COMPUTER SYSTEMS AND ORGANIZATION

Part 1

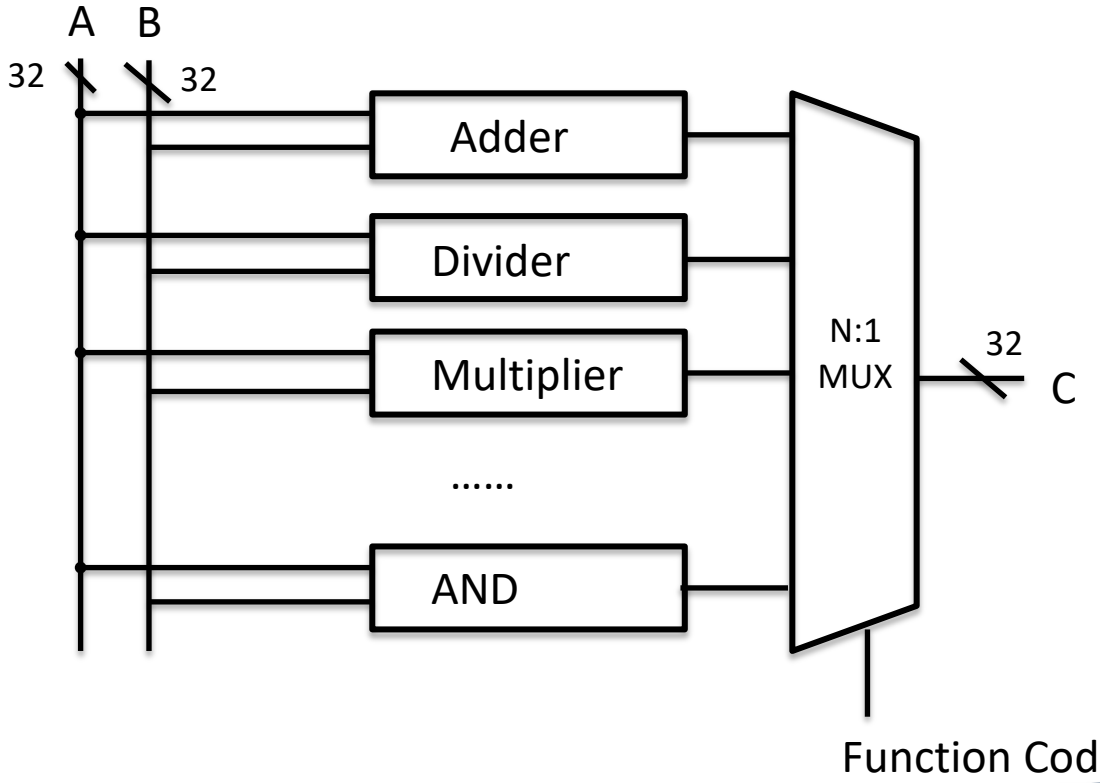
Instruction Set Architecture

Daniel G. Graham PhD

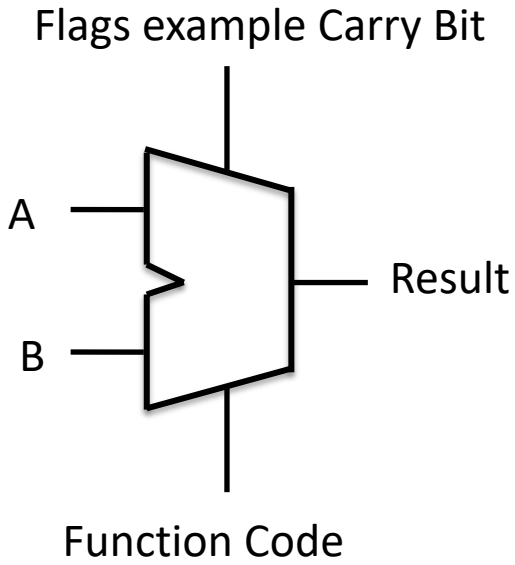
September 11, 2023

REVIEW

ARITHMETIC LOGIC UNIT



ALU SYMBOL AND INPUTS



TINY PROGRAM TO ASSEMBLY

```
m = 4
x = 2
b = -1
y = m*x*b
```

Looks like we need two types on instructions

1. An instruction to load values
2. An instruction to computation (multiply)

NOW LET'S TRANSLATE OUT PROGRAM TO ONES AND ZERO

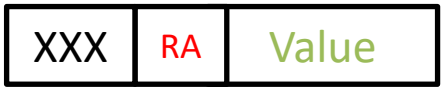
1. An instruction to load values into **Registers**

m = 4 → R0 = 3 → 000 | 00 | 011 → 0x03

x = 2 → R1 = 2 → 000 | 01 | 010 → 0x0A

b = -1 → R2 = -1 → 000 | 10 | 111 → 0x17

GREAT WE HAVE OUR FIRST INSTRUCTION



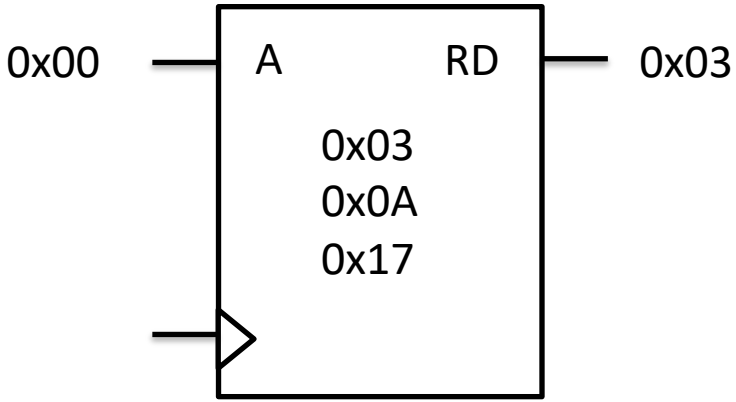
RA = Value

SO WHAT GETS LOADED INTO MEMORY

Great so we convert our program to hex and loaded it into memory

m = 3
x = 2
b = -1

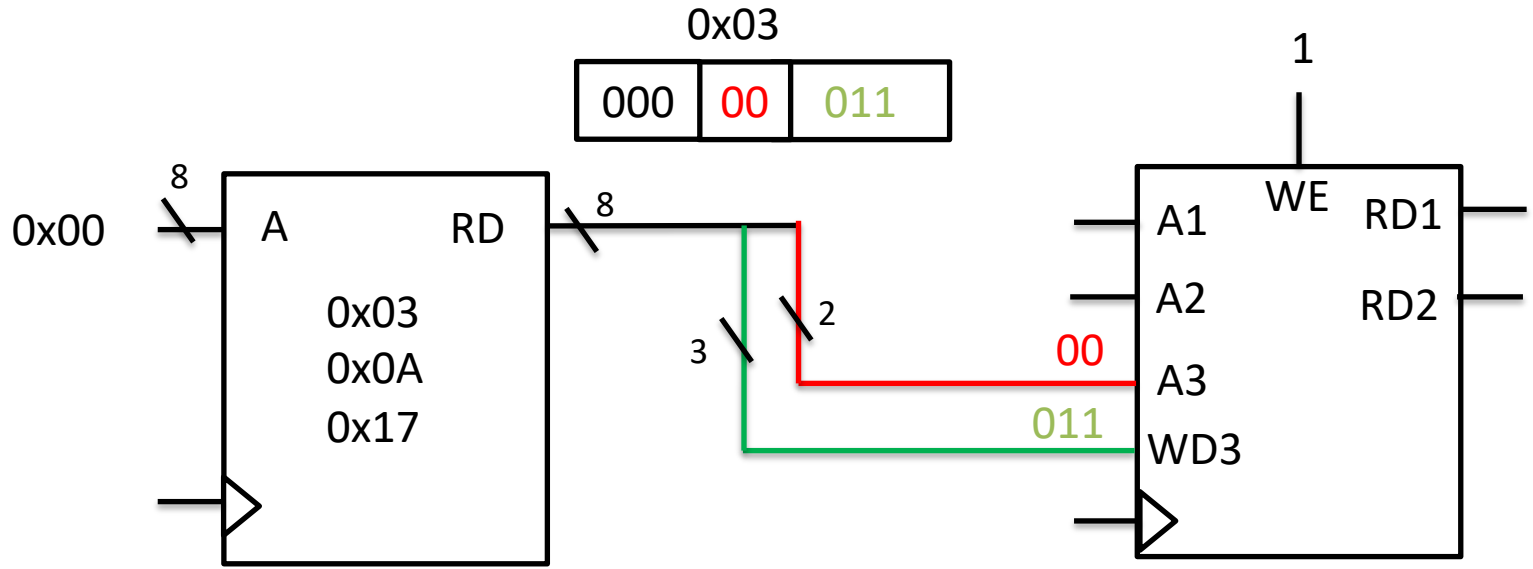
R0 =3
R1 = 2
R2 = -1



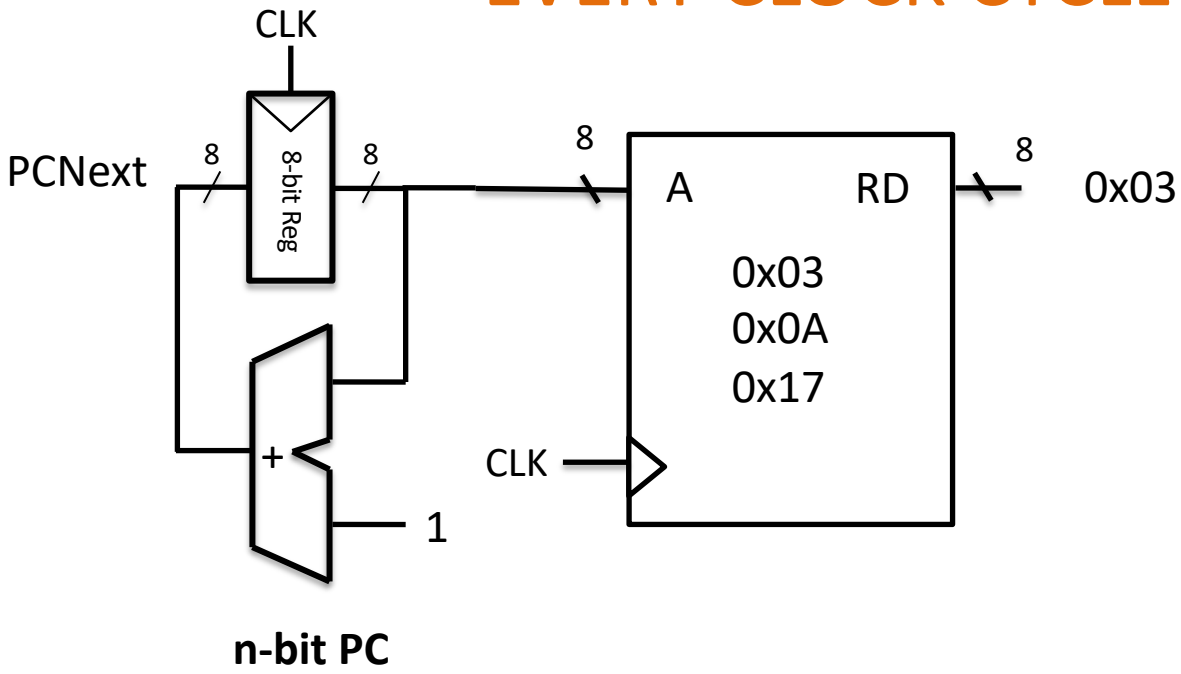
We still need to load our values into registers

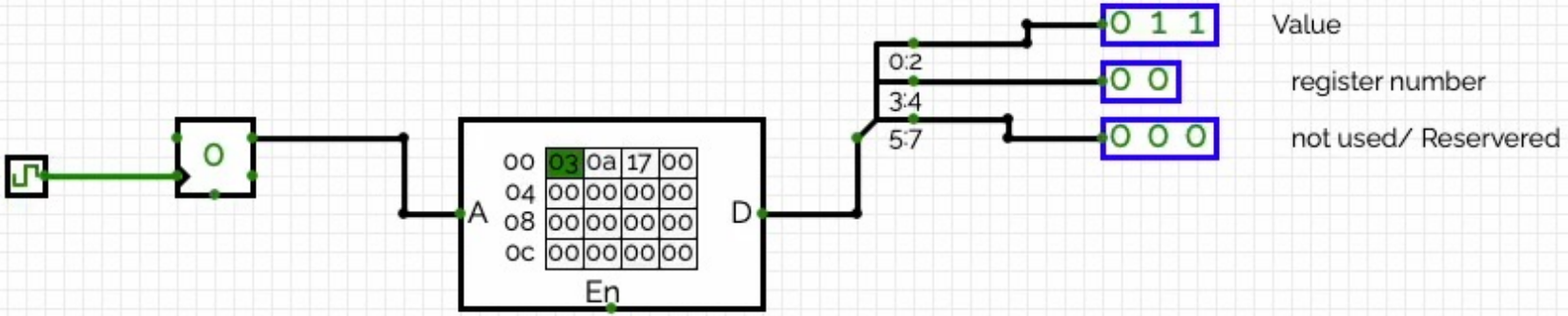
LETS ADD OUR REGISTER FILE

R0 =3
R1 = 2
R2 = -1

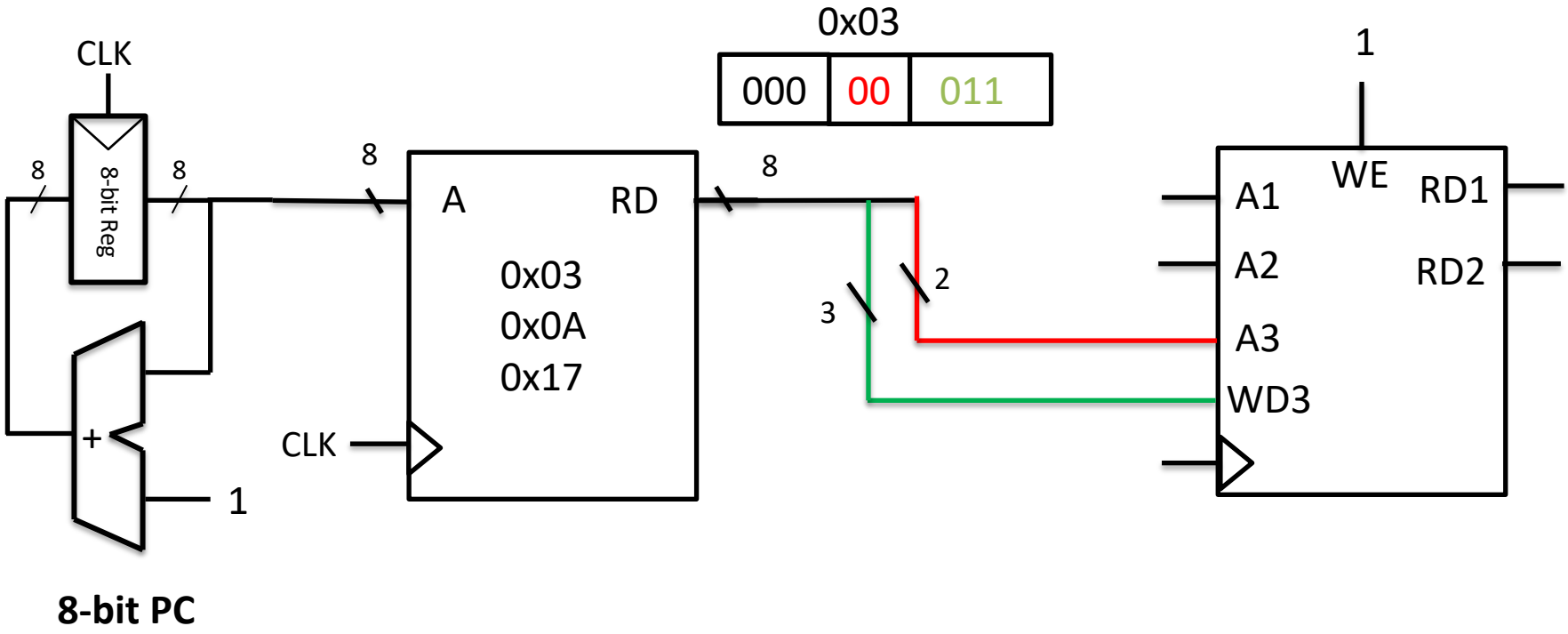


AUTOMATICALLY FETCH A NEW INSTRUCTION EVERY CLOCK CYCLE




0
A
00 03 0a 17 00
04 00 00 00 00
08 00 00 00 00
0c 00 00 00 00
En
D
0:2
3:4
5:7
0 1 1
0 0
0 0 0
Value
register number
not used/ Reserved

NOW LET'S ADD OUR REGISTER FILE



GREAT WE LOADED THE VALUES WHAT ABOUT MULTIPLICATION

An instruction to load values into **Registers**

```
m = 3
x = 2
b = -1
```

R0 = 3 (contains m)
R1 = 2 (contains x)
R2 = -1 (contains b)

But how do encode this in bits so that we can execute it.

An instruction to computation (multiply)

```
y = m*x*b
```

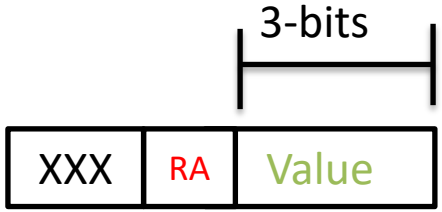
R0 *= R1 ⟵ m = m * x
R0 *= R2 ⟵ m = m * b

LET'S DECIDE HOW WE ARE GOING TO LAYOUT OUR BITS

Multiply **Registers**

```
y = m*x*b
```

```
R0 *= R1
R0 *= R2
```



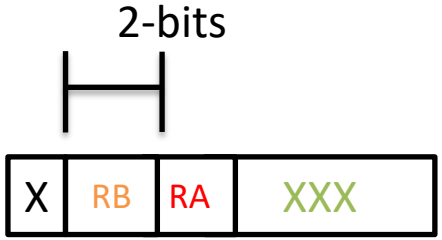
Don't real need the Value bits but we need another register so let's use the unused bits.

LET'S DECIDE HOW WE ARE GOING TO LAYOUT OUR BITS

Multiply **Registers**

```
y = m*x*b
```

```
R0 *= R1
R0 *= R2
```



Let's use some of unused bits to specify our register?

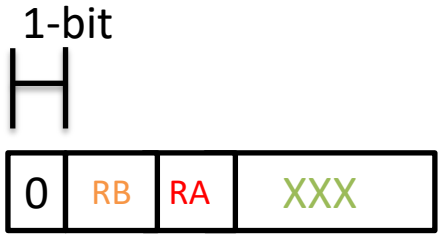
Need to be careful about which one is our destination register
Here the results get written to RA

OPCODE

Multiply **Registers**

```
y = m*x*b
```

```
R0 *= R1
R0 *= R2
```



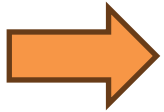
0 --> Multiply
1 --> Save Value to register

Finally, we need an opcode to distinguish our load instruction from our multiple

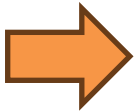
ENCODING

Let's multiply value in **Registers**

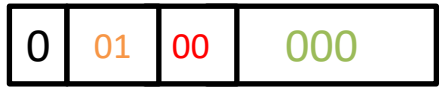
y=m*x*b



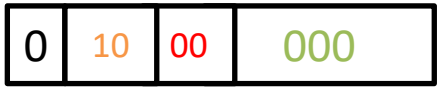
R0 *= R1

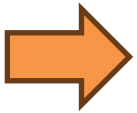


R0 *= R2



0x20

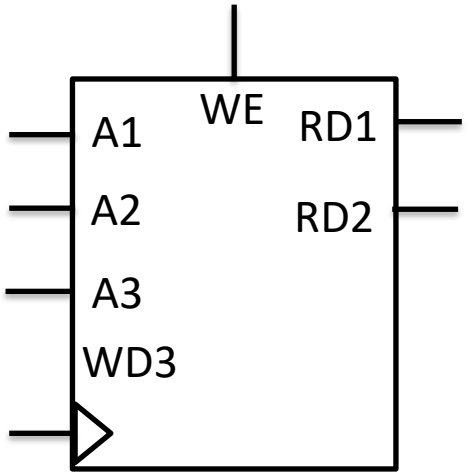


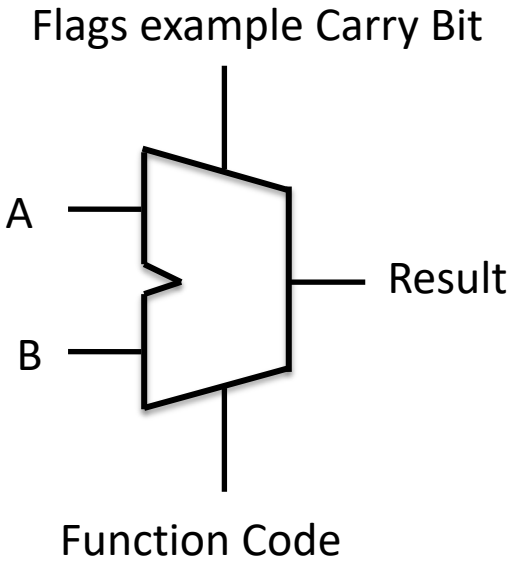


0x40

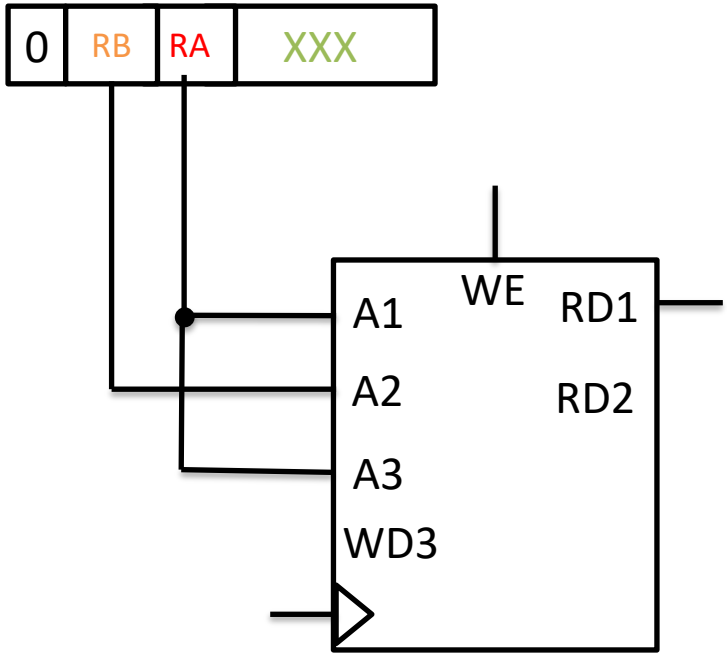
0x17

BUILDING MACHINE TO COMPUTE THIS

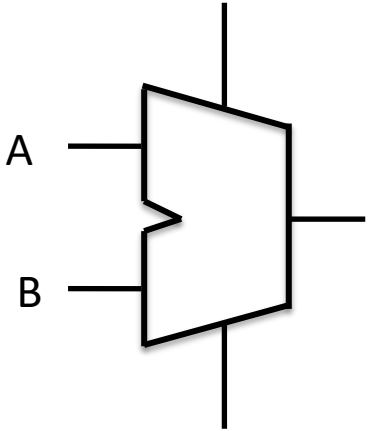




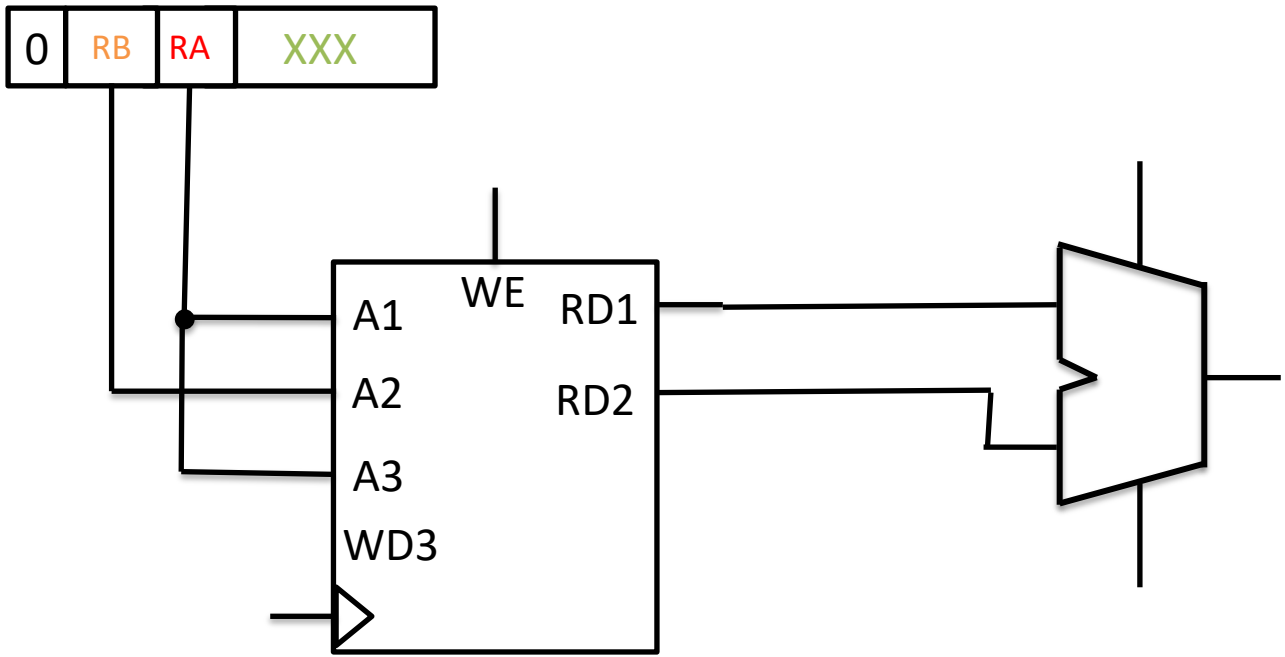
BUILDING MACHINE TO COMPUTE THIS



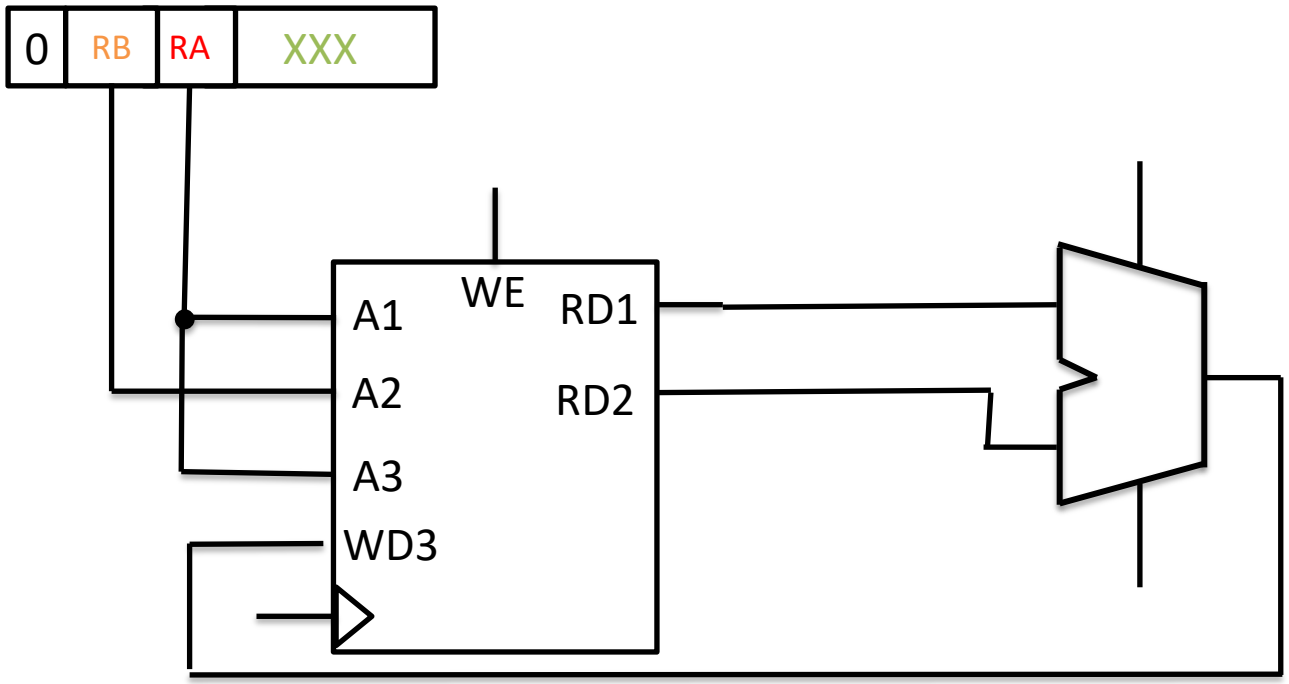
Flags example Carry Bit



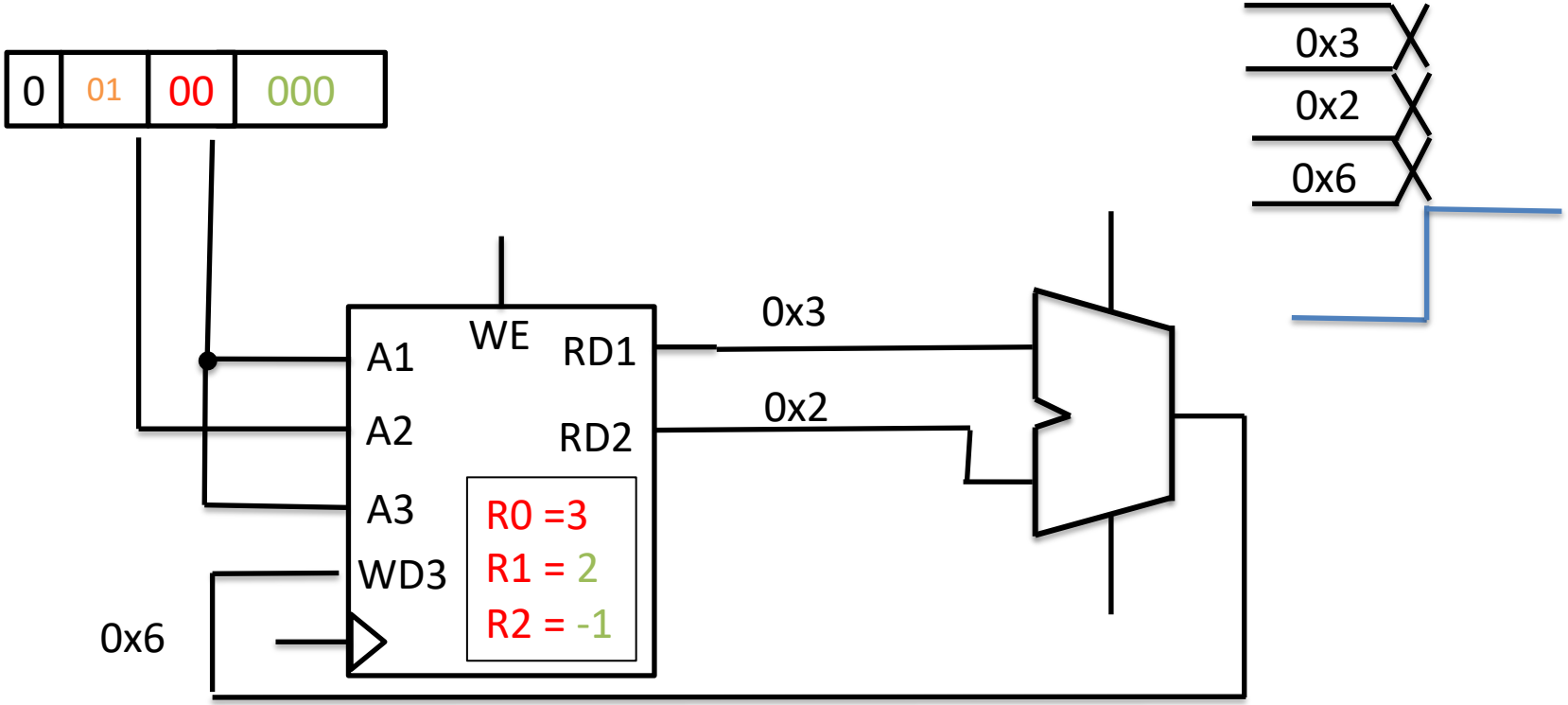
BUILDING MACHINE TO COMPUTE THIS



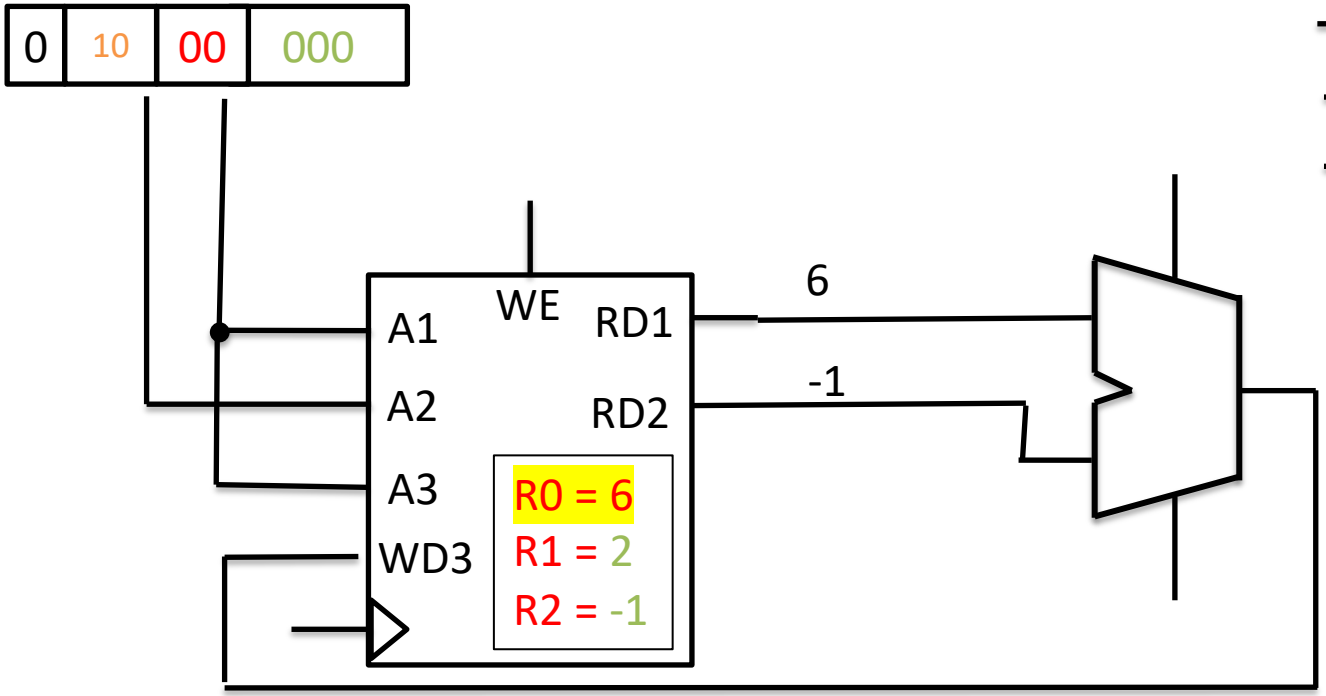
BUILDING MACHINE TO COMPUTE THIS

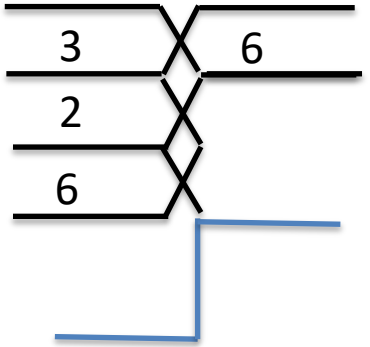


BUILDING MACHINE TO COMPUTE THIS



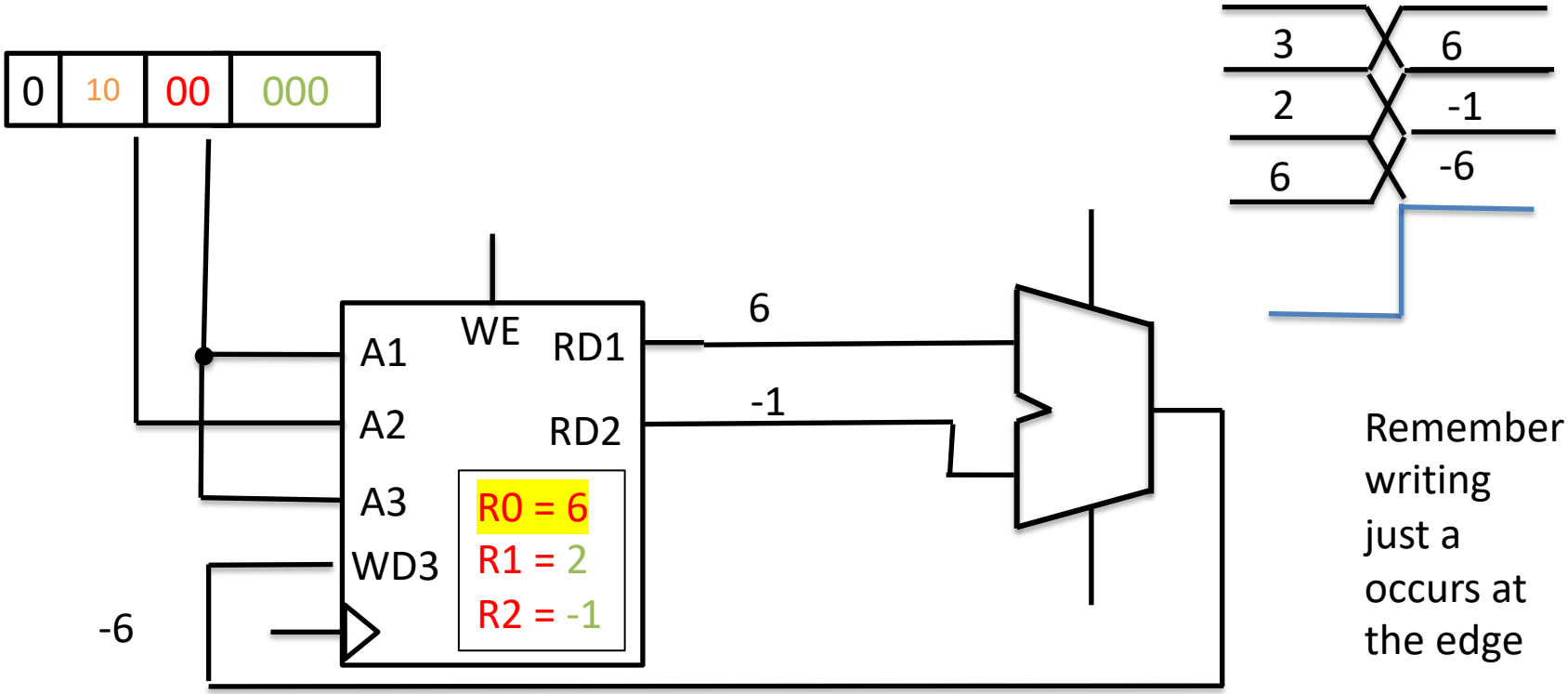
BUILDING MACHINE TO COMPUTE THIS





Remember writing just a occurs at the edge

BUILDING MACHINE TO COMPUTE THIS



Remember writing just a occurs at the edge

NOTE WE ALSO NEED TO UPDATE THE ENCODING OF OUR LOADS

1. An instruction to load values into **Registers**

m = 4 → R0 = 3 → 1 | 00 | 00 | 011 → 0x83

x = 2 → R1 = 2 → 1 | 00 | 01 | 010 → 0x8A

b = -1 → R2 = -1 → 1 | 00 | 10 | 111 → 0x97

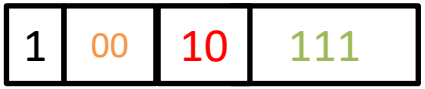
1. An instruction to load values into **Registers**

m = 4 → R0 = 3 → 

0x83

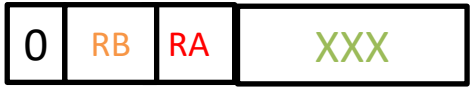
x = 2 → R1 = 2 → 

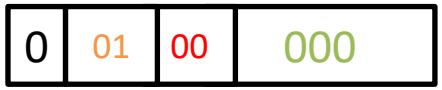
0x8A

b = -1 → R2 = -1 → 

0x97

Let's multiply value in **Registers**



y=m*x*b → R0 *= R1 → 

0x20

R0 *= R2 → 

0x40

0x17

INSTEAD GOING INSTRUCTION BY INSTRUCTION

LET'S DESIGN THE ISA AND THE MACHINE

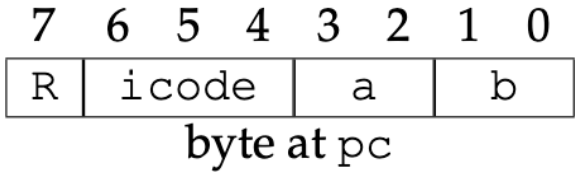
TODAY'S LECTURE

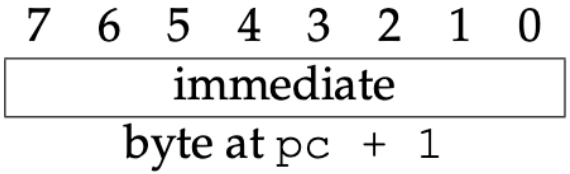
- Look at and Toy ISA that we designed
- Get comfortable encoding instructions in our Toy ISA
- Write small programs, encode them
- Run these programs in our simulator

TOY INSTRUCTION SET ARCHITECTURE (ISA)

The ISA defines:

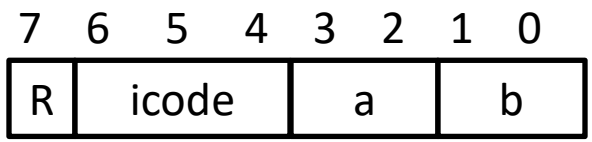
1. Instructions and their layout
2. Data types
3. Registers we'll have



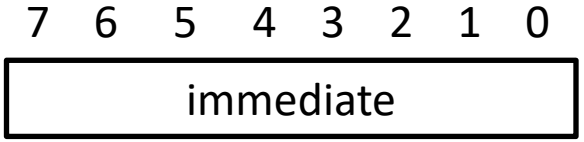


How instructions are laid out in our ISA

ENCODING OUR FIRST INSTRUCTION



byte at pc



byte at pc + 1

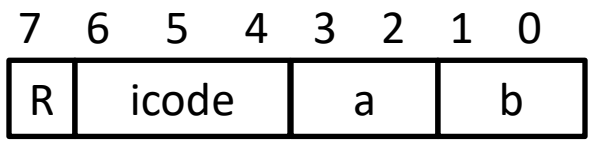
We'll assign it icode (instruction code) 0

RA = RB

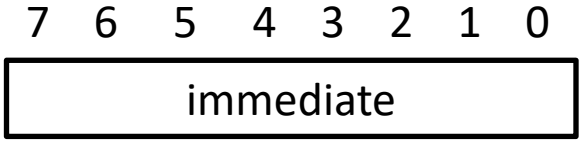
Try to encode the following instruction R0 = R1

ENCODING OUR FIRST INSTRUCTION

Try to encode the following instruction R0 = R1

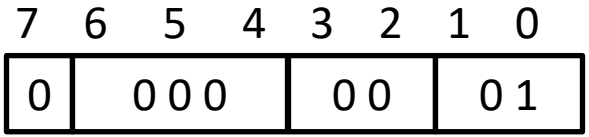


byte at pc



byte at pc + 1

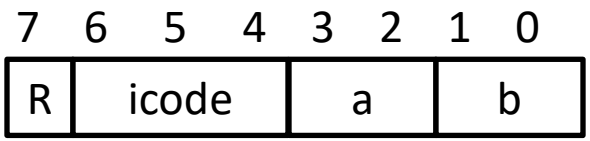
icode 0 RA = RB

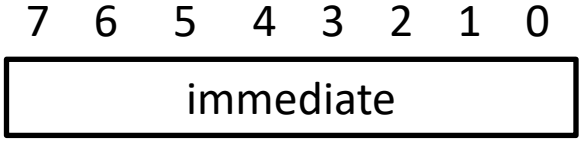


0x01

ENCODING OUR FIRST INSTRUCTION

Try to encode the following instruction R0 = R1

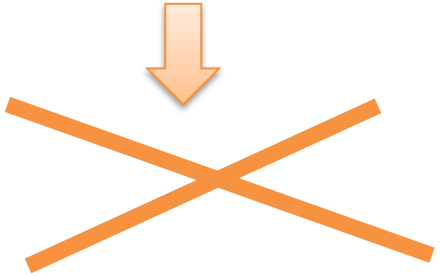




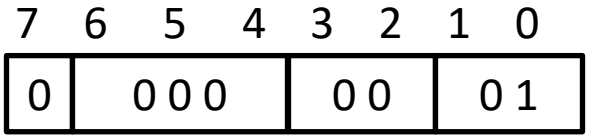
byte at pc

byte at pc + 1

icode 0 RA = RB

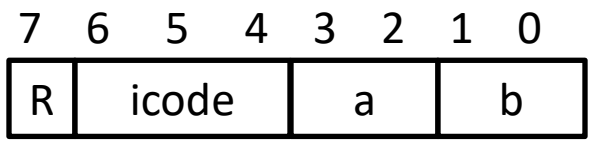


Not used
This instruction is not using a value

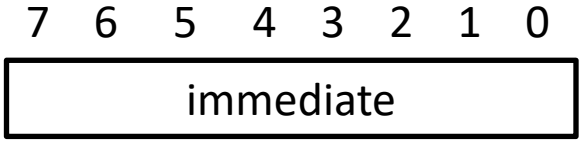


ENCODING OUR FIRST INSTRUCTION

Try to encode the following instruction R0 = R1

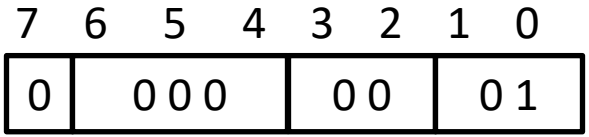


byte at pc



byte at pc + 1

icode 0 RA = RB



0x01

INSTRUCTIONS WE'LL ENCODE

icode	Behavior
0	rA=rB
1	rA+=rB
2	rA&=rB

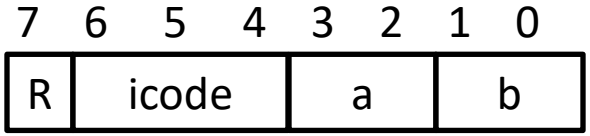
INSTRUCTIONS WE'LL ENCODE

icode	Behavior
0	rA=rB
1	rA+=rB

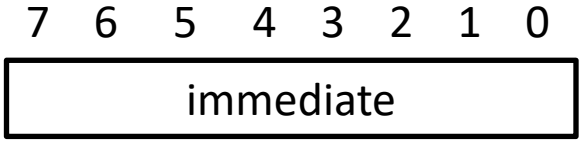
Let's do icode 1 next

icode	Behavior
1	rA+=rB

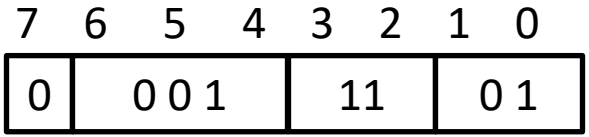
Let's encode R3 += R1 (Remember to pay attention to the destination)



byte at pc



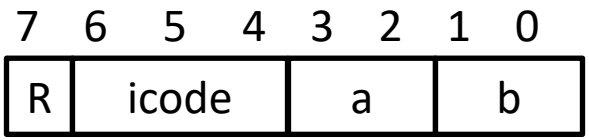
byte at pc + 1

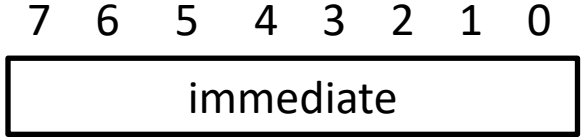


ACTIVITY

Write the following instruction r2 &= r3 in hex

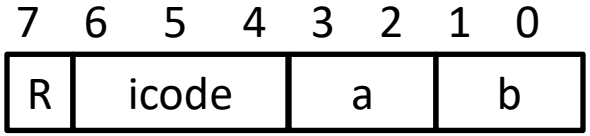
icode	Behavior
0	rA=rB
1	rA+=rB
2	rA&=rB



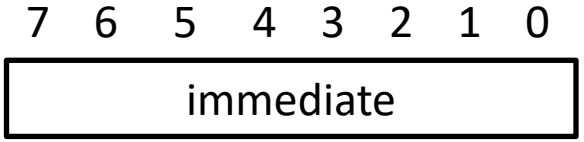


icode	Behavior
3	rA&=rB

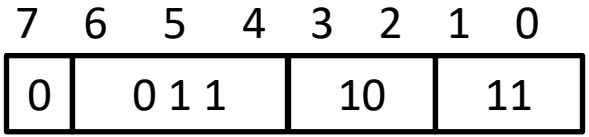
Let's encode R2 &= R3 (Remember to pay attention to the destination)

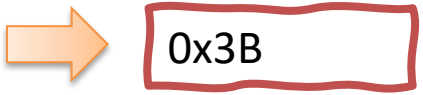


byte at pc



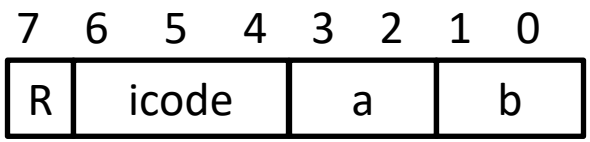
byte at pc + 1

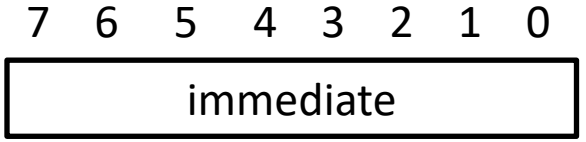




ICODE

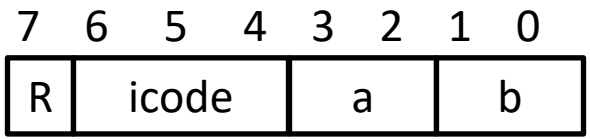
Our icode is only 3 bits. Does this mean that we can only have 2^3 instructions?
What if the instruction doesn't use **b** could repurpose it as a part of the code?
(Don't believe this best practice, but it is our toy ISA so let's have and be creative)

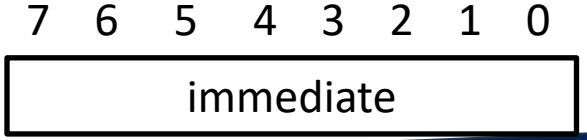




FUN WITH B

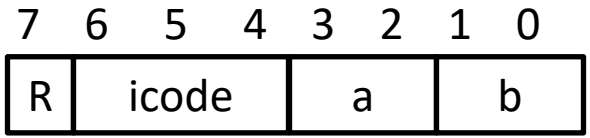
icode	b	Behavior
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]
	1	-------Coming Soon------
	2	-------Coming Soon------
	3	-------Coming Soon------

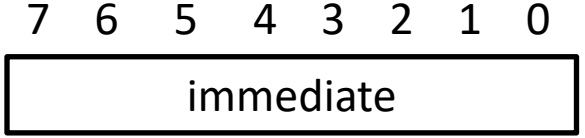




PUT IT ALL TOGETHER

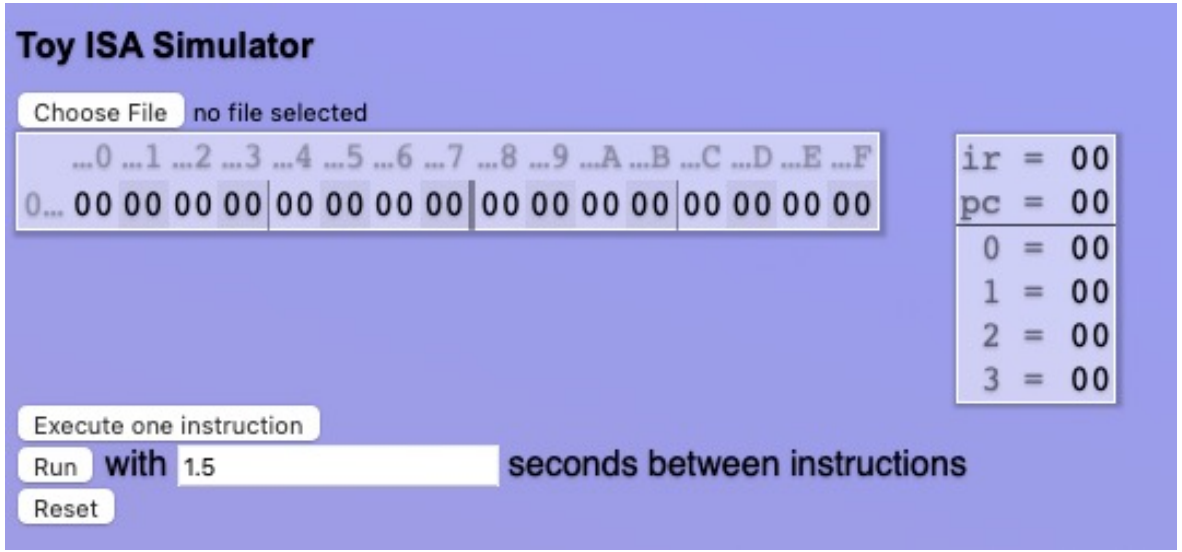
icode	b	Behavior
0		rA=rB
1		rA+=rB
2		rA&=rB
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]





CHALLENGE

Can we write a program in our Toy Machine Code, that adds two numbers? Can we run it in the online simulator?
https://researcher111.github.io/uva-cso1-F23-DG/homework/files/toy-isa-sim.html



STEP 0: WRITE PROGRAM IN PSEUDO CODE

```
x = 8
y = -1
z = x + y
```

STEP 1: REGISTER ALLOCATION AND TRANSLATION

Decide which variables will be stored in memory and which variables will be stored in registers. Choose registers and memory locations.

Rewrite the program using the instructions we have

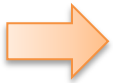
```
x = 8
y = -1
z = x + y
```

```
R0 = 8
R1 = -1
R0 += R1
```

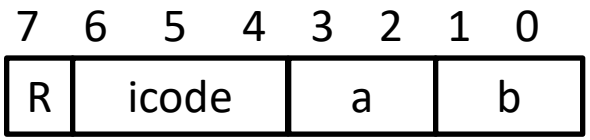
STEP 2: ENCODE INSTRUCTIONS

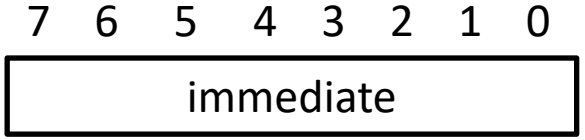
Use the ISA layout to encode the instructions

```
x = 8
y = -1
z = x + y
```



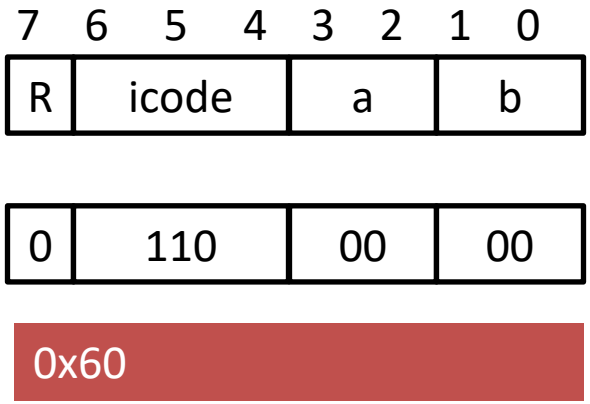
```
R0 = 8
R1 = -1
R0 += R1
```

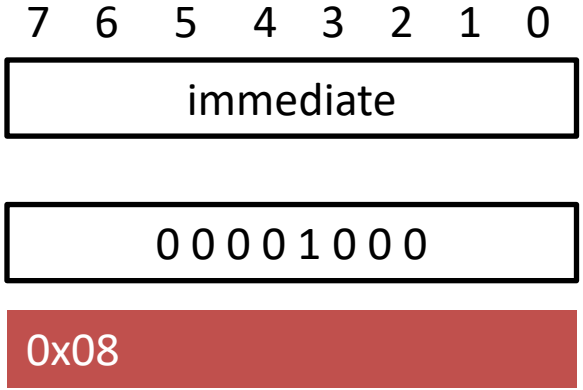




icode	b	Behavior
0		rA=rB
1		rA+=rB
2		rA&=rB
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]

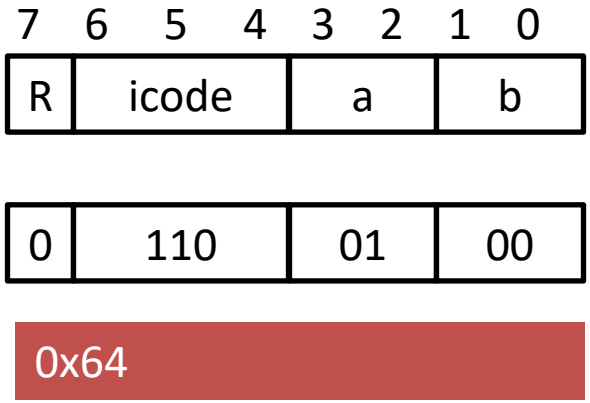
```
R0 = 8
R1 = -1
R0 += R1
```

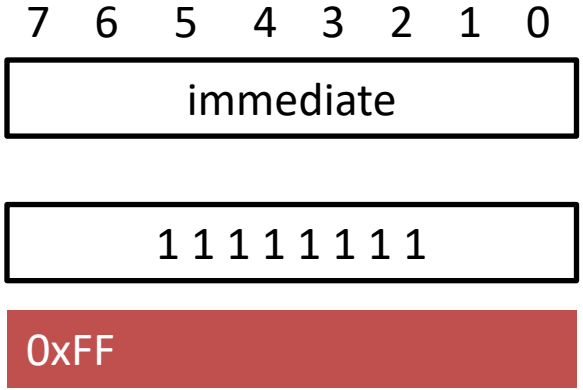




icode	b	Behavior
0		rA=rB
1		rA+=rB
2		rA&=rB
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]

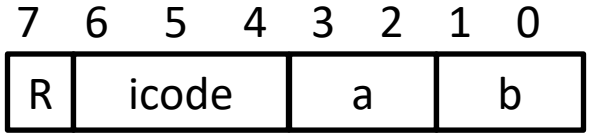
```
R0 = 8
R1 = -1
R0 += R1
```

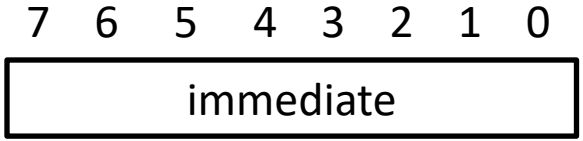




icode	b	Behavior
0		rA=rB
1		rA+=rB
2		rA&=rB
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]

```
R0 = 8
R1 = -1
R0 += R1
```



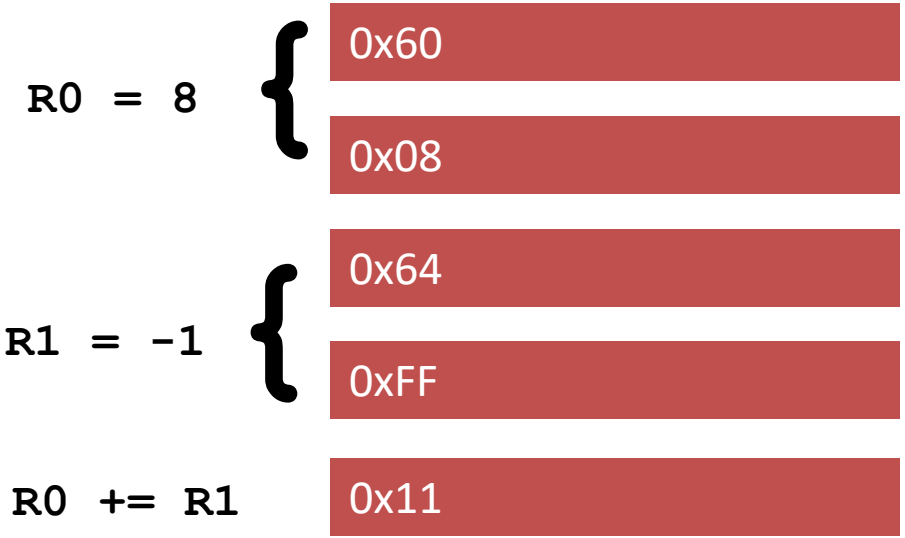


Immediate not used

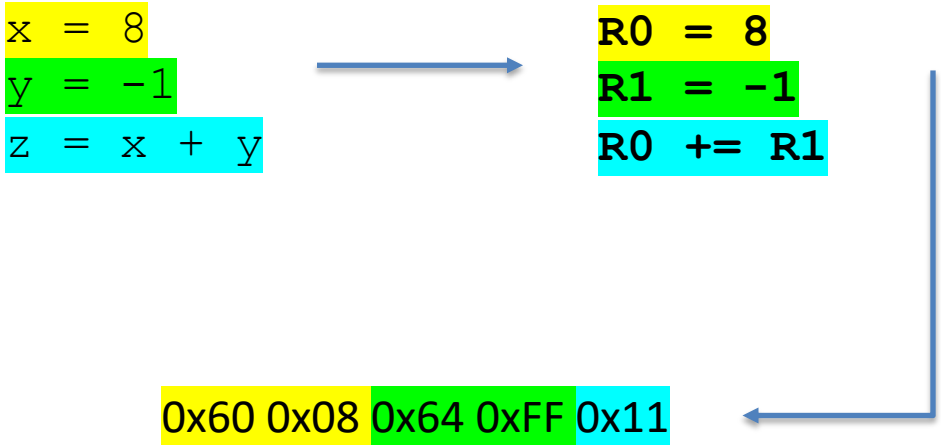
0x11


R0 = 8
0x60
0x08
R1 = -1
0x64
0xFF
R0 += R1
0x11

Notice that we have to increment the Program Counter by **two** for these instructions. Because they are two bytes long while the other instructions are only 1 byte



THE FLOW



Toy ISA Simulator

Choose File no file selected

	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	60	08	64	FF	11	00	00	00	00	00	00	00	00	00	00	00

```
ir = 00
pc = 00
 0 = 00
 1 = 00
 2 = 00
 3 = 00
```

Execute one instruction

Run with 1.5 seconds between instructions

Reset