COMPUTER SYSTEMS AND ORGANIZATION
Part 1

Instruction Set Architecture

Daniel G. Graham PhD

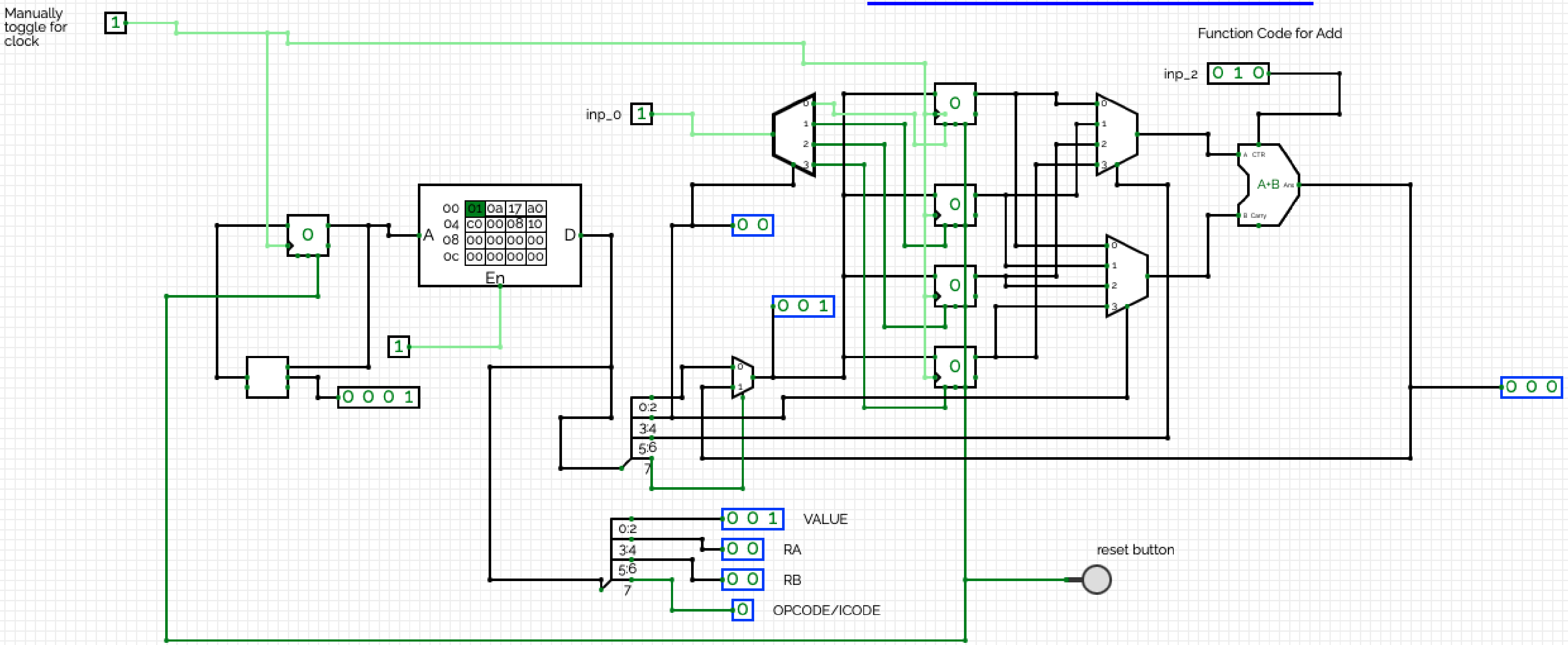
September 11, 2023

University of Virginia | Engineering

REVIEW

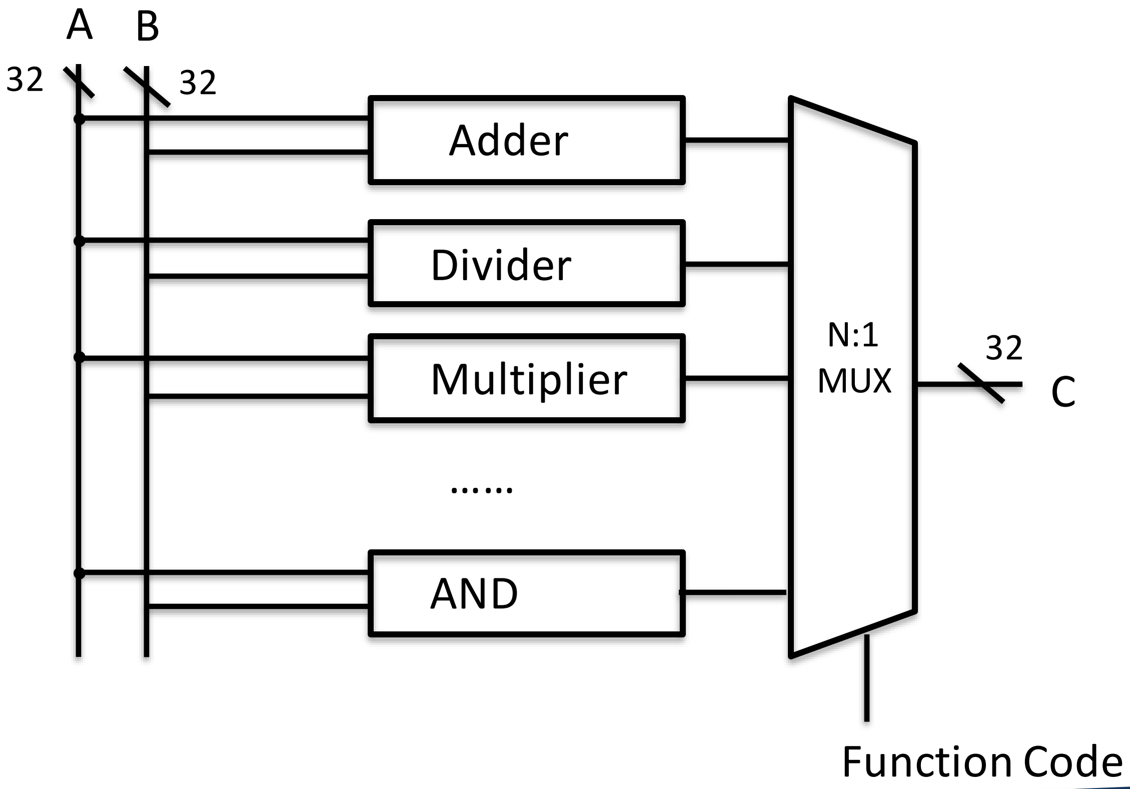
Two instruction machine : Load and Add

https://researcher111.github.io/uva-cso1-F23-DG/lectures/TWO-Instruction-Machine-Load-and-Add.cv

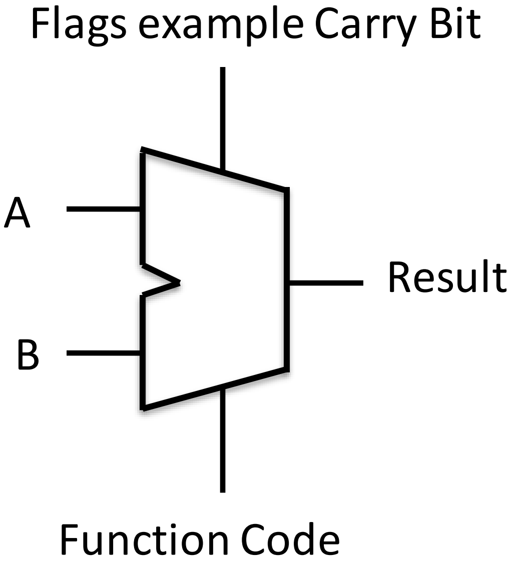


00	01	0a	17	a0
04	c0	00	08	10
08	00	00	00	00
0c	00	00	00	00

ARITHMETIC LOGIC UNIT



ALU SYMBOL AND INPUTS



Flags example Carry Bit

A

B

Result

Function Code

LET’S START BY JUST DESIGNING A MACHINE THAT LOADS VALUES

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

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

→

R0 = 1
R1 = 2
R2 = -1

We’ll map variables to registers

LET'S START BY JUST DESIGNING A MACHINE THAT LOADS VALUES

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

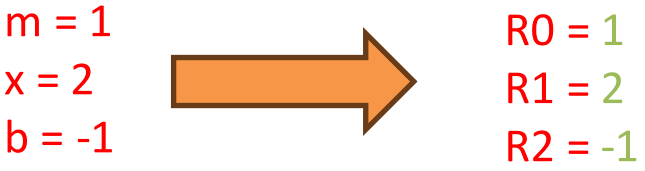
```
m = 1
x = 2
b = -1
```

R0 = 1
R1 = 2
R2 = -1

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

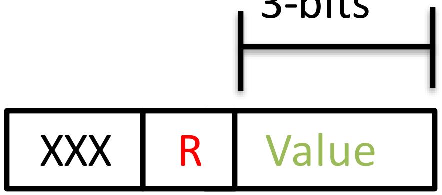
LET’S DECIDE HOW WE ARE GOING TO LAY OUT OUR BITS

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



m = 1
x = 2
b = -1

R0 = 1
R1 = 2
R2 = -1



Store the value to write

example 1 =001

2 = 010

-1 = 111

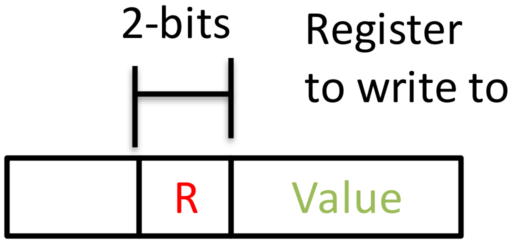
LET’S DECIDE HOW WE ARE GOING TO LAY OUT OUR BITS

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

m = 1
x = 2
b = -1

→

R0 = 1
R1 = 2
R2 = -1



State the register to write to

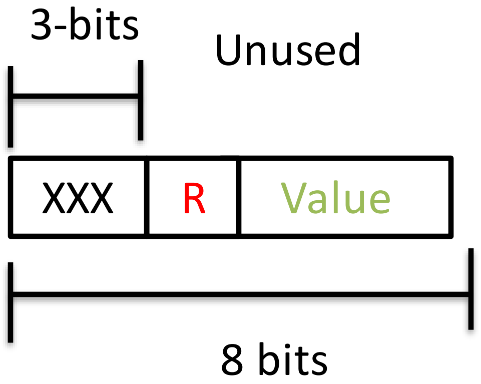
R0 = 00
R1 = 01
R2 = 10

LET’S DECIDE HOW WE ARE GOING TO LAY OUT OUR BITS

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

m = 1
x = 2
b = -1

R0 = 1
R1 = 2
R2 = -1



3-bits

Unused

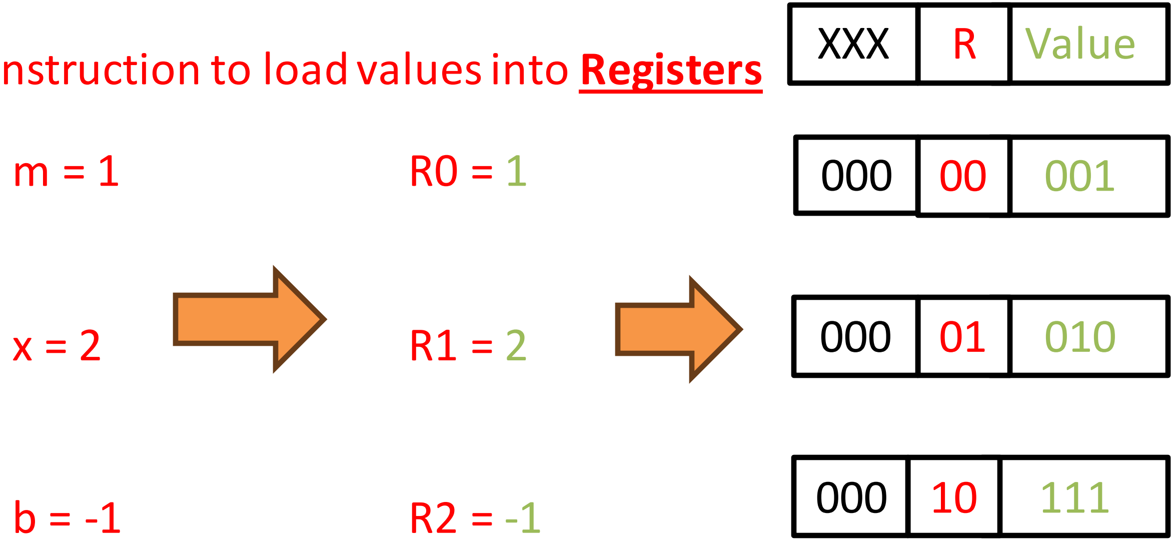
XXX	R	Value

8 bits

We just make
these zeros
XXX = 000

NOW LET'S TRANSLATE OUR PROGRAM TO ONES AND ZEROS

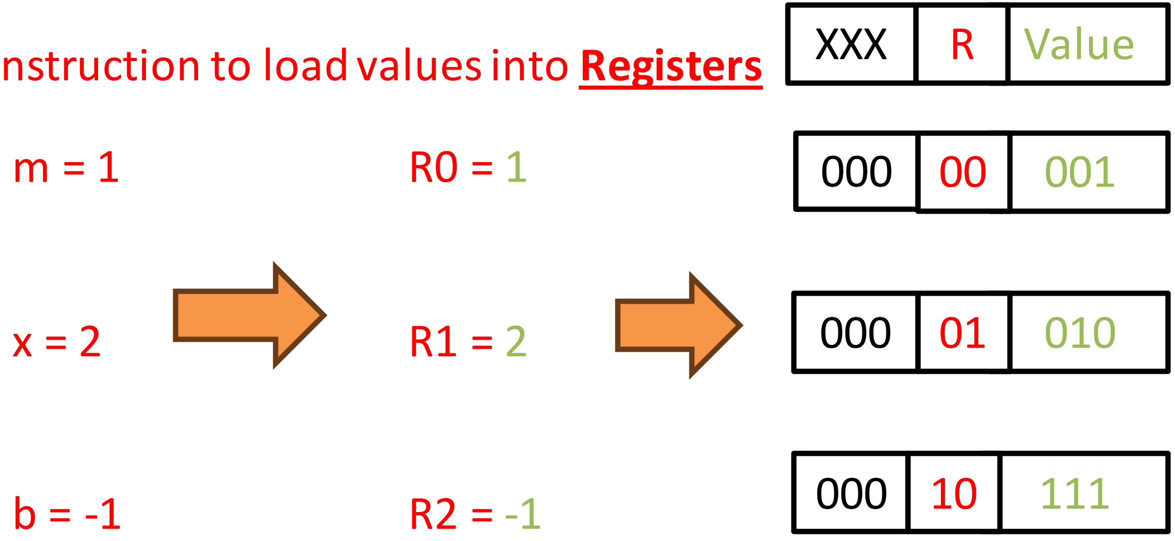
1. An instruction to load values into **Registers**



		XXX	R	Value
m = 1	R0 = 1	000	00	001
x = 2	R1 = 2	000	01	010
b = -1	R2 = -1	000	10	111

NOW LET'S TRANSLATE OUR PROGRAM TO ONES AND ZEROS

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



m = 1

x = 2

b = -1

R0 = 1

R1 = 2

R2 = -1

XXX	R	Value
000	00	001
000	01	010
000	10	111

NOW LET’S TRANSLATE OUR PROGRAM TO ONES AND ZEROS

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

		XXX	R	Value	
m = 1	R0 = 1	000	00	001	0x01
x = 2	R1 = 2	000	01	010	0x0A
b = -1	R2 = -1	000	10	111	0x17

GREAT! WE HAVE OUR FIRST INSTRUCTION

XXX	RA	Value

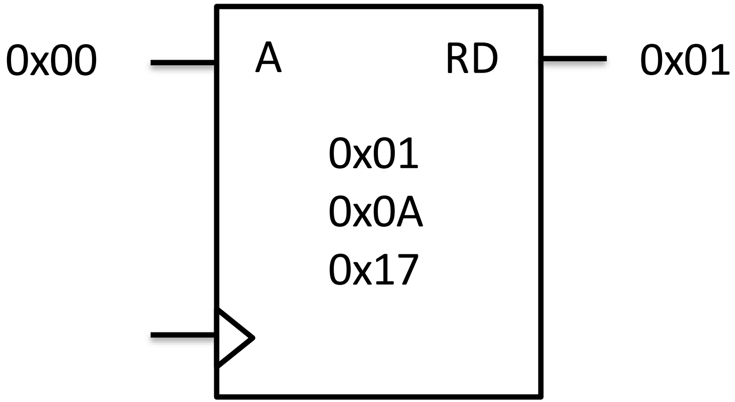
RA = Value

SO, WHAT GETS LOADED INTO MEMORY

Great! So, we converted our program to hex and loaded it into memory.

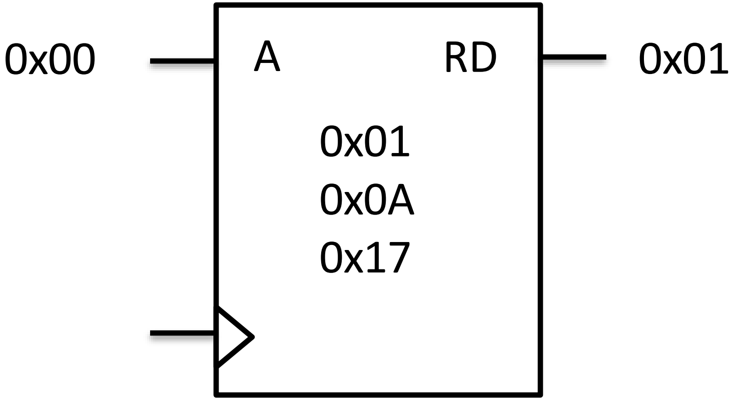
m = 1	R0 = 1
x = 2	R1 = 2
b = -1	R2 = -1

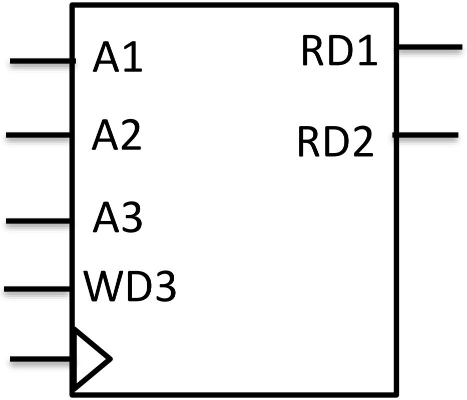
We still need to load our values into Registers.



LET'S ADD OUR REGISTER FILE

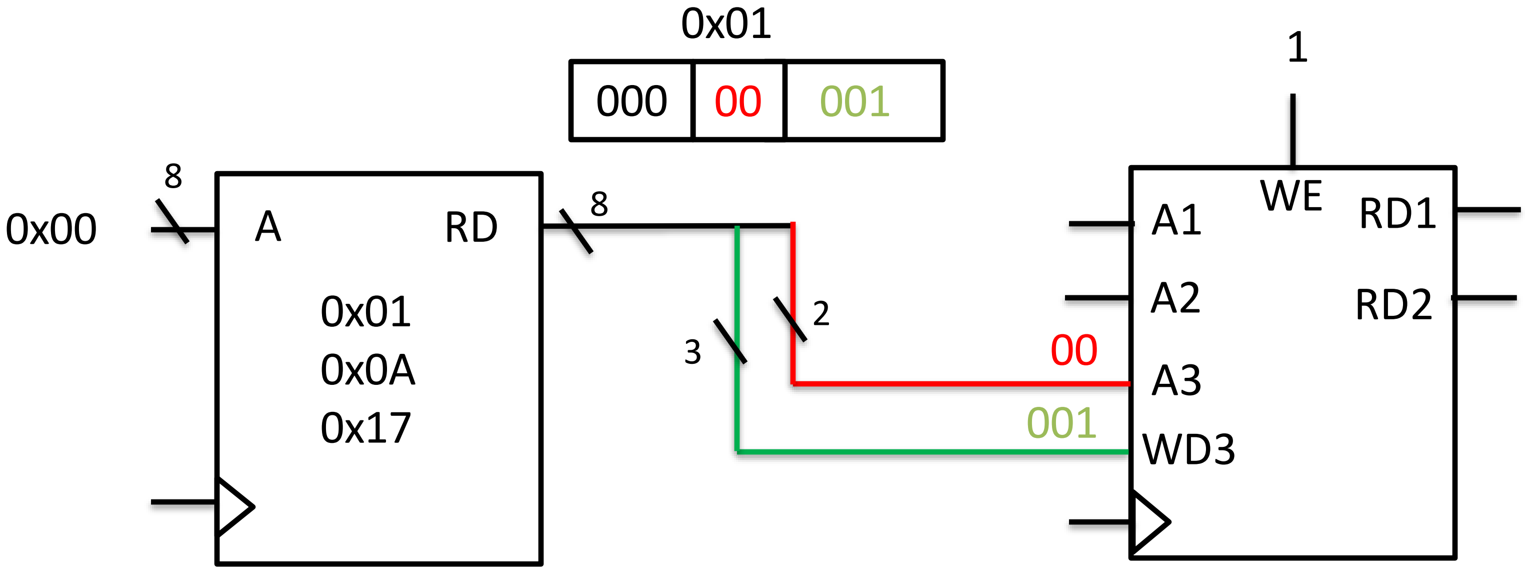
m = 3	R0 = 1
x = 2	R1 = 2
b = -1	R2 = -1





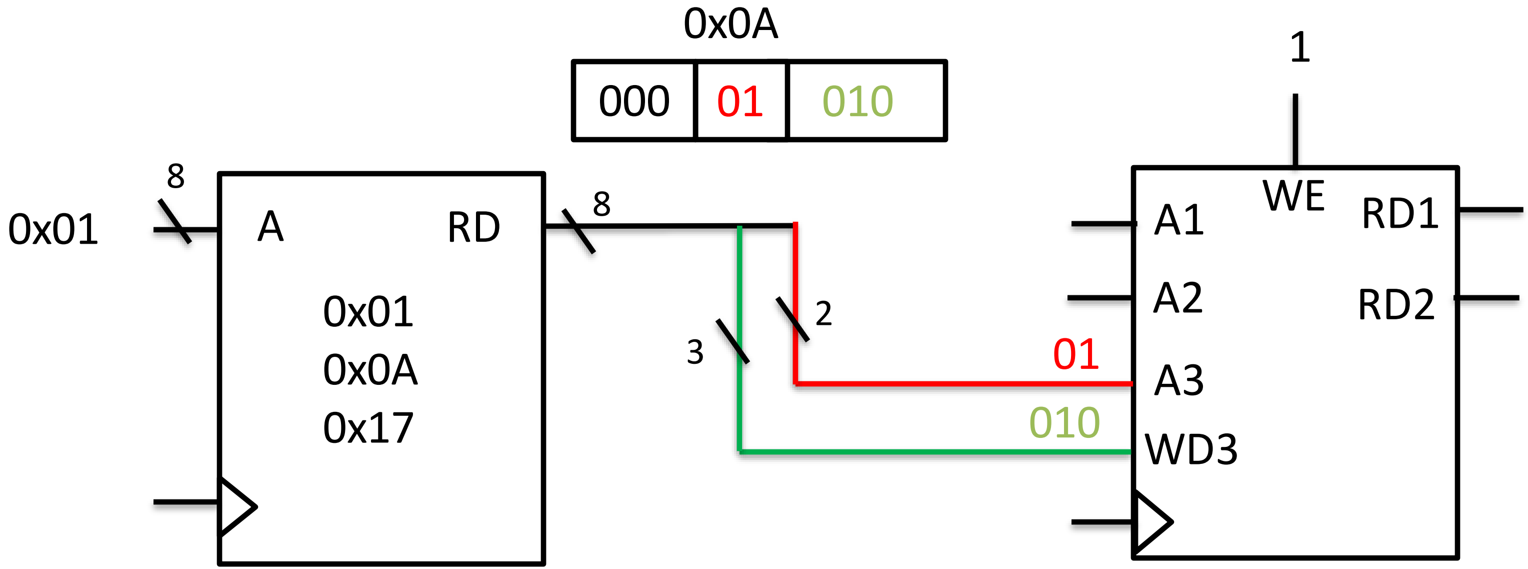
LET'S ADD OUR REGISTER FILE

R0 = 1
R1 = 2
R2 = -1



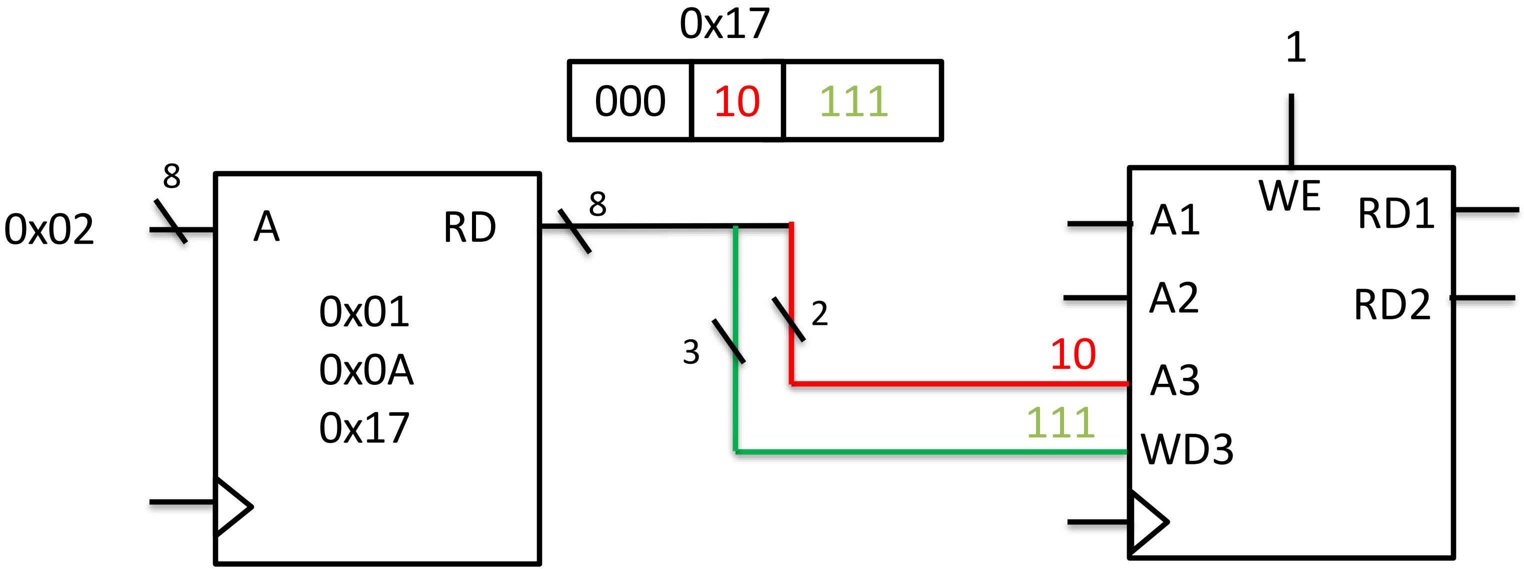
LET'S ADD OUR REGISTER FILE

R0 = 1
R1 = 2
R2 = -1



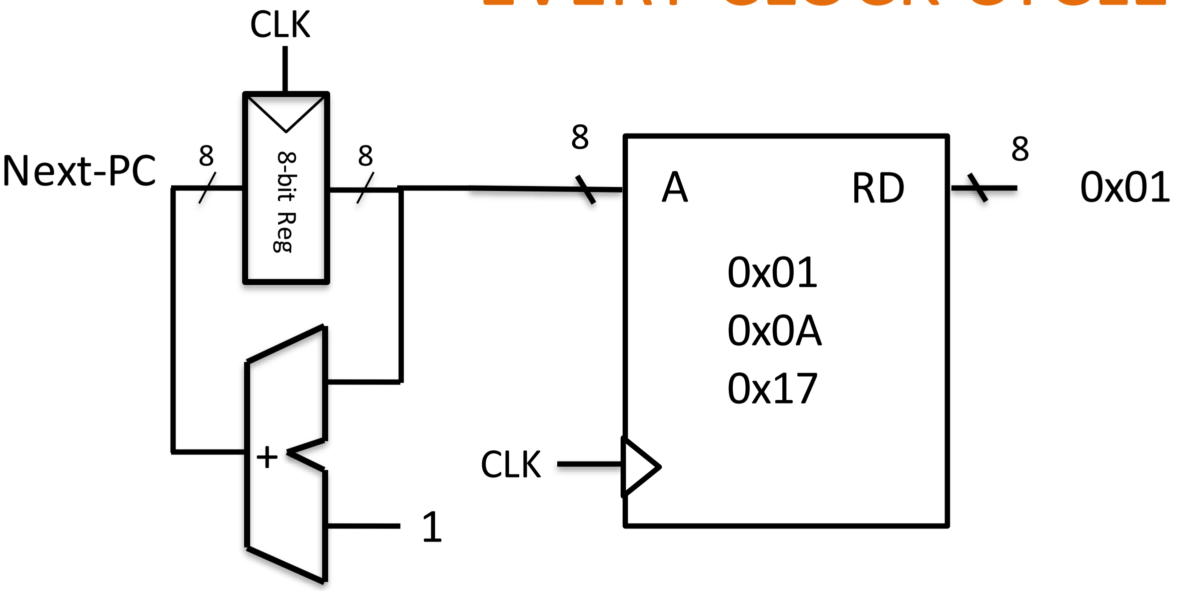
LET'S ADD OUR REGISTER FILE

R0 = 1
R1 = 2
R2 = -1



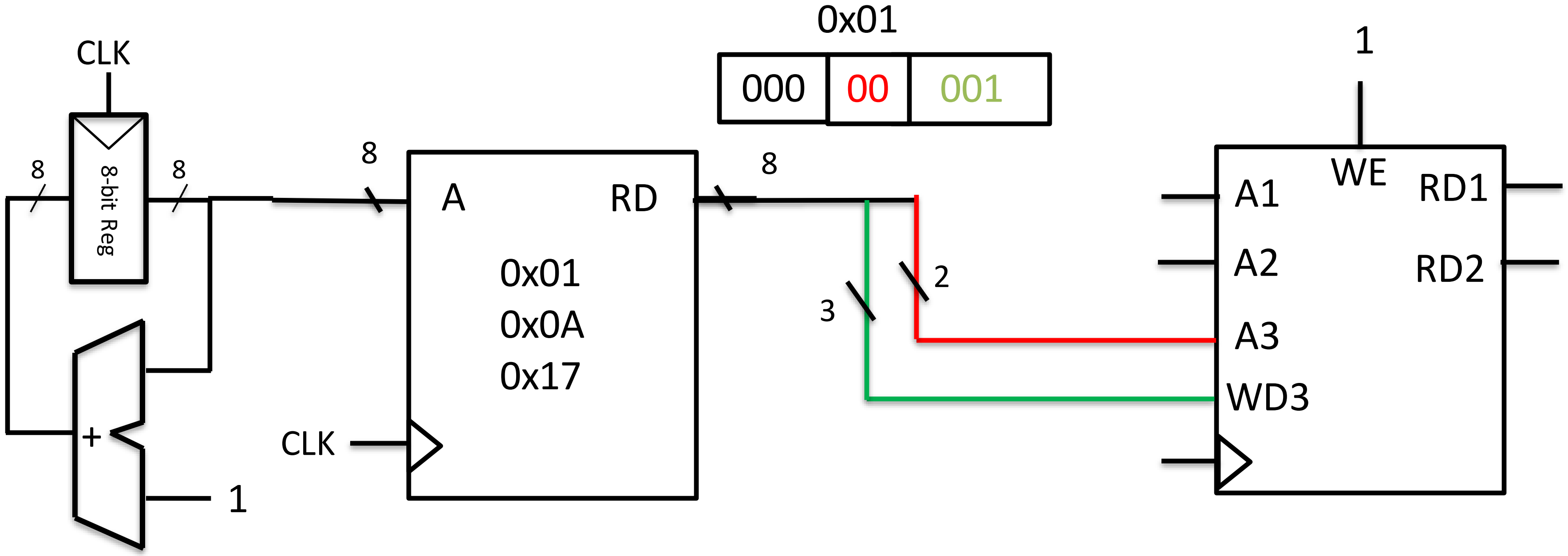
HOW CAN WE AUTOMATICALLY CHANGE THE ADDRESS WITH EVERY CLOCK CYCLE?

AUTOMATICALLY FETCH A NEW INSTRUCTION EVERY CLOCK CYCLE

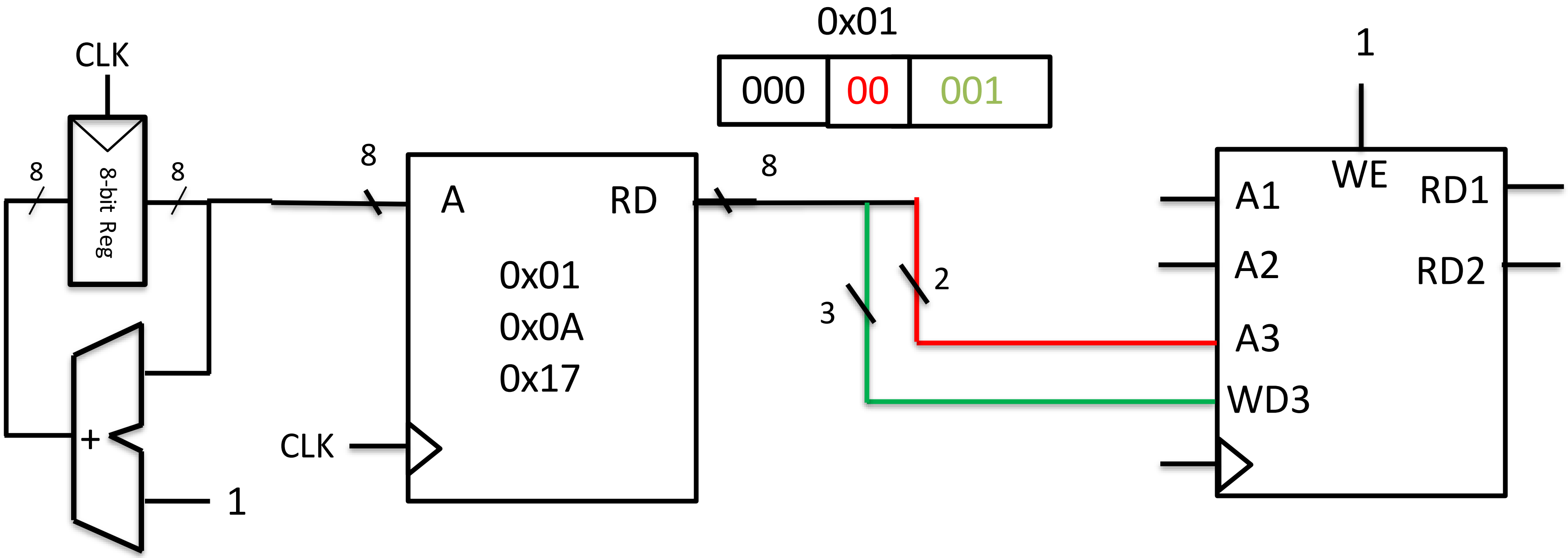


n-bit PC

NOW LET'S ADD OUR REGISTER FILE



8-bit PC



8-bit PC

Our program would have loaded values into the register file

R0 = 1

R1 = 2

R2 = -1

GREAT! WE LOADED THE VALUES.

WHAT ABOUT ADDITION?

An instruction to load values into **Registers**

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

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

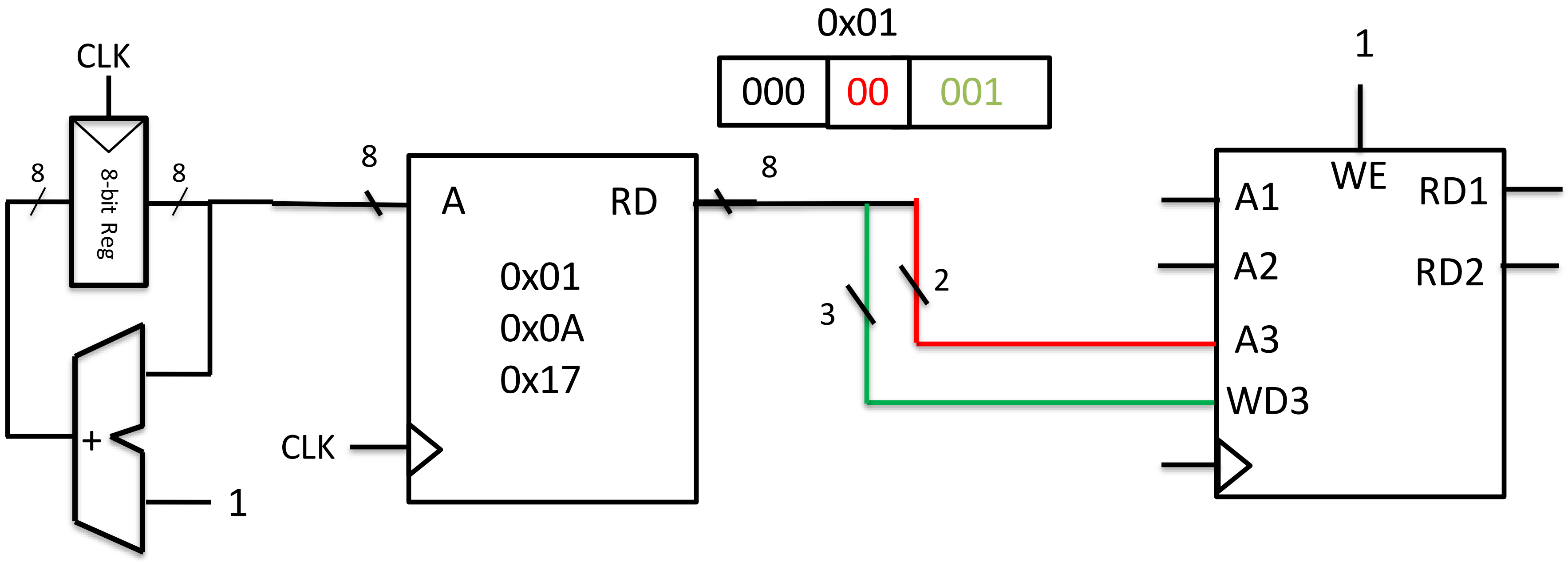
But how do we encode this in bits so that we can execute it?

An instruction to computation (addition)

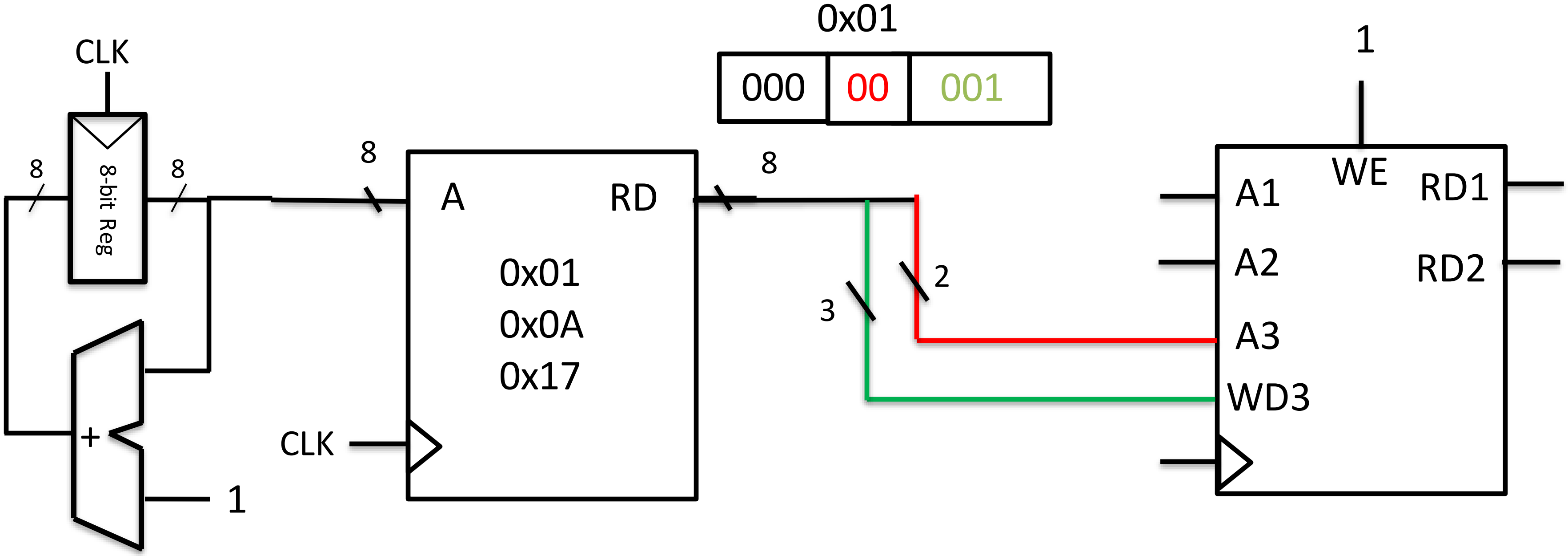
```
y = m+x+b
```

R0 += R1 ← m = m + x
R0 += R2 ← m = m + b

NOW LET'S ADD OUR REGISTER FILE



8-bit PC



8-bit PC

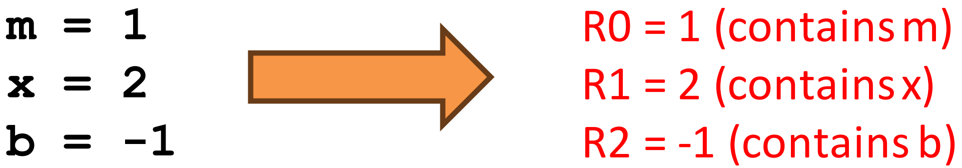
Our program would have loaded values into the register file

R0 = 1
R1 = 2
R2 = -1

GREAT! WE LOADED THE VALUES.

WHAT ABOUT ADDITION?

An instruction to load values into **Registers**

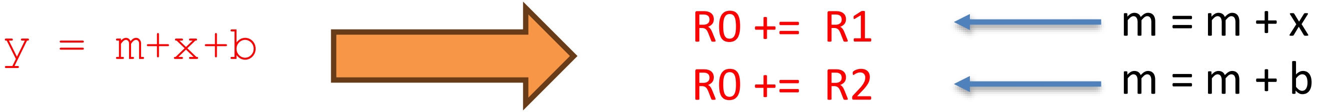


```
m = 1
x = 2
b = -1
```

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

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

An instruction to computation (addition)



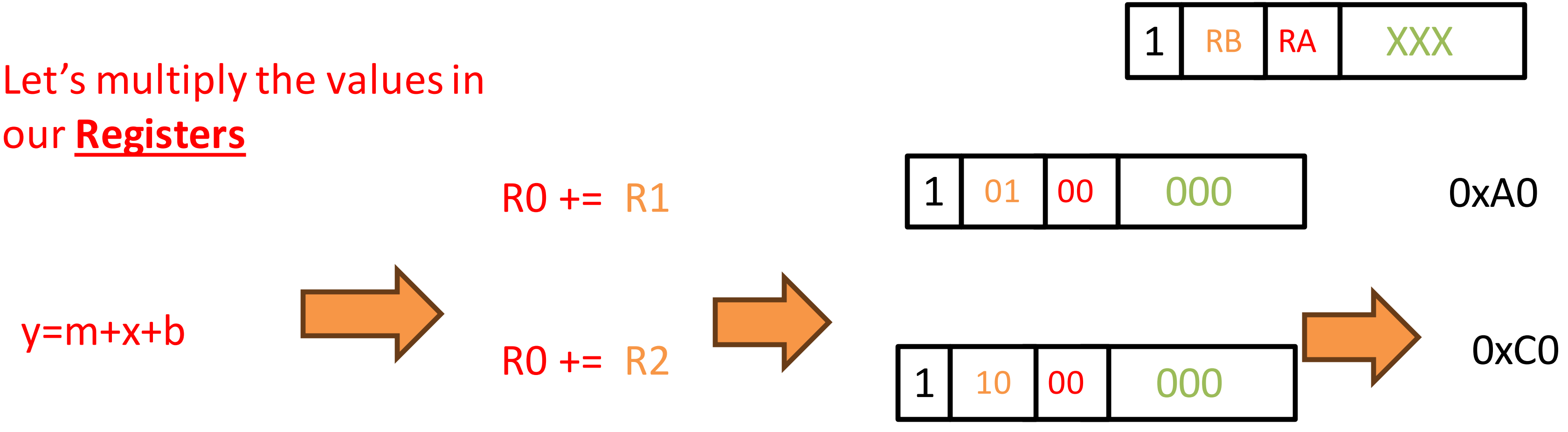
```
y = m+x+b
```

R0 += R1
R0 += R2

m = m + x
m = m + b

ENCODING

Let’s multiply the values in our **Registers**



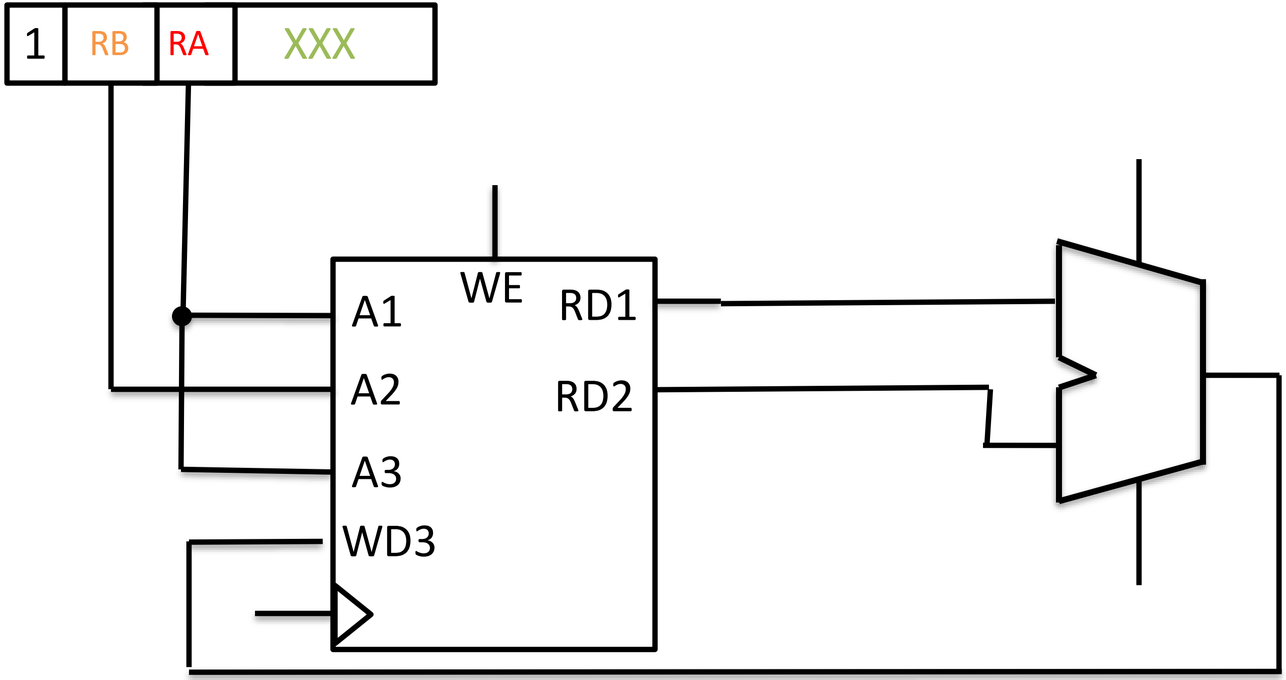
y=m+x+b

R0 += R1

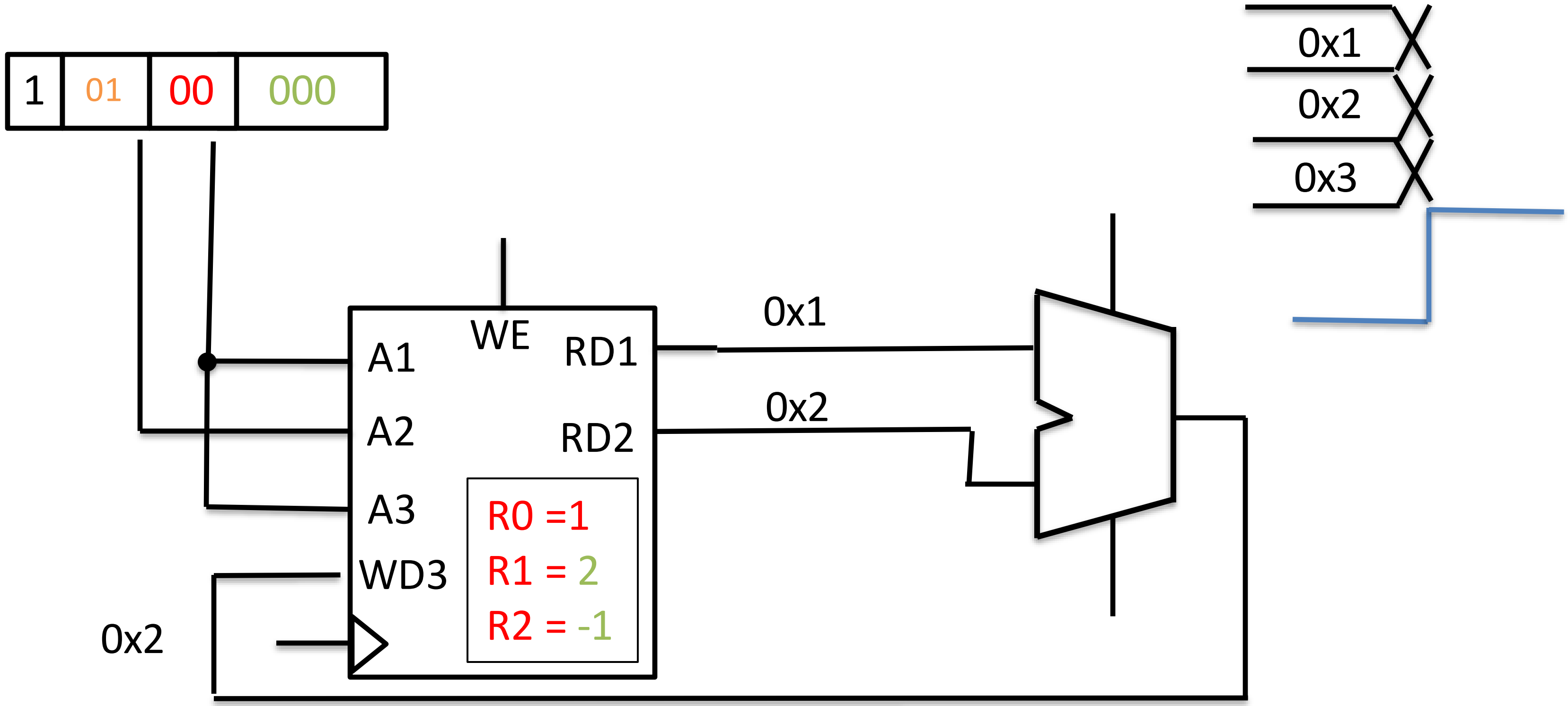
R0 += R2

1	RB	RA	XXX	
1	01	00	000	0xA0
1	10	00	000	0xC0

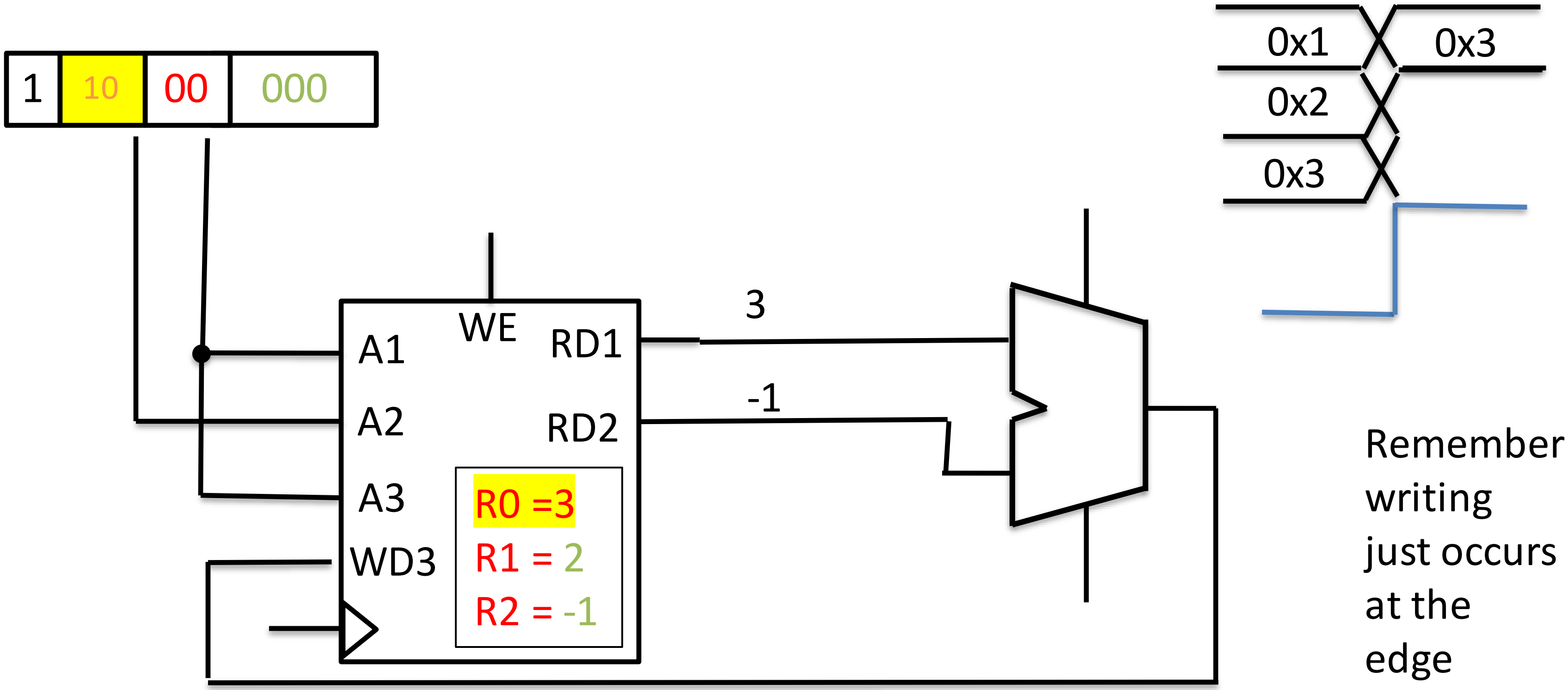
BUILDING MACHINE TO COMPUTE THIS



BUILDING MACHINE TO COMPUTE THIS

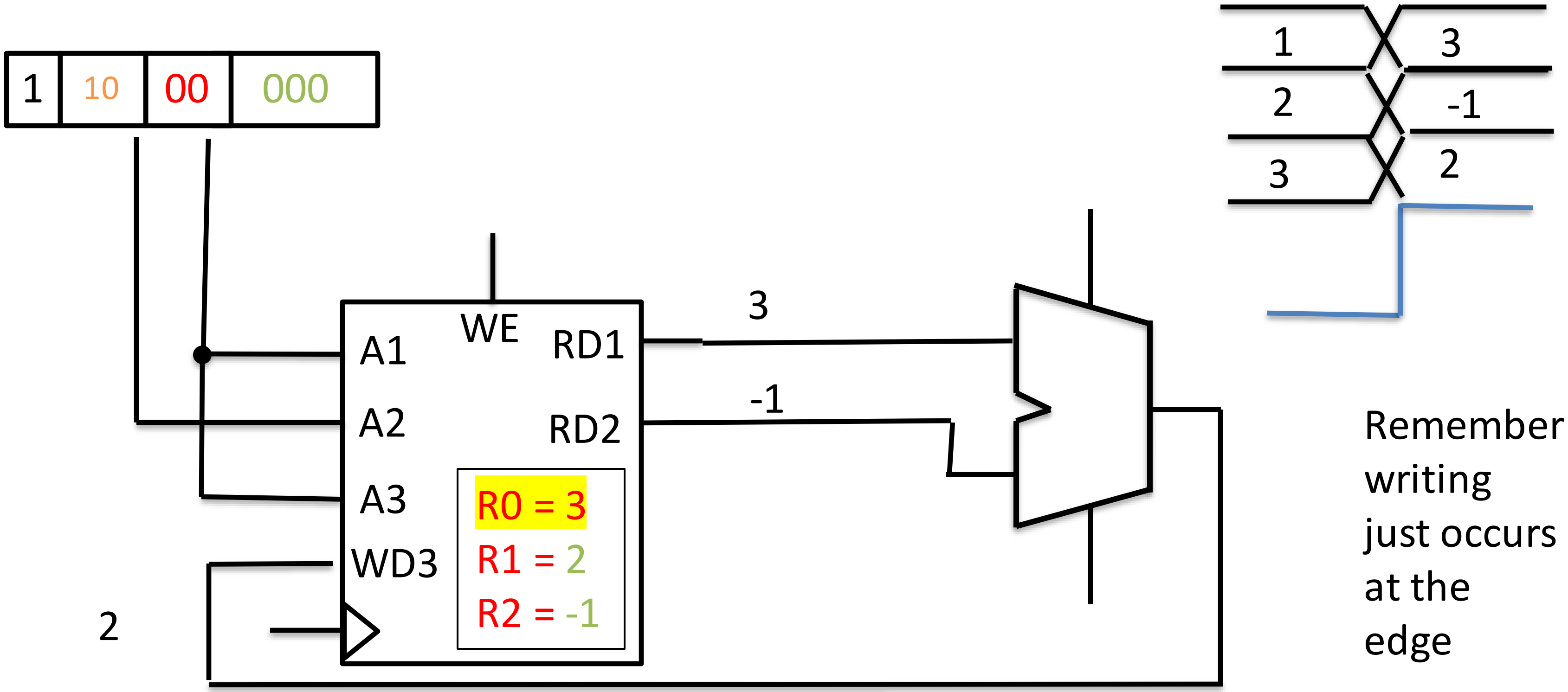


BUILDING MACHINE TO COMPUTE THIS



Remember writing just occurs at the edge

BUILDING MACHINE TO COMPUTE THIS



Remember writing just occurs at the edge

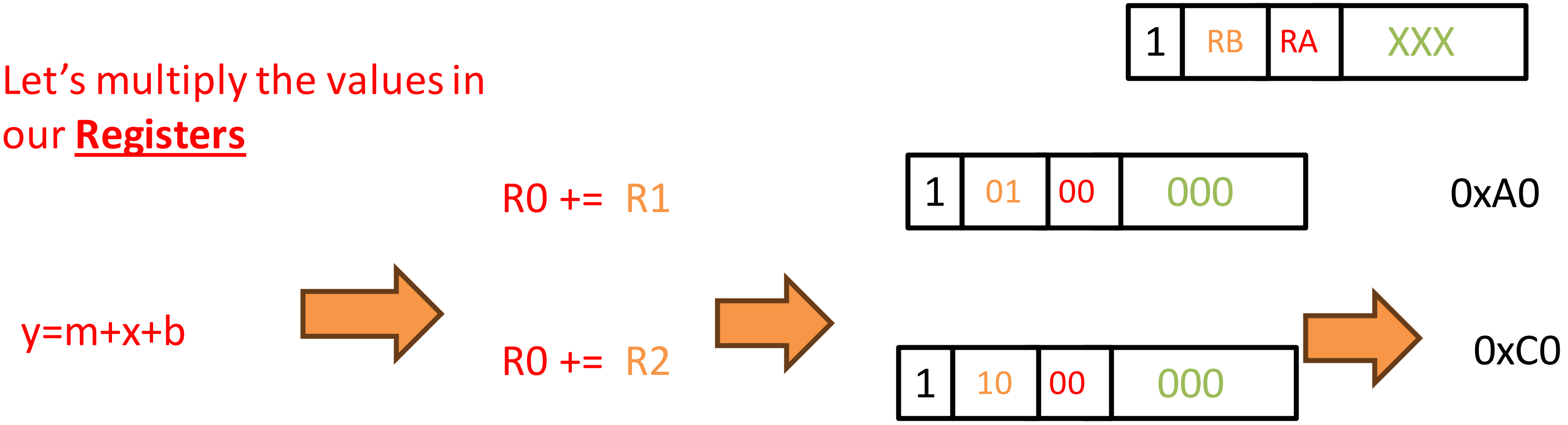
ENCODING

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

		XXX	R	Value	
m = 1	R0 = 1	000	00	001	0x01
x = 2	R1 = 2	000	01	010	0x0A
b = -1	R2 = -1	000	10	111	0x17

ENCODING

Let’s multiply the values in our **Registers**



y=m+x+b

R0 += R1

R0 += R2

1	RB	RA	XXX
1	01	00	000
1	10	00	000

0xA0

0xC0

FINAL PROGRAM

m = 1 R0 = 1 0x01

x = 2 R1 = 2 0x0A

b = -1 R2 = -1 0x17

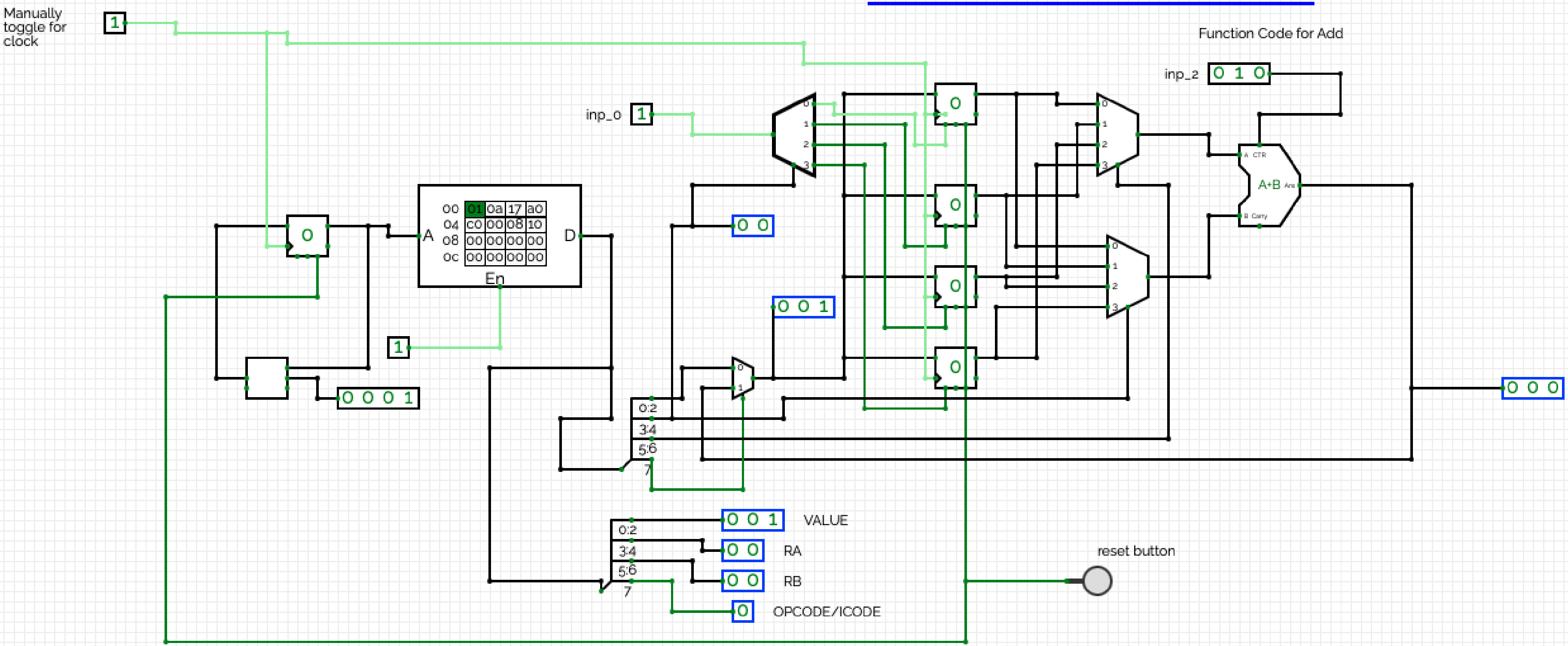
y=m+x+b

R0 += R1 0xA0

R0 += R2 0xC0

Two instruction machine : Load and Add

https://researcher111.github.io/uva-cso1-F23-DG/lectures/TWO-Instruction-Machine-Load-and-Add.cv



00	01	0a	17	a0
04	c0	00	08	10
08	00	00	00	00
0c	00	00	00	00

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

7	6	5	4	3	2	1	0
R	icode			a		b	

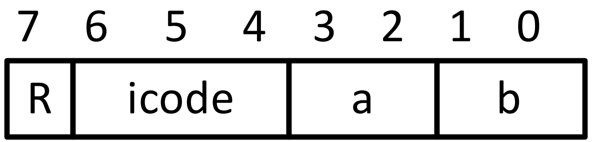
byte at `pc`

7	6	5	4	3	2	1	0
immediate							

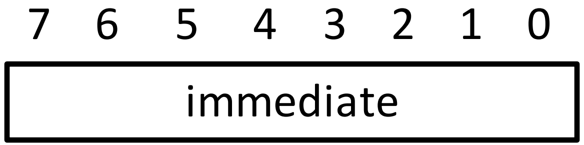
byte at `pc + 1`

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

7	6 5 4	3 2	1 0
R	icode	a	b

byte at pc

7 6 5 4 3 2 1 0
immediate

byte at pc + 1

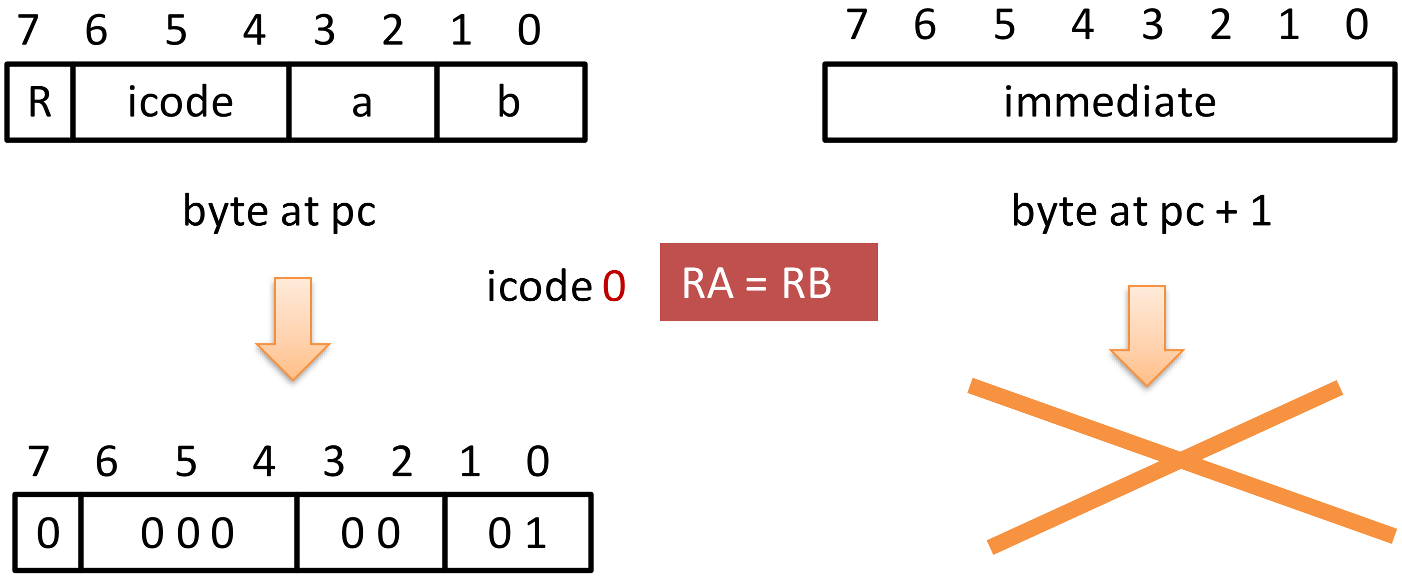
icode 0 RA = RB

7	6 5 4	3 2	1 0
0	0 0 0	0 0	0 1

0x01

ENCODING OUR FIRST INSTRUCTION

Try to encode the following instruction R0 = R1



7	6 5 4	3 2	1 0
R	icode	a	b

byte at pc

7 6 5 4 3 2 1 0
immediate

byte at pc + 1

icode 0 RA = RB

7	6 5 4	3 2	1 0
0	0 0 0	0 0	0 1

Not used
This
instruction
is not using
a value

ENCODING OUR FIRST INSTRUCTION

Try to encode the following instruction R0 = R1

7 6 5 4 3 2 1 0

R	icode	a	b

byte at pc

7 6 5 4 3 2 1 0

immediate

byte at pc + 1

icode 0 RA = RB

7 6 5 4 3 2 1
0

0	0 0 0	0 0	0 1

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)

7	6	5	4	3	2	1	0
R	icode			a		b	

byte at pc

7	6	5	4	3	2	1	0
immediate							

byte at pc + 1

↓

7	6	5	4	3	2	1	0
0	0 0 1			11		0 1	

→ 0x1D

ACTIVITY

Write the following instruction r2 &= r3 in hex

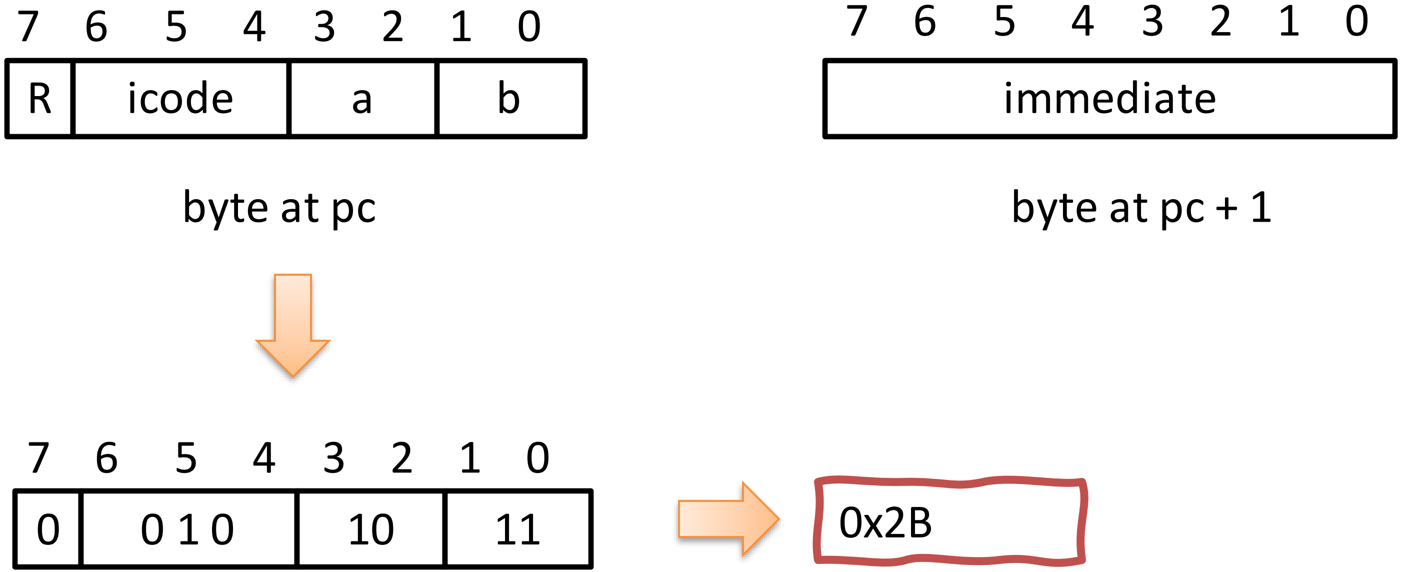
icode	Behavior
0	rA=rB
1	rA+=rB
2	rA&=rB

7	6	5	4	3	2	1	0
R	icode			a		b	

7	6	5	4	3	2	1	0
immediate							

icode	Behavior
2	rA&=rB

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



7 6 5 4 3 2 1 0

R | icode | a | b

byte at pc

7 6 5 4 3 2 1 0

immediate

byte at pc + 1

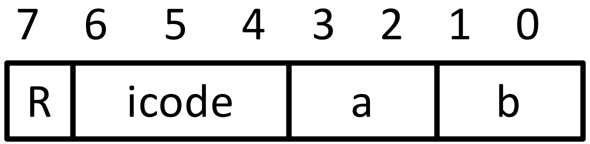
7 6 5 4 3 2 1 0

0 | 0 1 0 | 10 | 11

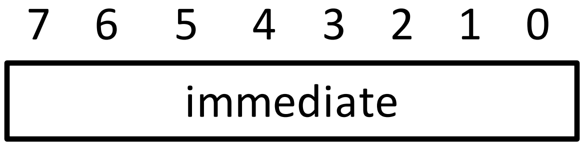
0x2B

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)



7	6	5	4	3	2	1	0
R	icode			a		b	



7	6	5	4	3	2	1	0
immediate							

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------

7	6	5	4	3	2	1	0
R	icode			a		b	

7	6	5	4	3	2	1	0
immediate							

PUT IT ALL TOGETHER

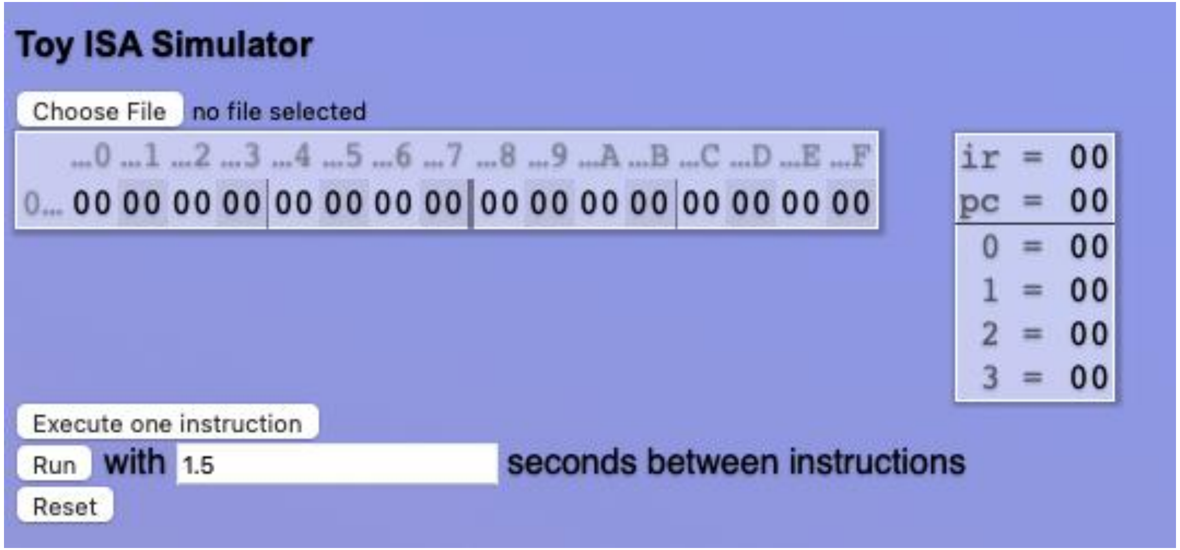
icode	b	Behavior
0		rA=rB
1		rA+=rB
2		rA&=rB
6	0	rA=read from memory at pc + 1 Also written as rA = M[pc+1]

7	6	5	4	3	2	1	0
R	icode			a		b	

7	6	5	4	3	2	1	0
immediate							

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



Toy ISA Simulator

no file selected

	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

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

with 1.5 seconds between instructions

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
```

7	6	5	4	3	2	1	0
R	icode			a		b	

7	6	5	4	3	2	1	0
immediate							

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
```

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="3">icode</td><td colspan="2">a</td><td colspan="2">b</td></tr>
<tr><td>0</td><td colspan="3">110</td><td colspan="2">00</td><td colspan="2">00</td></tr>
</table>

0x60

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">immediate</td></tr>
<tr><td colspan="8">0 0 0 0 1 0 0 0</td></tr>
</table>

0x08

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
```

7	6 5 4	3 2	1 0
R	icode	a	b
0	110	01	00

0x64

7 6 5 4 3 2 1 0
immediate
1 1 1 1 1 1 1 1

0xFF

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
```

7 6 5 4 3 2 1 0

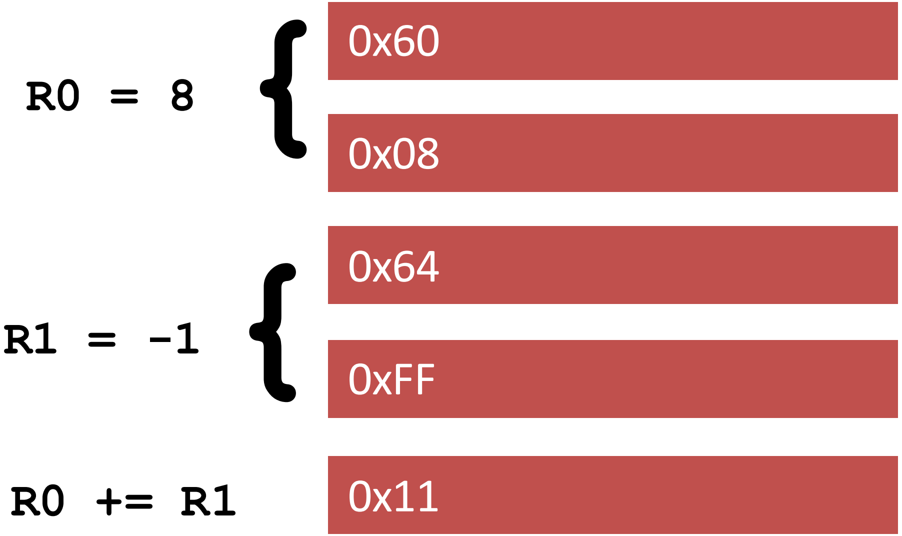
R	icode	a	b
0	001	00	01

0x11

7 6 5 4 3 2 1 0

immediate

Immediate not used



`R0 = 8`

0x60

0x08

`R1 = -1`

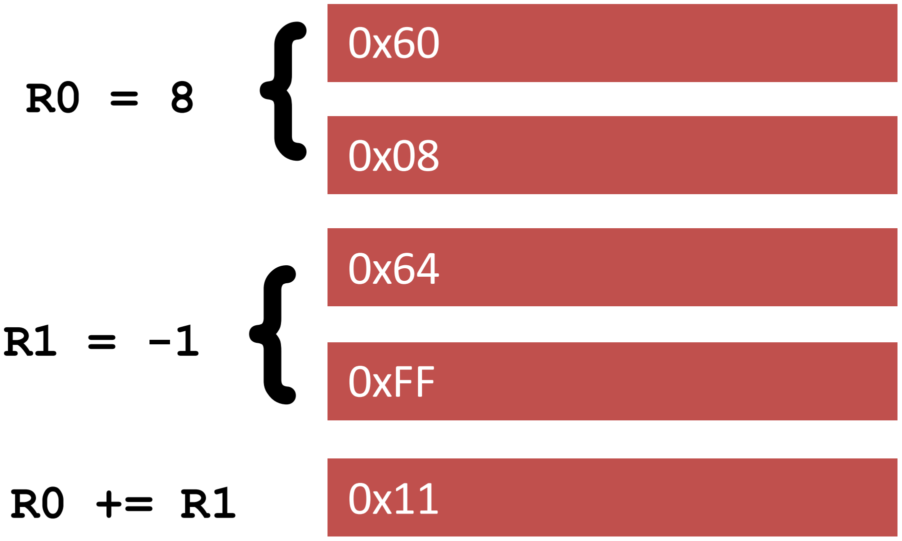
0x64

0xFF

`R0 += R1`

0x11

Notice that we have to increment the Program Counter by **two** for these instructions. Instructions that read from the immediate, like icode 6, are two bytes long while the other instructions are only 1 byte.



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

0x60

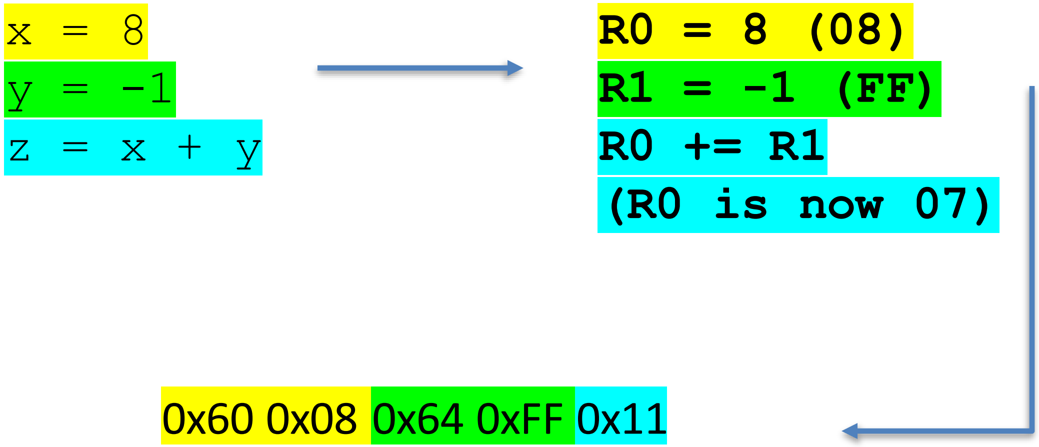
0x08

0x64

0xFF

0x11

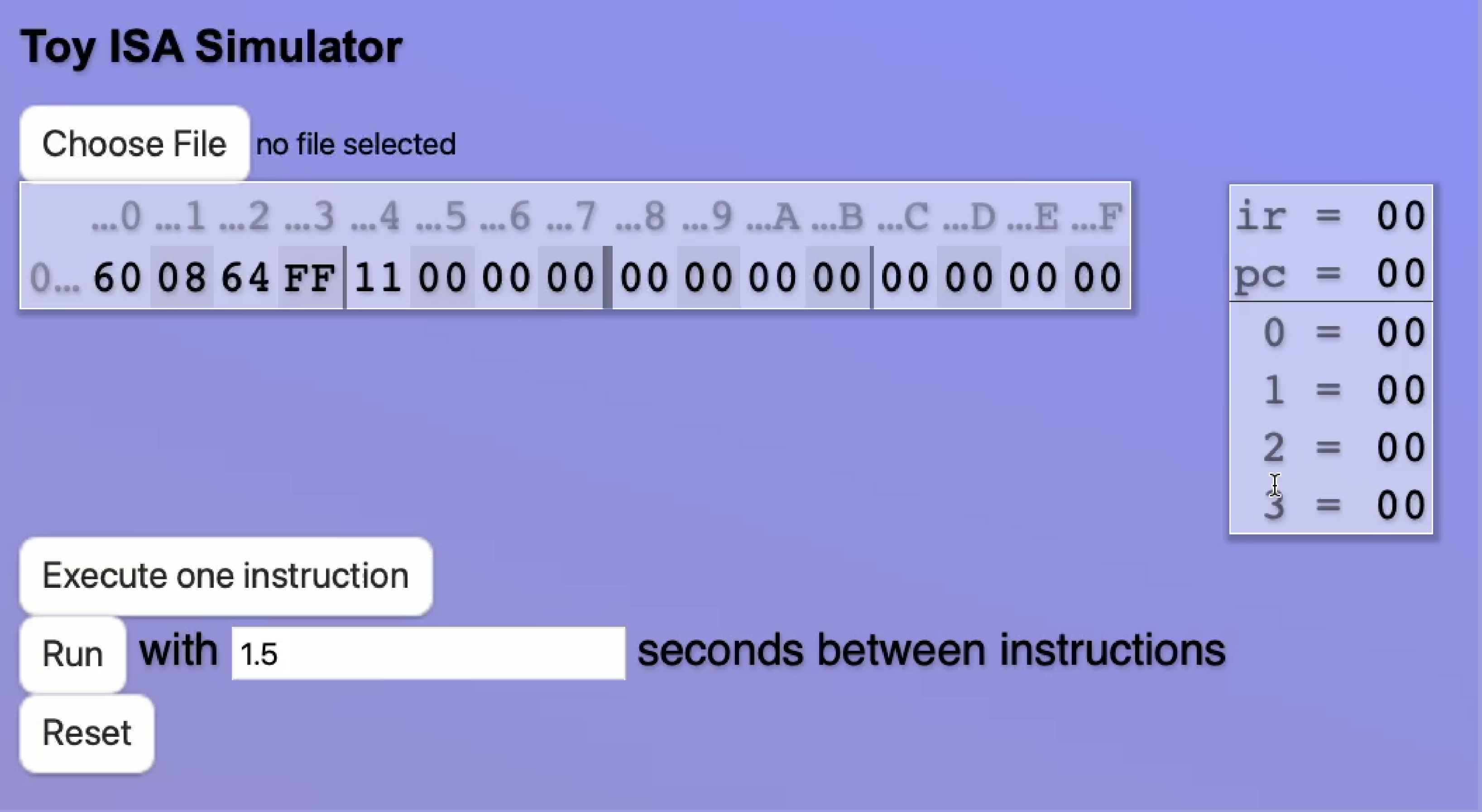
THE FLOW



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

```
R0 = 8 (08)
R1 = -1 (FF)
R0 += R1
(R0 is now 07)
```

```
0x60 0x08 0x64 0xFF 0x11
```



Toy ISA Simulator

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
```

Run with 1.5 seconds between instructions

UNIVERSITY of VIRGINIA | ENGINEERING