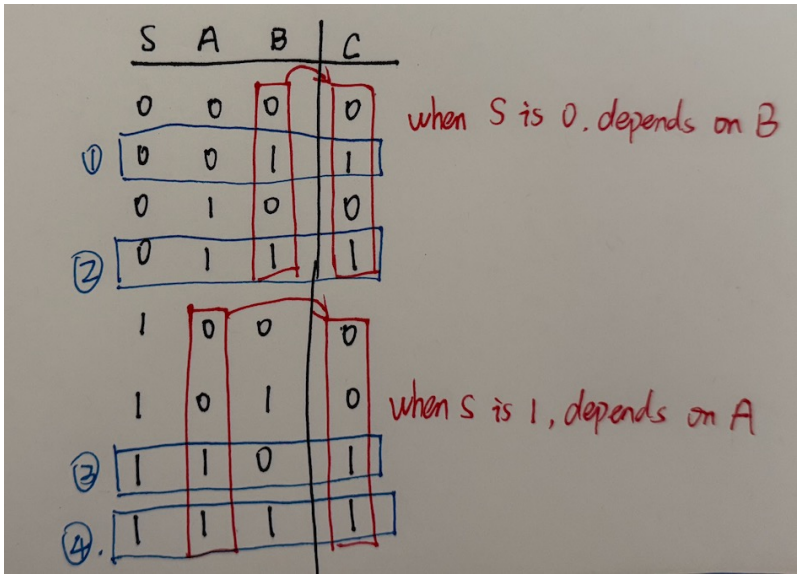
Multiplexer (mux)

x=a ? b : c

A multiplexer (mux) is commonly drawn as a trapezoid in circuit diagrams.




x = a ? b : c
if (a) {
b
}
else {
c
}
truth table:
S A B | C
0 B → (depends on B)
1 A → (depends on A)
A
B
C
S



True:

① : !S & !A & B

② : !S & A & B

③ : S & A & !B

④ : S & A & B

⇒

Combine all of them:

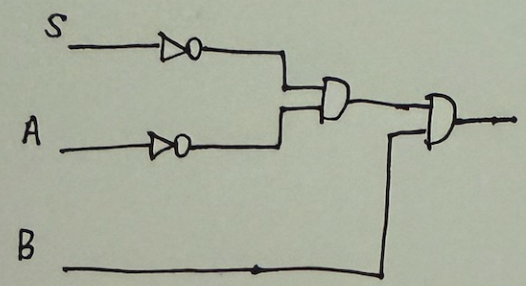
(!S&!A&B) | (!S&A&B) |

(S&A&!B) | (S&A&B)

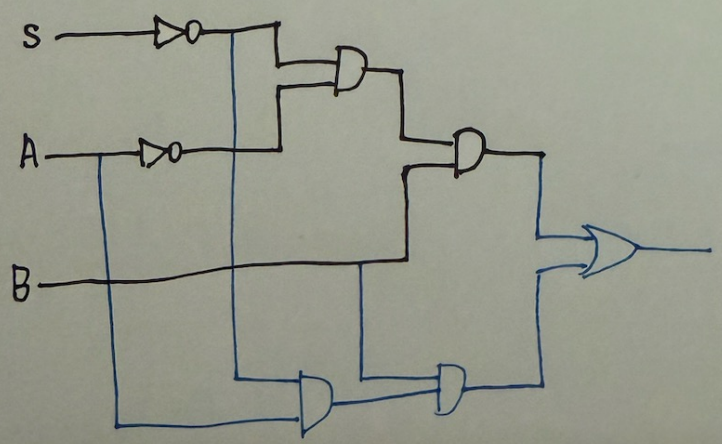
Can we simplify this expression?

Yes! But not now! Later!

① !S &!A&B

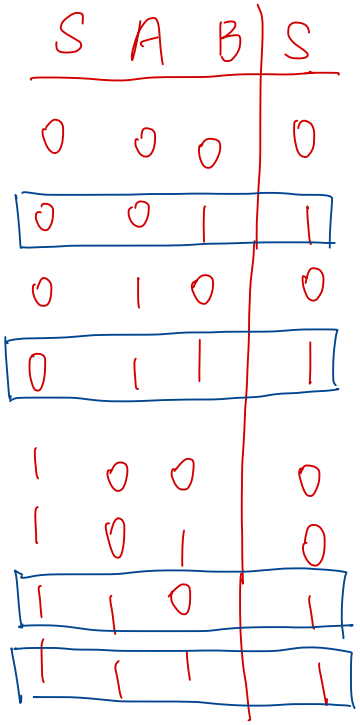


② (!S &!A&B) | (!S &A&B)



③ add other things step by step .....

Multiplexer (mux)



True:

1. !S & !A & B
2. !S & A & B
3. S & A & !B
4. S & A & B

Combine:

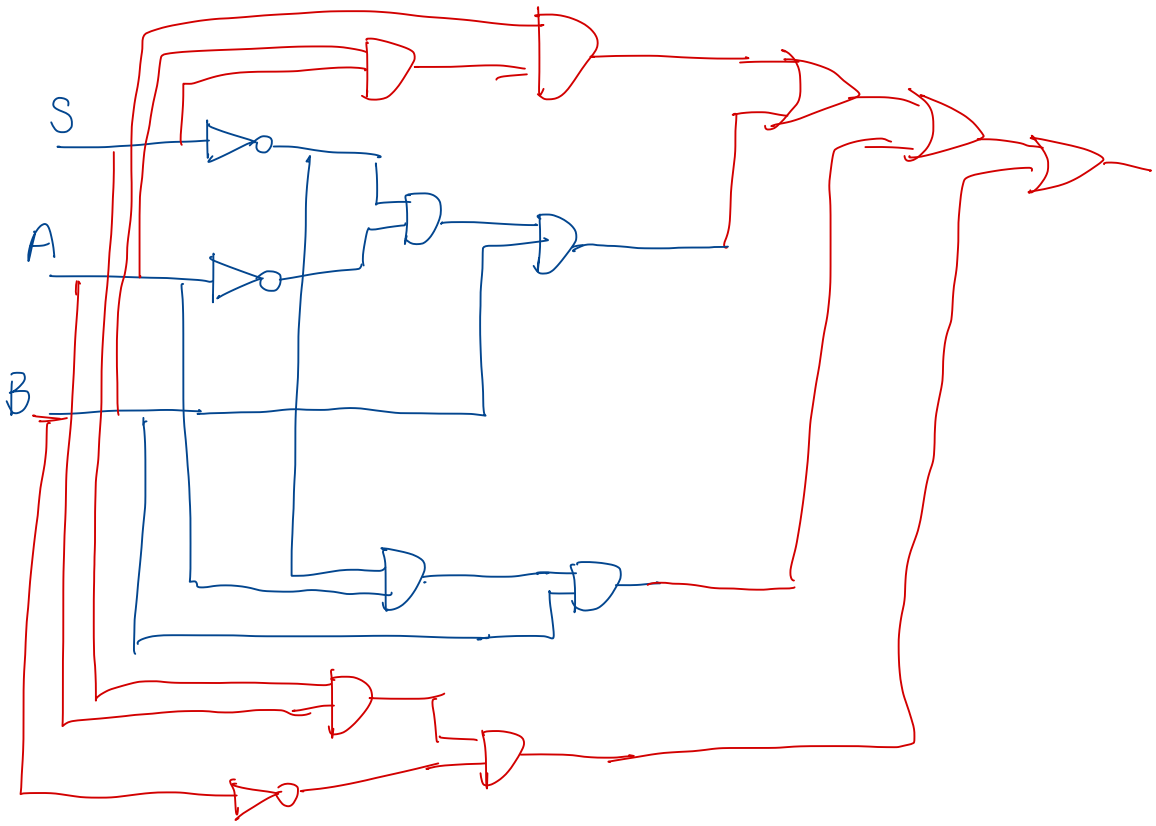
(!S & !A & B) | (!S & A & B) |

(S & A & !B) | (S & A & B)

Multiplexer (mux)

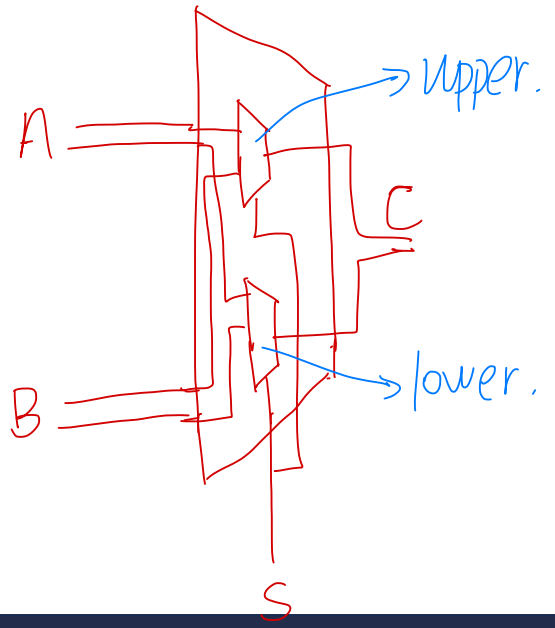
(!S& !A &B) | (!S&A&B) |

(S& A& !B) | (S&A&B)



2-bit Multiplexer (mux)

2-bit values instead of 1-bit values



In parallel.

Binary

Any downsides to binary?

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	0	0	1	0	1	0	0	1	0

Turn 2130_{10} into base-2:

2130 − 2048 = 82 82 − 64 = 18 18 − 16 = 2

hint: find largest power of 2 and subtract

2 exercises:

1. hexdecimal to binary:

5b42 ⇒

5	b	4	2
0101	1011	0100	0010

$5\times16^3+11\times16^2+4\times16+2\times1$

$=5\times4096+11\times256+4\times16+2\times1$

$=20480+2816+64+2$

$=23362$.

2. binary to hexdecimal

1101 1111 0010

0001	1010	1111	0010
1	A	F	2

Binary Addition

01101011 + 01100101

```
 01101011
+01100101
----------
 11010000
```

Range for an 8-bit number:

0000 0000 — 1111 1111

0 255

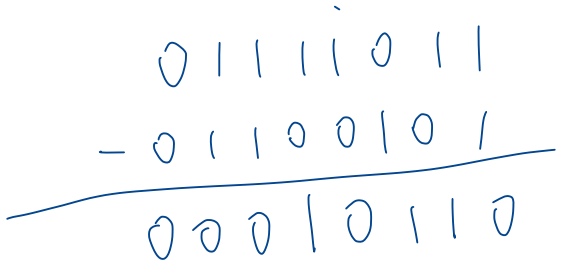
0 2^8-1

11101011 + 11100101

```
 11101011
+11100101
----------
111010000   <- overflow!
```

Binary Subtraction

01111011 - 01100101



Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

What is its value in decimal?

1. Flip all bits
2. Add 1

① Flip all bits:

1 1 0 1 0 0 1 1 → 0 0 1 0 1 1 0 0

② Add 1: 0 0 1 0 1 1 0 0 + 1 = 0 0 1 0 1 1 0 1

③ what is 0 0 1 0 1 1 0 1 in decimal ?

32 16 8 4 2 1

0 0 1 0 1 1 0 1 = 32+8+4+1 = 45

④. So the value of this negtive number is −45.

Values of Two's Complement Numbers

Consider the following decimal number:

-117

What is its value in 8-bit binary binary?

① positive 117 in binary: 01110101

② invert the bits: 10001010

③ +1 : 10001010 +1 = 10001011

Operations (on Integers)

Bit vector: fixed-length sequence of bits (ex: bits in an integer)

- Manipulated by bitwise operations

Bitwise operations: operate over the bits in a bit vector

- Bitwise not: ~x - flips all bits (unary)
- Bitwise and: x & y - set bit to 1 if *x*, *y* have 1 in same bit
- Bitwise or: x | y - set bit to 1 if either *x* or *y* have 1
- Bitwise xor: x ^ y - set bit to 1 if *x*, *y* bit differs

Your Turn!

What is:

0x1a ^ 0x72

① to binary:

1 a 7 2

0001 1010 0111 0010

0 0 0 1 1 0 1 0

^ 0 1 1 1 0 0 1 0

0 1 1 0 1 0 0 0

b 8 ⇒ 0x68

Floating Point Example

1 bit : sign
4 bits : exponent
3 bits: fraction

101.011_2 Scientific 1.01011×2^{2}

round to "011"

① 2's complement for 2:
0010

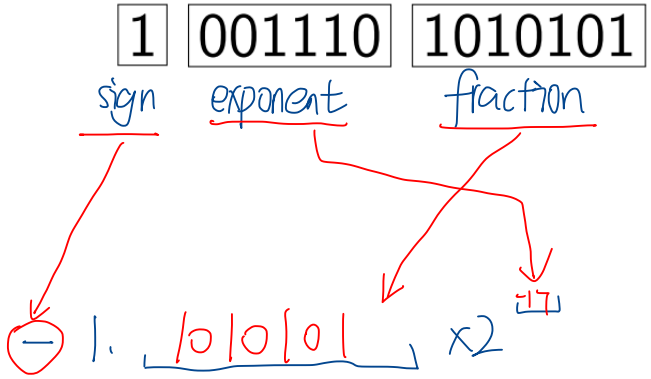
② add the bias to get the biased integer:

```
 0010
+0111
 1001
```

0	1001	011
Sign	exponent	fraction

Floating Point Example

What does the following encode?



Calculate the exponent:

```
  001110
- 011111
--------
  101111   → -17
```

flip: 010000

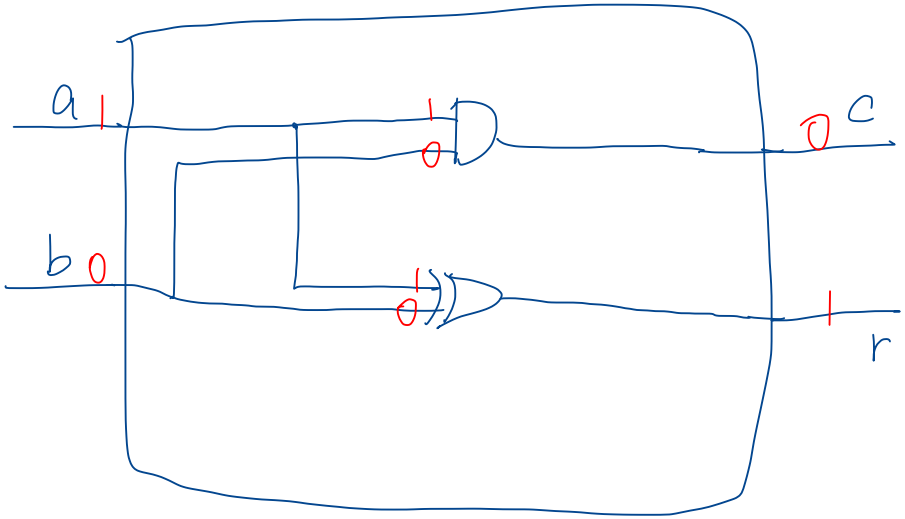
+1: 010001 → 17

Adder

Add 2 1-bit numbers: *a*, *b*

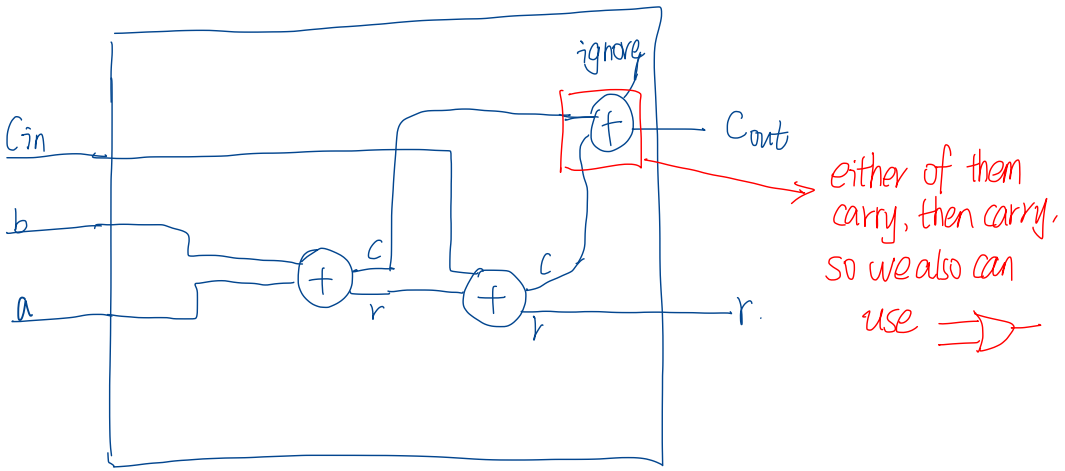
```
  a
+ b
-----
 c r
```

a	b	c	r
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

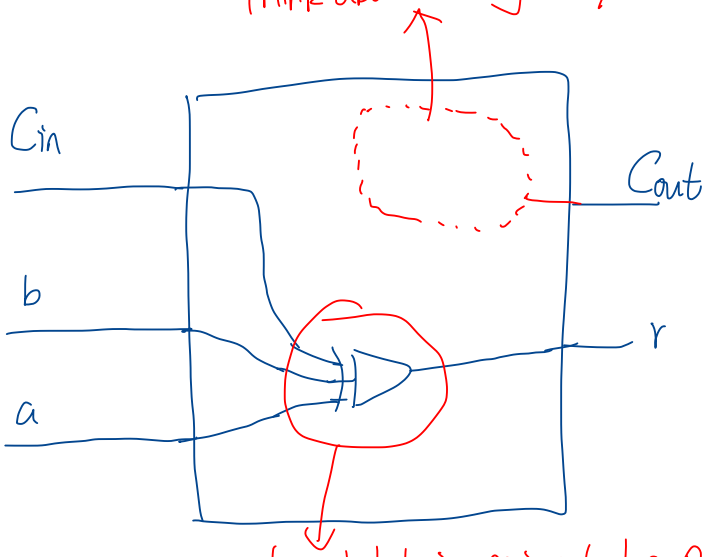


3-input Adder

Add 3 1-bit numbers: a, b, c



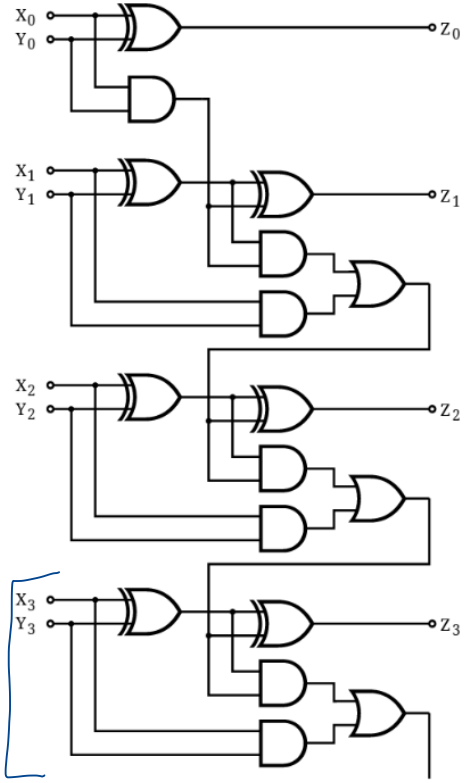
for this part, you can think:
1 if and only if ≥2.
Think about using and/or ?



2 ones ⇒ even ⇒ lowest bit is going to be 0
1 one and 2 zeros ⇒ lowest bit: 1

Ripple-Carry Adder

$X_3\ X_2\ X_1\ X_0$
$+\ Y_3\ Y_2\ Y_1\ Y_0$
$Z_3\ Z_2\ Z_1\ Z_0$



repeat if more bits

verify:

unsigned?

```
  1 1 1 1   (15)
+ 1 1 1 1   (15)
-------------
1 1 1 1 0   (14)
```

drop ← (1)

overflow!

signed?

```
  1 1 1 1   (-1)
+ 1 1 1 1   (-1)
-------------
1 1 1 1 0   (-2)
```

works!

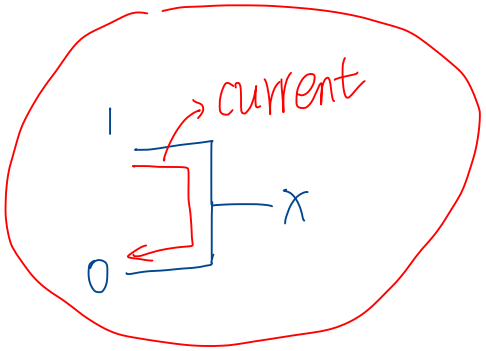
Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y
- What about the following?

  x = 1

  x = 0



doesn't work!

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output → bits are equal iff the XOR is 0
- != - same as == without not of output
- < - consider x < 0
- >, <=, => are similar

Compute d = x ^ y. if all bits of d are 0 ⇒ equal

x < y : x - y < 0

→ (x >> 31) & 1

if the result is 1, then negative.

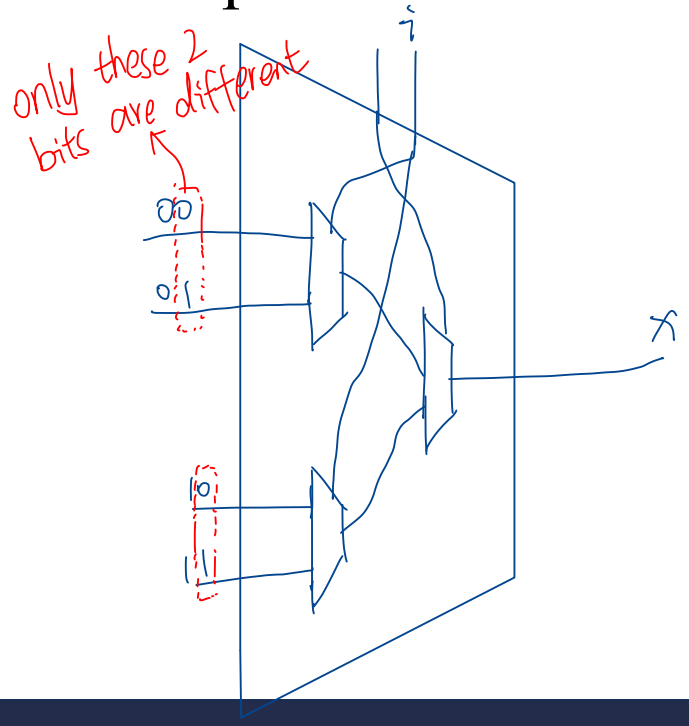
Indexing

Indexing with square brackets: [ ]

- **Register bank** (or **register file**) - an array of registers
  - Can programmatically pick one based on index
  - I.e., can determine which register while running
- Two important operations:
  - x = R[i] - Read from a register
  - R[j] = y - Write to a register

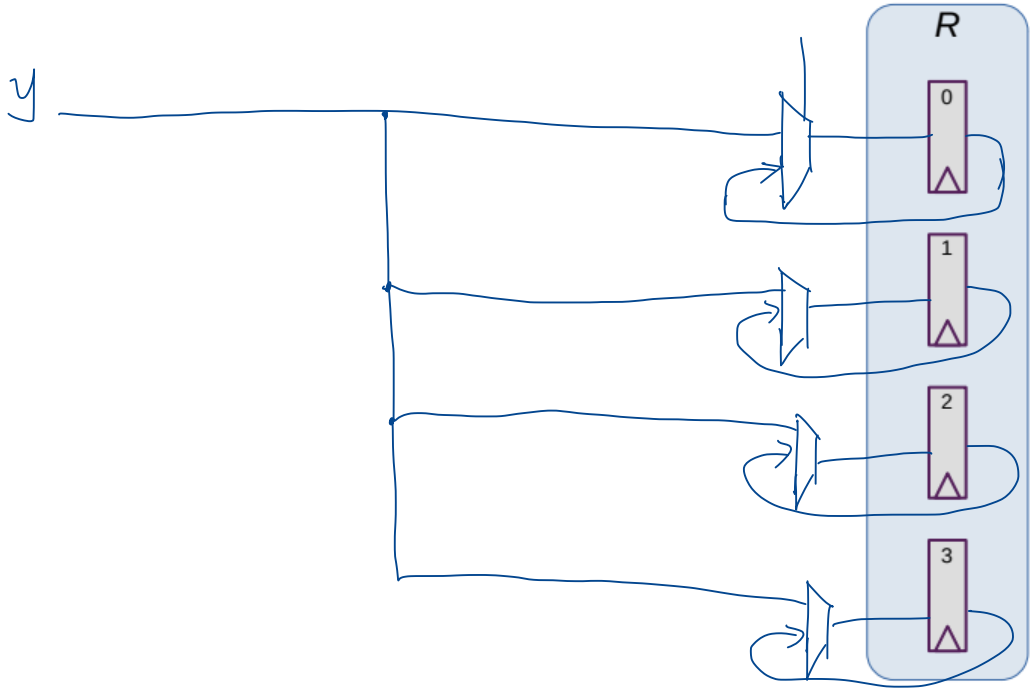
Aside: 4-input Mux

How do we build a 4-input mux? How many wires should *i* be?



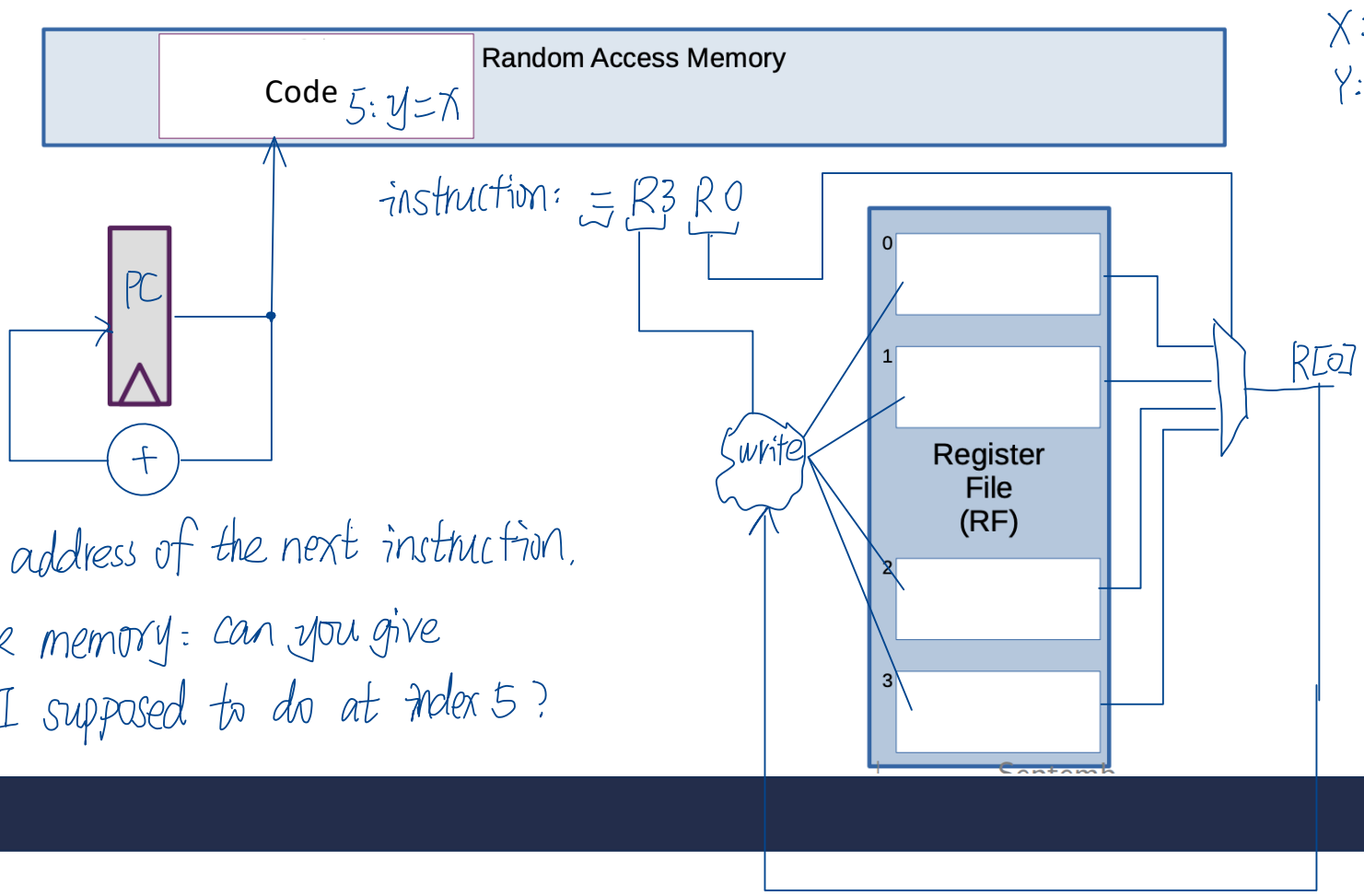
Writing

R[j] = y - connect *y* to input of registers based on index *j*



Every clock cycle, I need to keep my current value, I don't want to loose it.

Building a Computer



PC has the address of the next instruction.
we can ask memory: can you give me what I supposed to do at index 5?

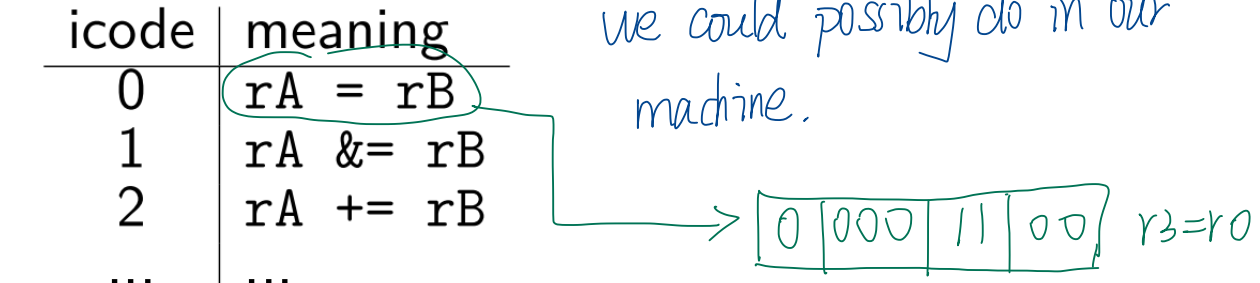
Encoding Instructions

Encoding of Instructions (**icode** or **opcode**)

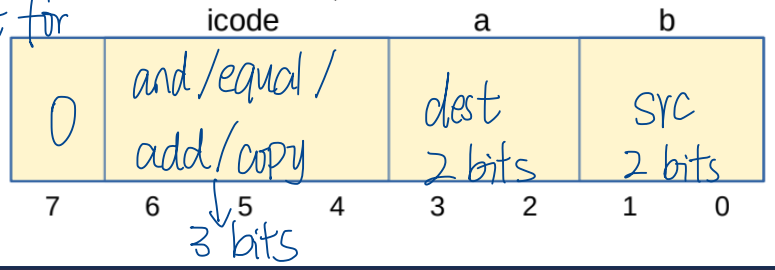
- Numeric mapping from icode to operation

icode	meaning
0	`rA = rB`
1	`rA &= rB`
2	`rA += rB`
...	...

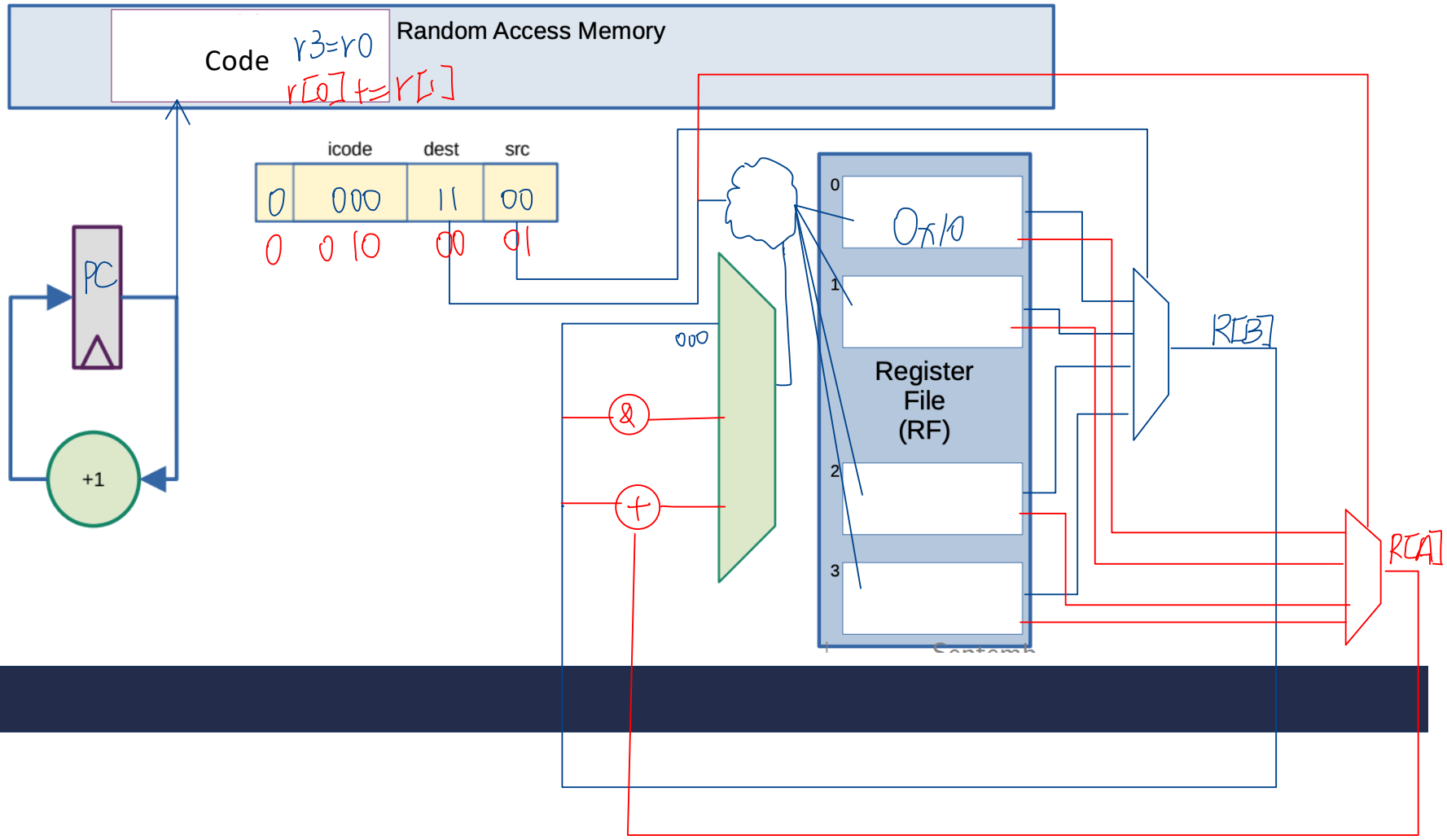
list out all the things that we could possibly do in our machine.



keep this bit for future newer things



Building a Computer



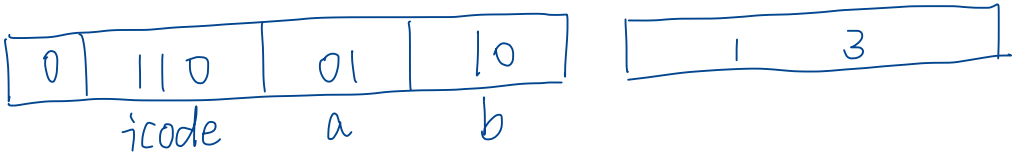
Instructions

icode	b	meaning
0		`rA = rB`
1		`rA &= rB`
2		`rA += rB` 2 registers
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` immediate value.
	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

Encoding Instructions

Example 1: r1 += 19 → in decimal

19 in hexdecimal: 0x13



hex: 0b13

Encoding Instructions

idea: ①. I have a value in memory at address hex 82

Example 2: M[0x82] += r3 ②. I want to add whatever in R3 to that value.

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.

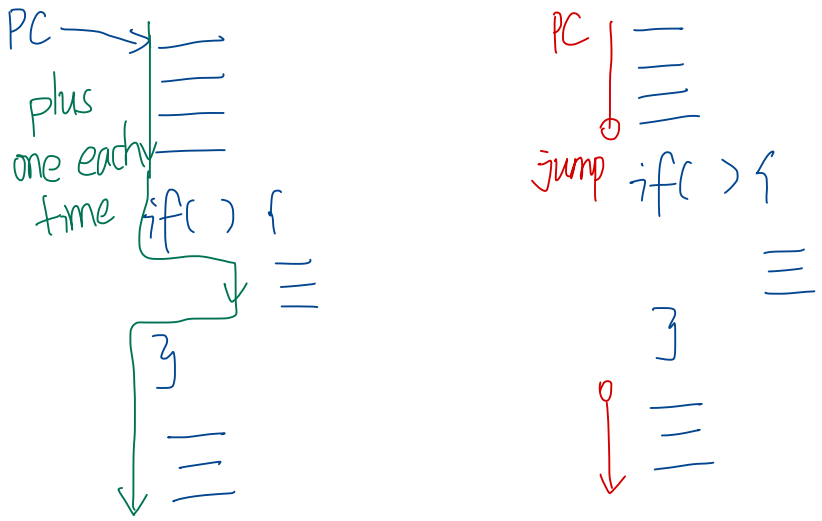
r2 = 0x82	0 110 | 10 00 (6 | 8)	82
r1 = M[r2]	0 100 | 01 10 (4 | 6)	
r1 += r3	0 010 | 01 11 (2 | 7)	
M[r2] = r1	0 101 | 01 10 (5 | 6)	

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

One interesting finding: first hex is always our icode!

Jumps

- For example, consider an if



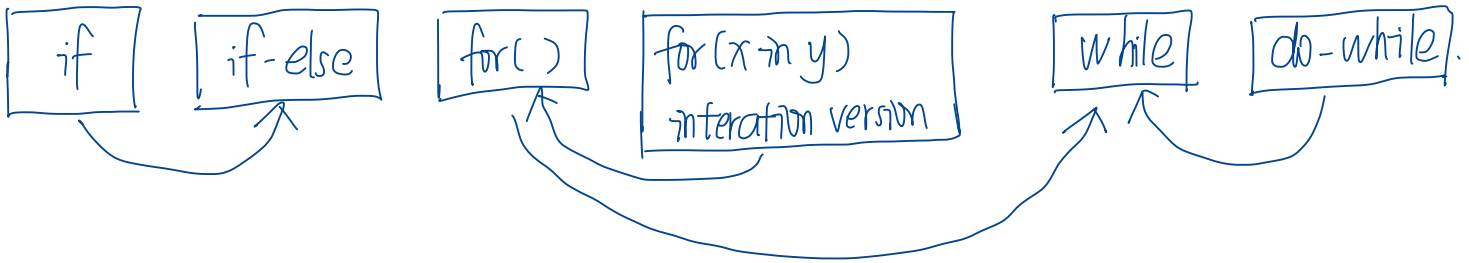
when we got "if", we have a choice

{ 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)

Our code to this machine code

How do we turn our control constructs into jump statements?



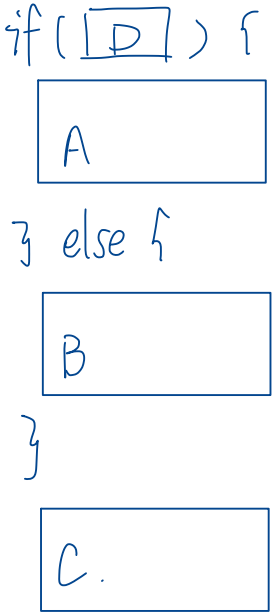
how to convert for to while:

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

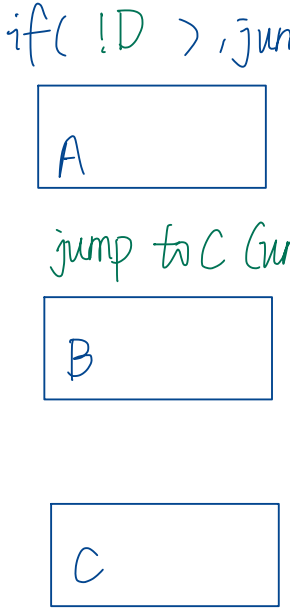
⇒

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

if/else to jump



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



If D is true, no need to jump, continue to A.
So we can think about it from an opposite way, if(!D), jump to B.

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

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! Infinate 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.

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 ⇔ r0<=8 ⇔ (r0-8)<=0
⇔ r0+=-8 (0xF8)
r0<=0

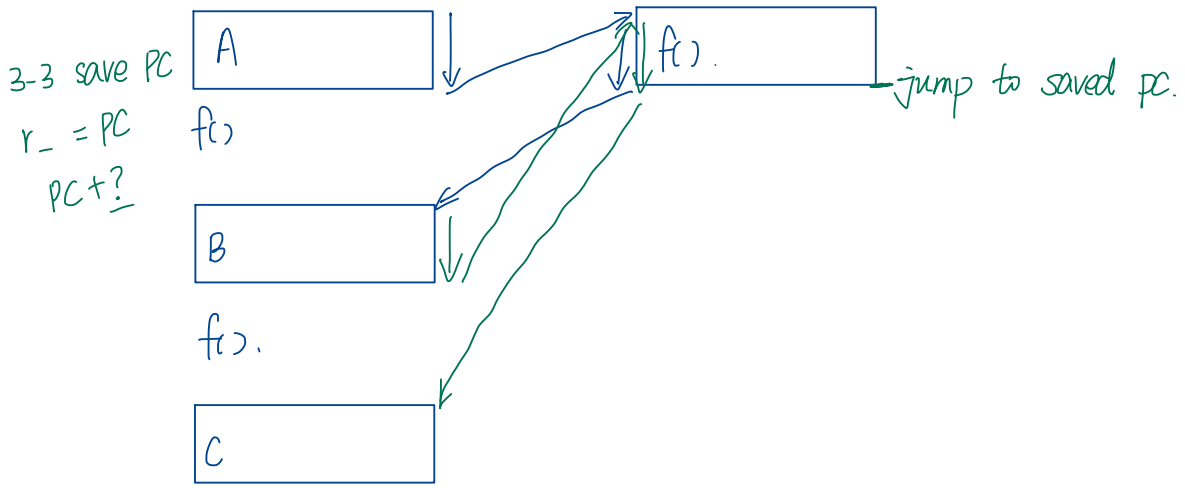
r1 = 0x42 — 0 110 01 00 — 42
6 4 42

r0 += F8 — 0 110 00 10 — F8
6 2 F8

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

b442b2F871

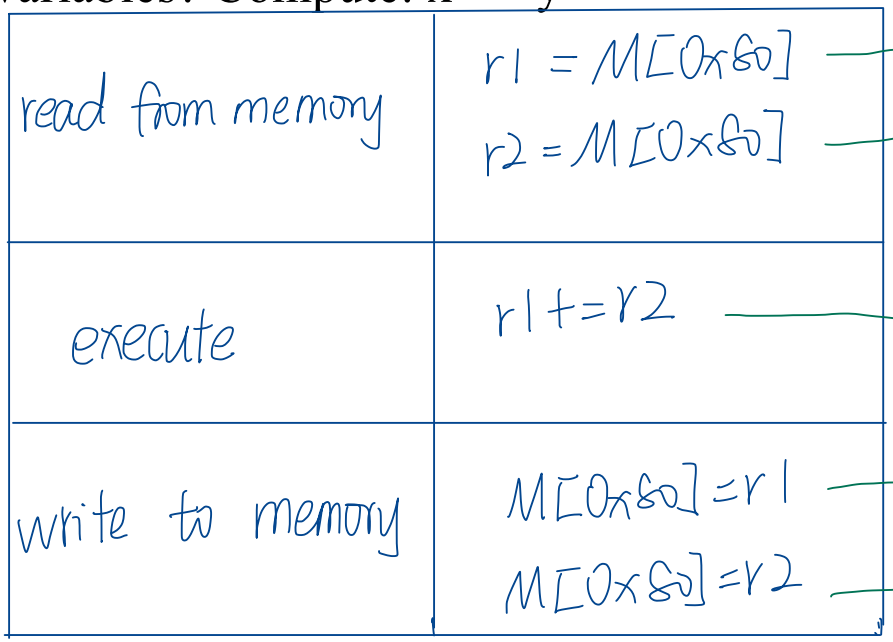
Function Calls



Dealing with Variables and Memory

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

x = 0x80
y = 0x81



0111 → 67 80
6B 81 ← 1011
2b ← 0110
r0 = 0x80 → 60 80
M[r0] = r1 → 54
r0 = 0x81 → 60 81
M[r0] = r2 → 58

67 80 6B 81 2b 60 80 54 60 81 58

Instructions Set Architecture

Instruction Set Architecture (ISA) is an abstract model of a computer defining how the CPU is controlled by software

- Provides an abstraction layer between:
  - Everything computer is really doing (hardware)
  - What programmer using the computer needs to know (software)

CSO: covering many of the times we'll need to think across this barrier

We're in general at this point, going to start staying just above this barrier and to the software side.