COMPUTER SYSTEMS AND ORGANIZATION Part 1

Instruction Set Architecture

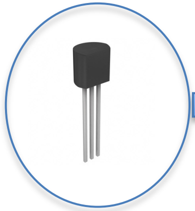
Daniel G. Graham PhD

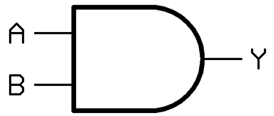
September 11, 2023

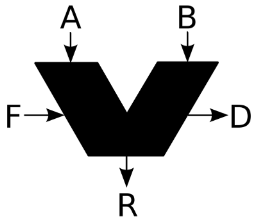
UNIVERSITY of VIRGINIA | ENGINEERING

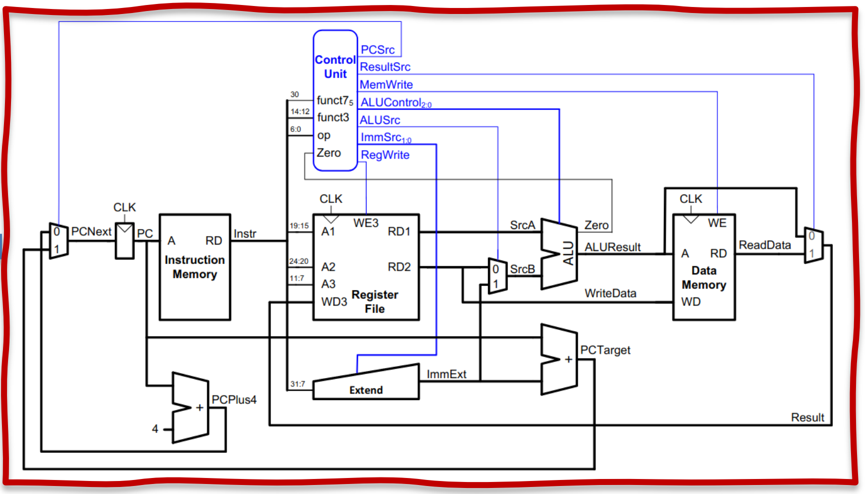
REVIEW

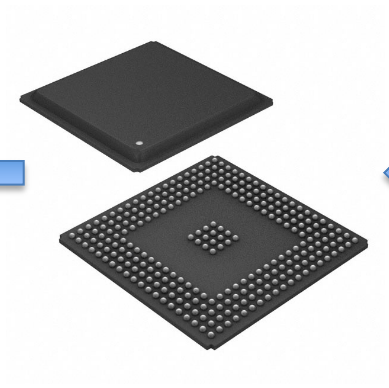
THE MAP (THE MACHINE)





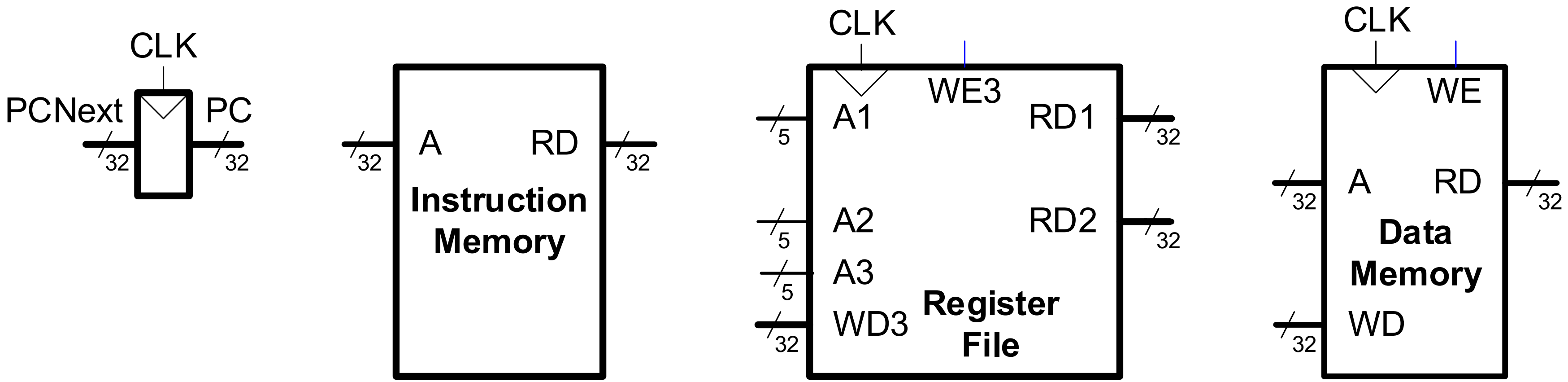




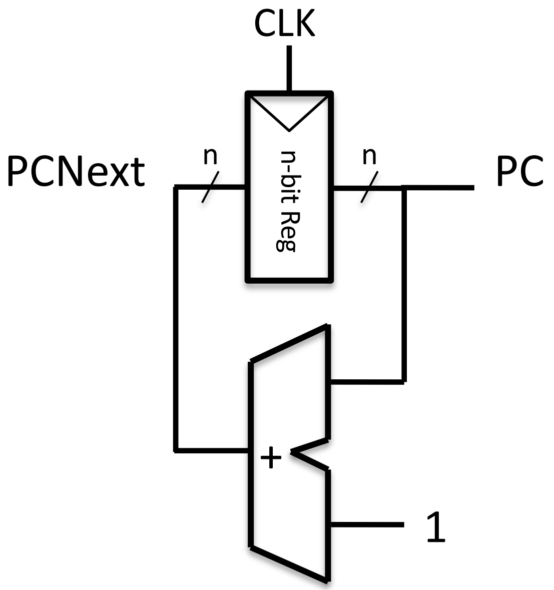


https://github.com/MKrekker/SINGLE-CYCLE-RISC-V

MEMORY COMPONENTS OF A PROCESSOR

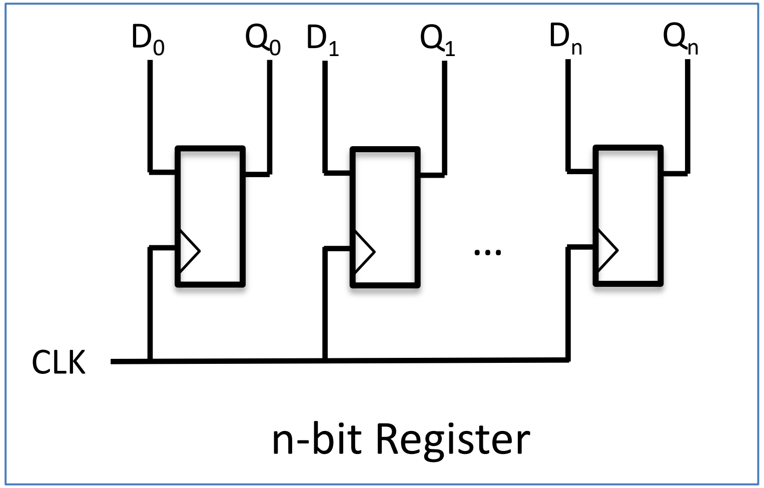


PROGRAM COUNTER



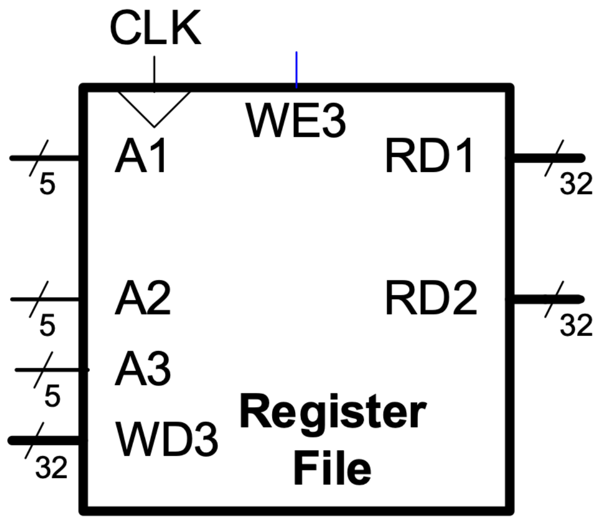
n-bit PC

- To track where we are in a program

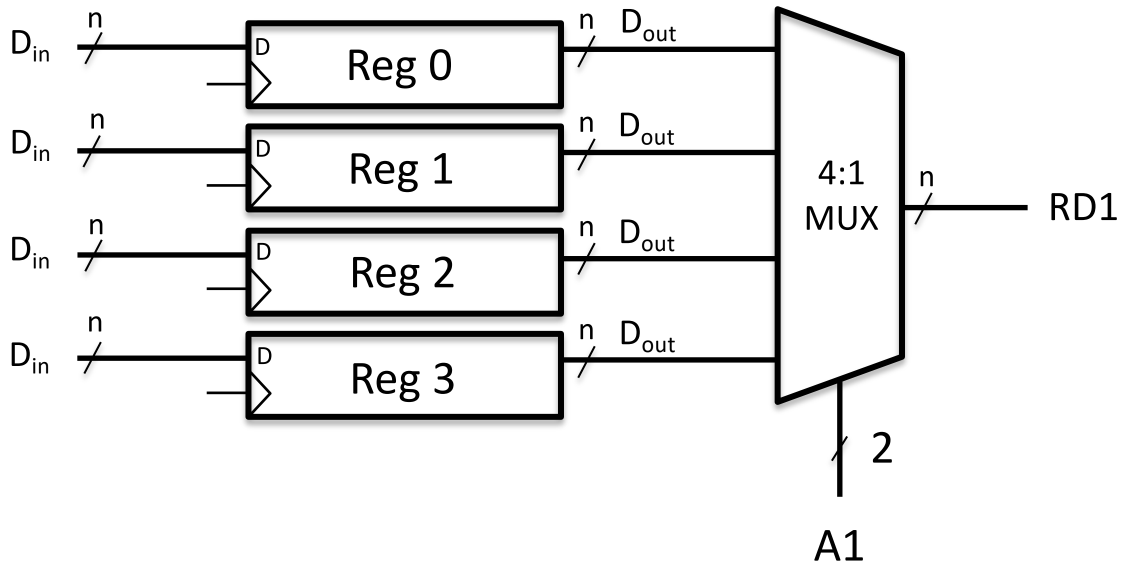


REGISTER FILE

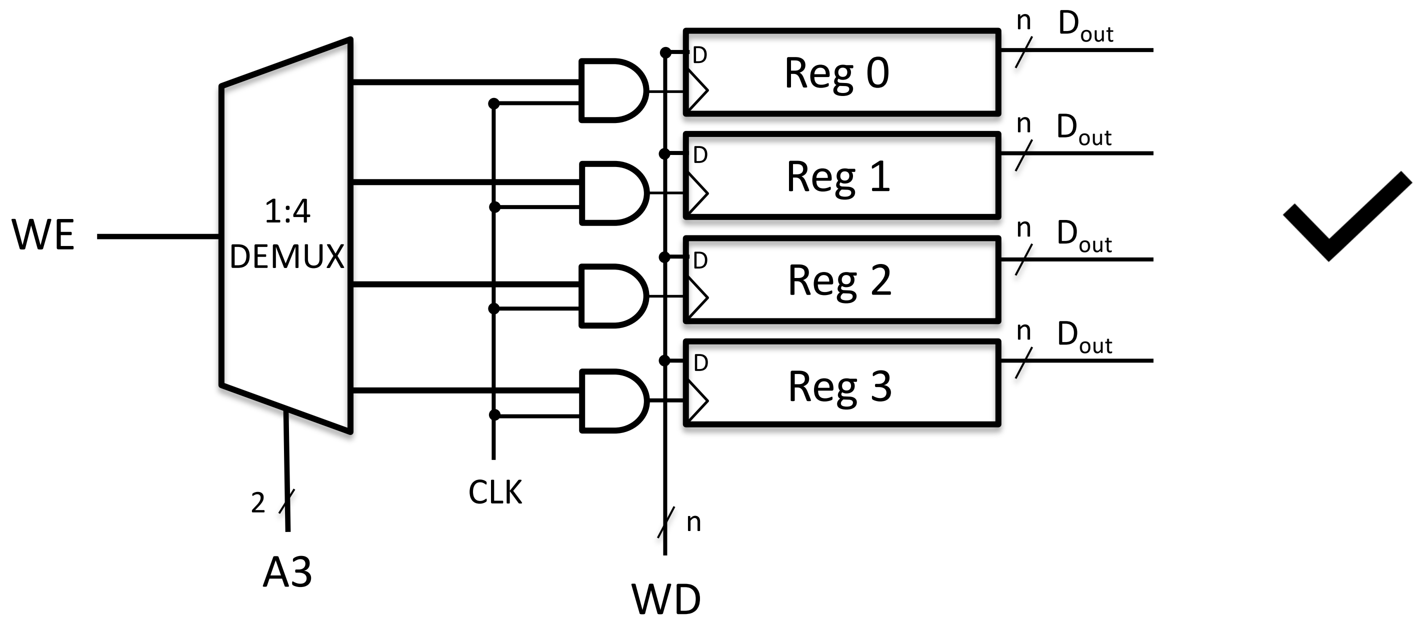
- Temporary storage location
- Stores immediately needed variables
- External interface
  - Addresses: A1, A2, A3
  - Data: RD1, RD2, WD3
  - Enable: WE3
  - Clock: CLK



READ FROM A REGISTER FILE



WRITE TO A REGISTER FILE



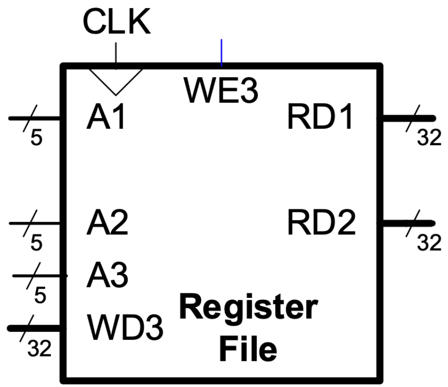
UNIVERSITY of VIRGINIA | ENGINEERING

32 32-BIT REGISTER FILE

Simultaneously read from two registers and write into one register

Components:

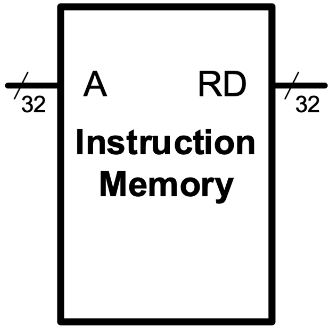
1. Multiplexers
2. Registers
3. Demultiplexers



INSTRUCTION MEMORY

- Stores the program

➢ Read data (RD) for a given address (A)



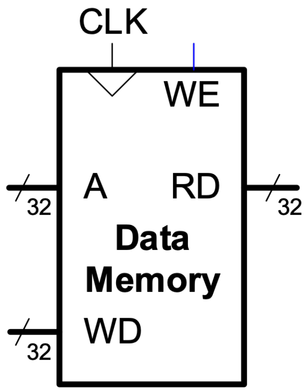
For this class, we will assume we cannot write to Instruction Memory.

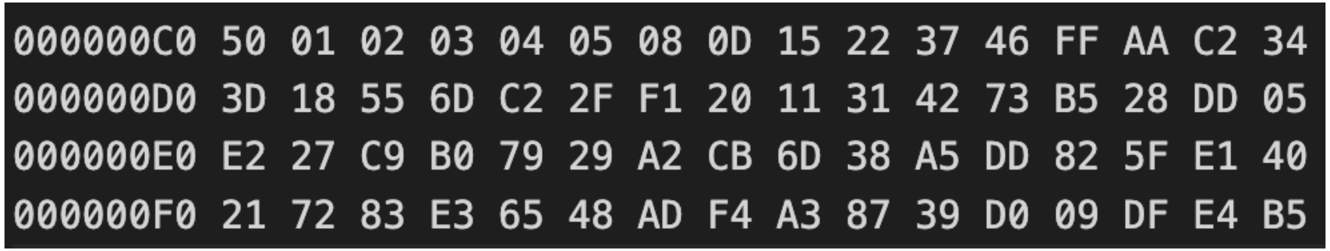
DATA MEMORY

- Contains data needed by the program

➢ Read data (RD) from a given address (A)

➢ Write data (WD) to a given address (A)





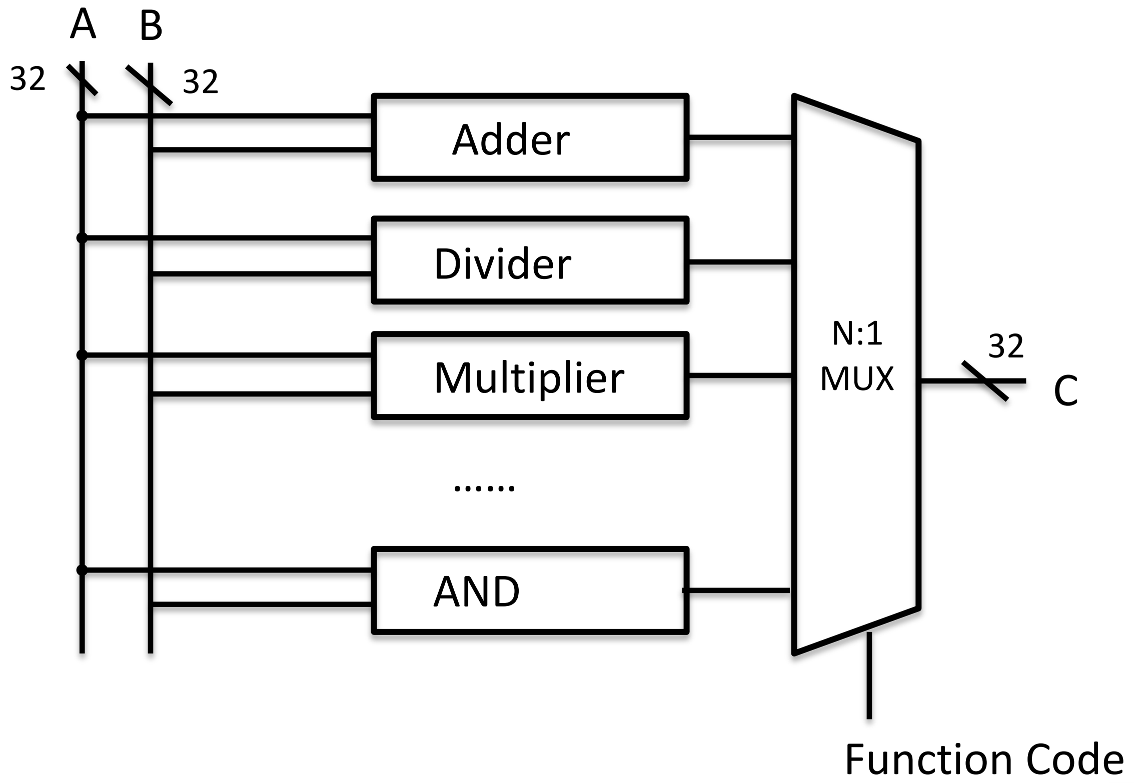
```
000000C0 50 01 02 03 04 05 08 0D 15 22 37 46 FF AA C2 34
000000D0 3D 18 55 6D C2 2F F1 20 11 31 42 73 B5 28 DD 05
000000E0 E2 27 C9 B0 79 29 A2 CB 6D 38 A5 DD 82 5F E1 40
000000F0 21 72 83 E3 65 48 AD F4 A3 87 39 D0 09 DF E4 B5
```

TODAYS LECTURE

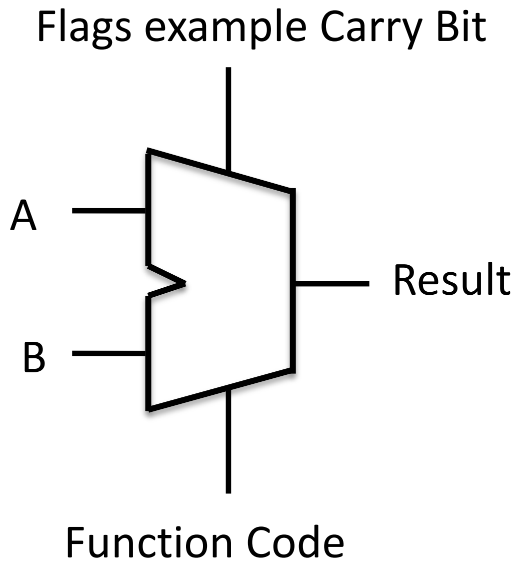
TODAYS LECTURE

- Introduce the Athematic Logic Unit (ALU)
- Combine components to build a simple machine.
- Introduce Instruction Set Architectures.
  - What is instruction set architecture?
- Begin discussing our Toy Instruction set architecture.

ARITHMETIC LOGIC UNIT



ALU SYMBOL AND INPUTS



TINY PROGRAM LANGUAGE

Let’s write a program that multiplies three numbers.

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

Now let’s design a processor that can run this program?

First need to convert this program into instruction that processor can execute.

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)

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

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

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

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

R0 = 3
R1 = 2
R2 = -1

We'll map variables to registers

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

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

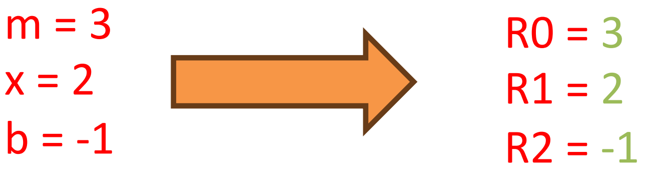
```
m = 3
x = 2
b = -1
```

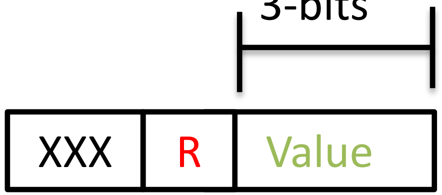
R0 = 3
R1 = 2
R2 = -1

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

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

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

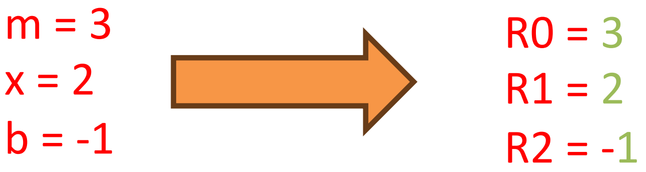


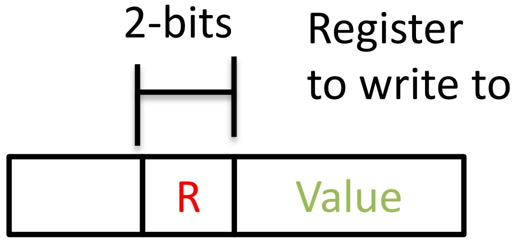


Store the value to write
example 3 =011
2 = 010
-1 = 111

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

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





State the register to write to

R0 = 00

R1 = 01

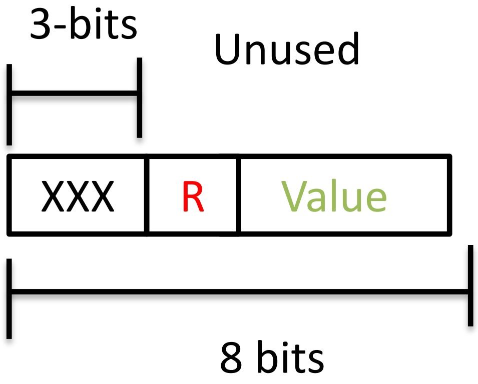
R2 = 10

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

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

m = 3
x = 2
b = -1

R0 =3
R1 = 2
R2 = -1



We just make these zeros
XXX = 000

NOW LET'S TRANSLATE OUT PROGRAM TO ONES AND ZERO

XXX	R	Value

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

m = 4	R0 = 3	000	00	011
x = 2	R1 = 2	000	01	010
b = -1	R2 = -1	000	10	111

NOW LET'S TRANSLATE OUT PROGRAM TO ONES AND ZERO

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

XXX	R	Value

m = 4	R0 = 3	000	00	011
x = 2	R1 = 2	000	01	010
b = -1	R2 = -1	000	10	111

NOW LET'S TRANSLATE OUT PROGRAM TO ONES AND ZERO

XXX	R	Value

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

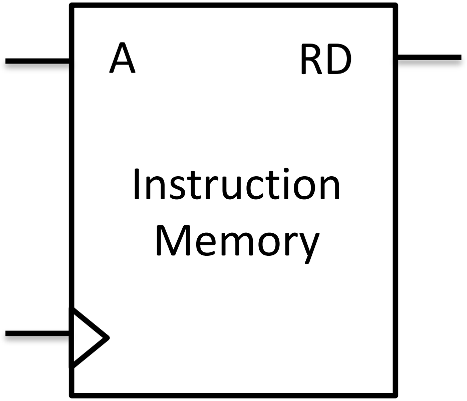
XXX	RA	Value

RA = Value

SO WHAT GET LOADED INTO MEMORY

Here is our program let's load it into memory

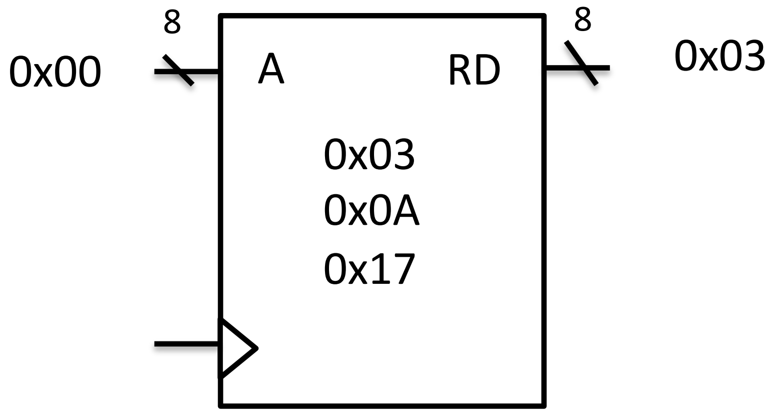
0x03 0x0A 0x17



SO WHAT GET LOADED INTO MEMORY

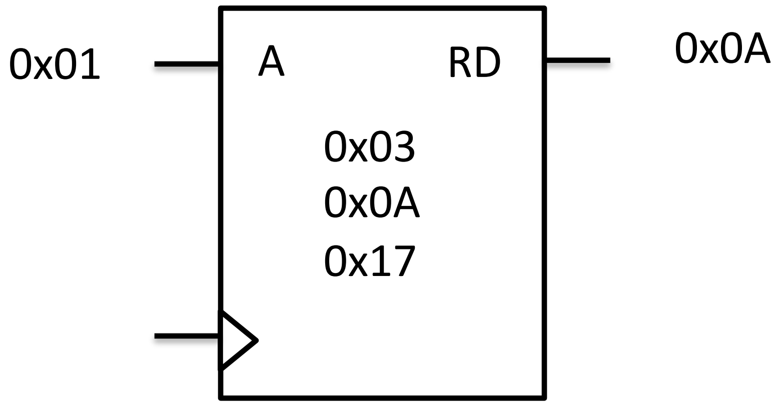
Here is our program let's load it into memory

Let's assume that Instruction Memory reads one byte at a time.



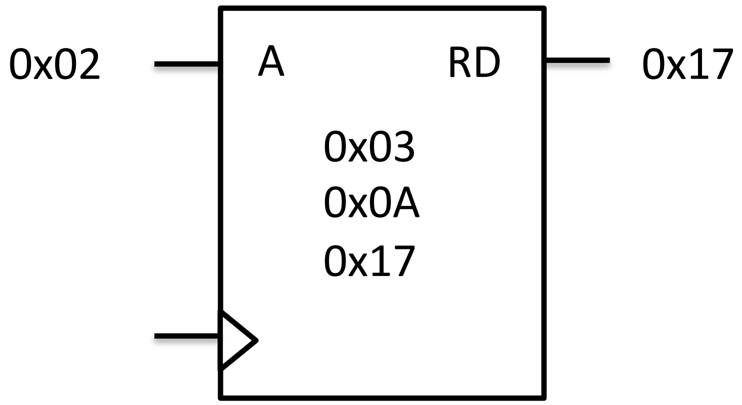
SO WHAT GET LOADED INTO MEMORY

Here is our program. let's load it into memory



SO WHAT GET LOADED INTO MEMORY

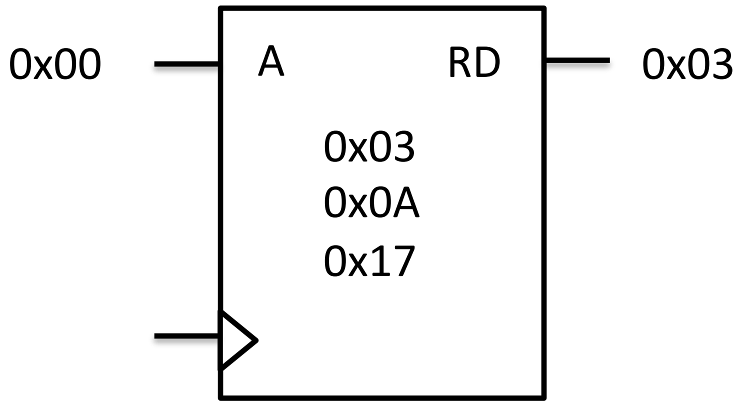
Here is our program. let's load it into memory



SO WHAT GETS LOADED INTO MEMORY

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

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

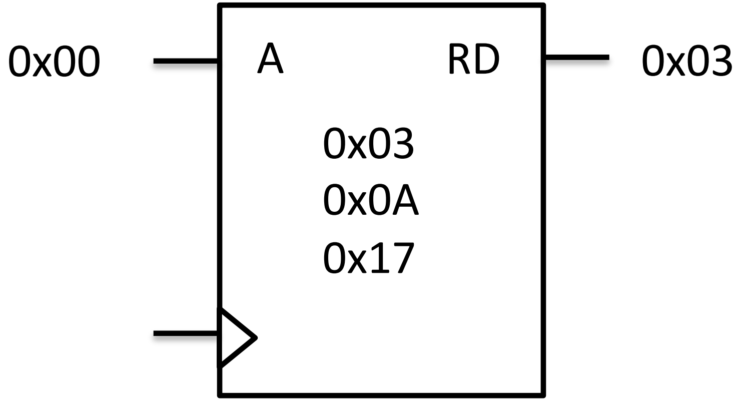


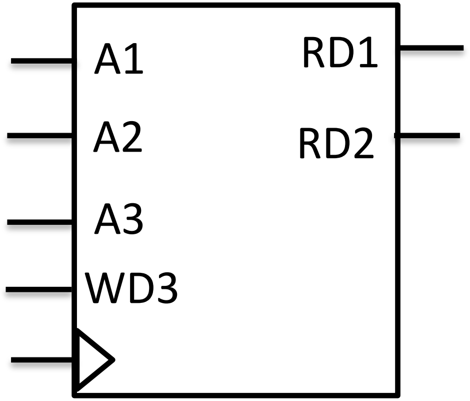
We still need to load our values into registers

LETS ADD OUR REGISTER FILE

m = 3
x = 2
b = -1

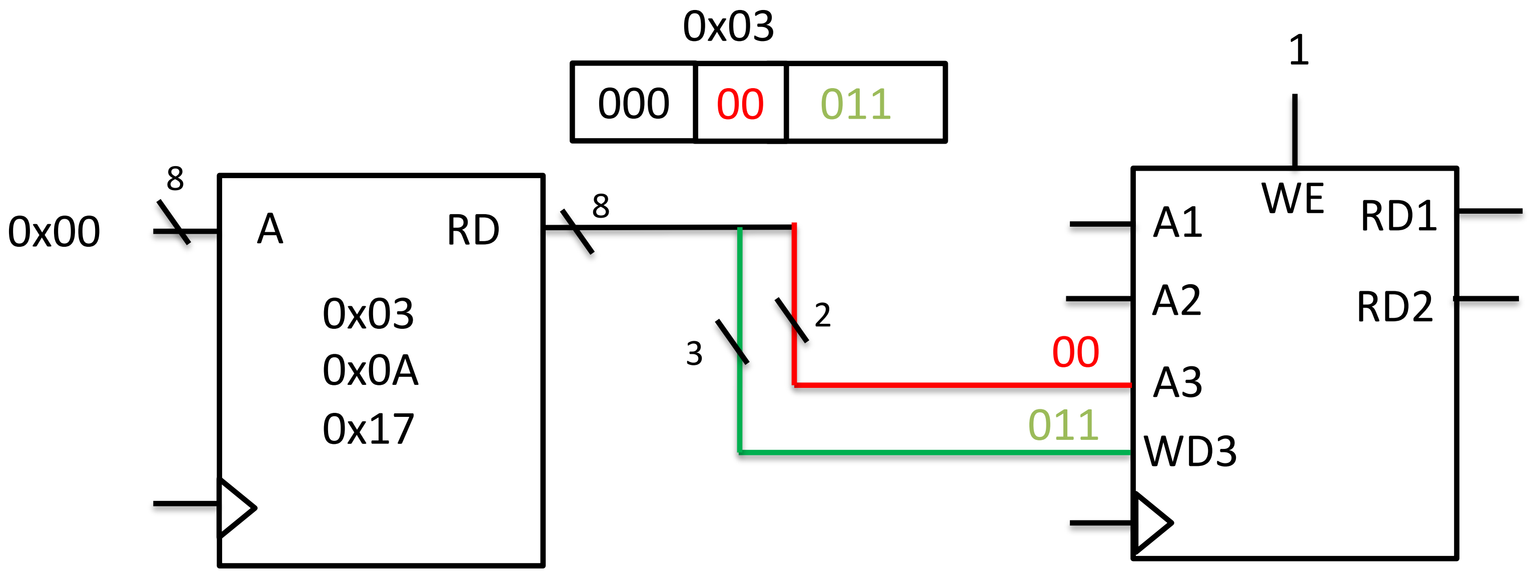
R0 =3
R1 = 2
R2 = -1





LETS ADD OUR REGISTER FILE

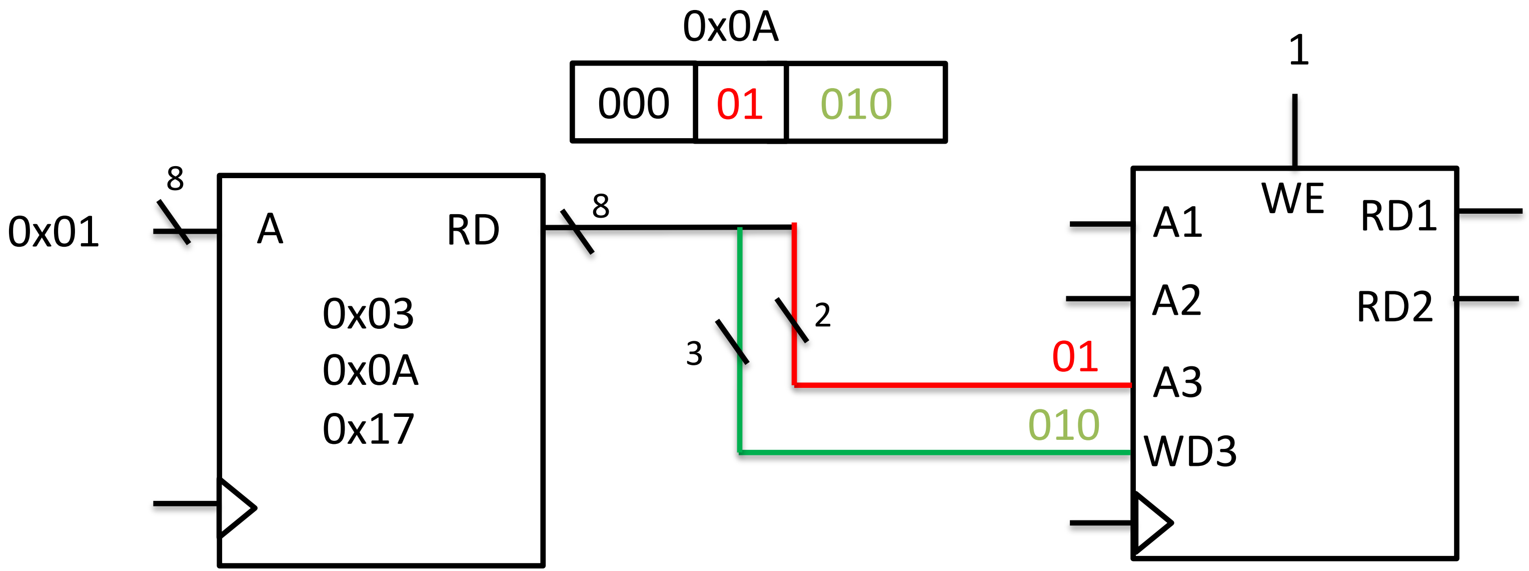
R0 =3
R1 = 2
R2 = -1



University of Virginia ENGINEERING

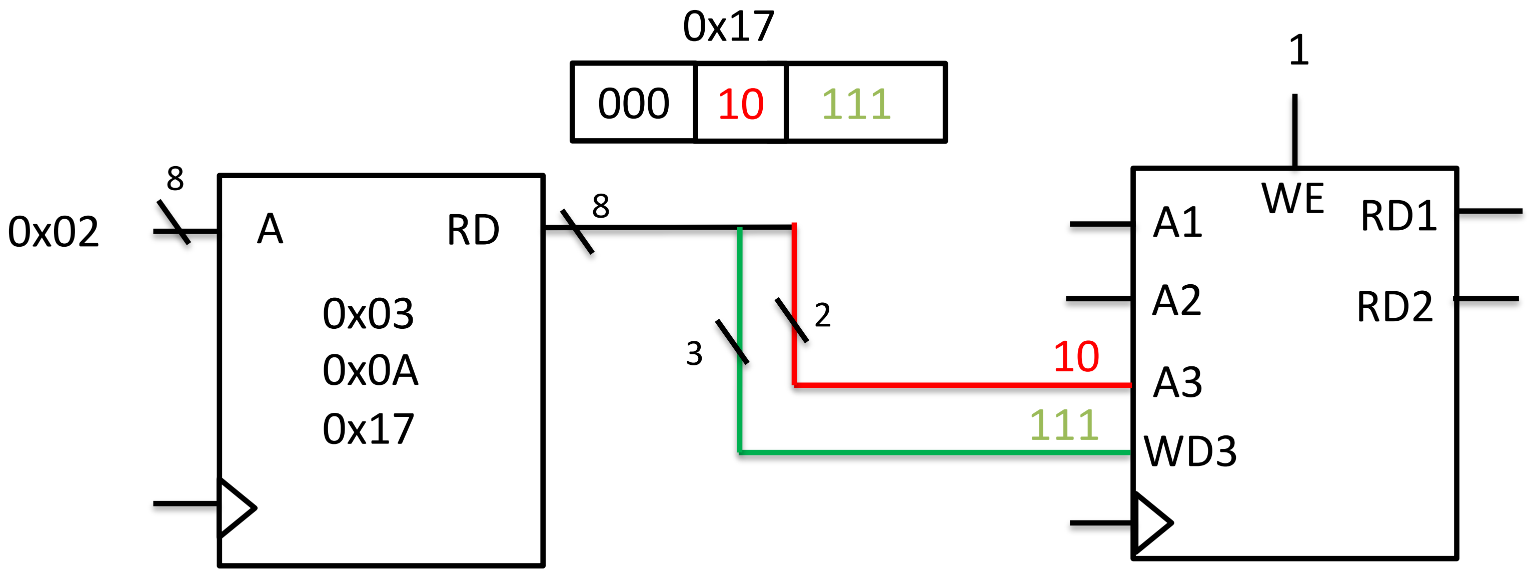
LETS ADD OUR REGISTER FILE

R0 =3
R1 = 2
R2 = -1



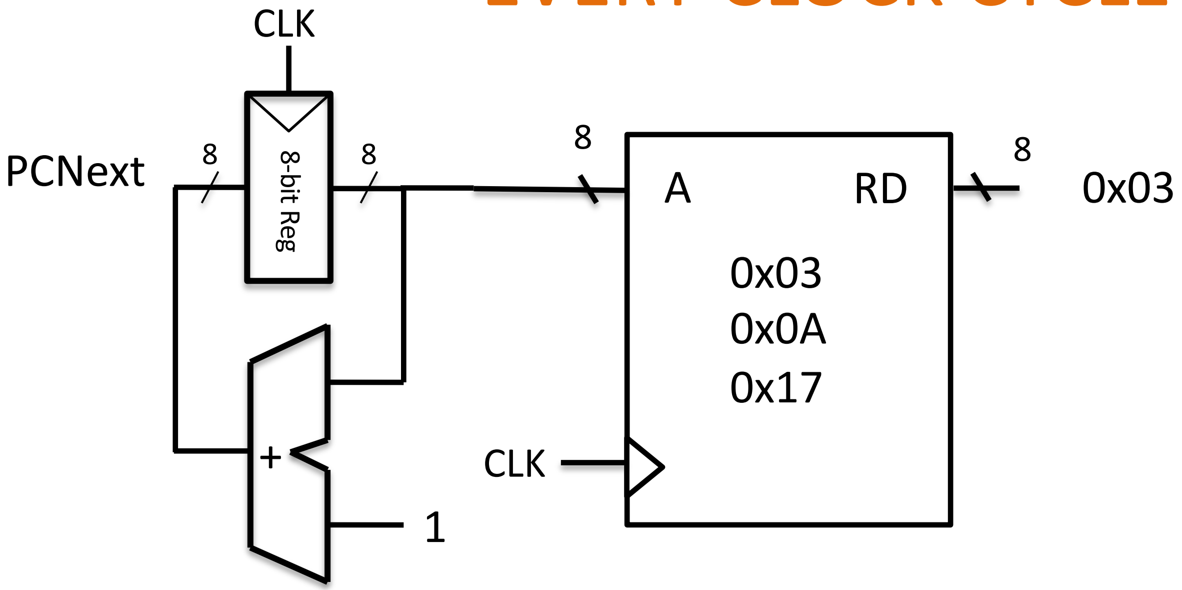
LETS ADD OUR REGISTER FILE

R0 =3
R1 = 2
R2 = -1



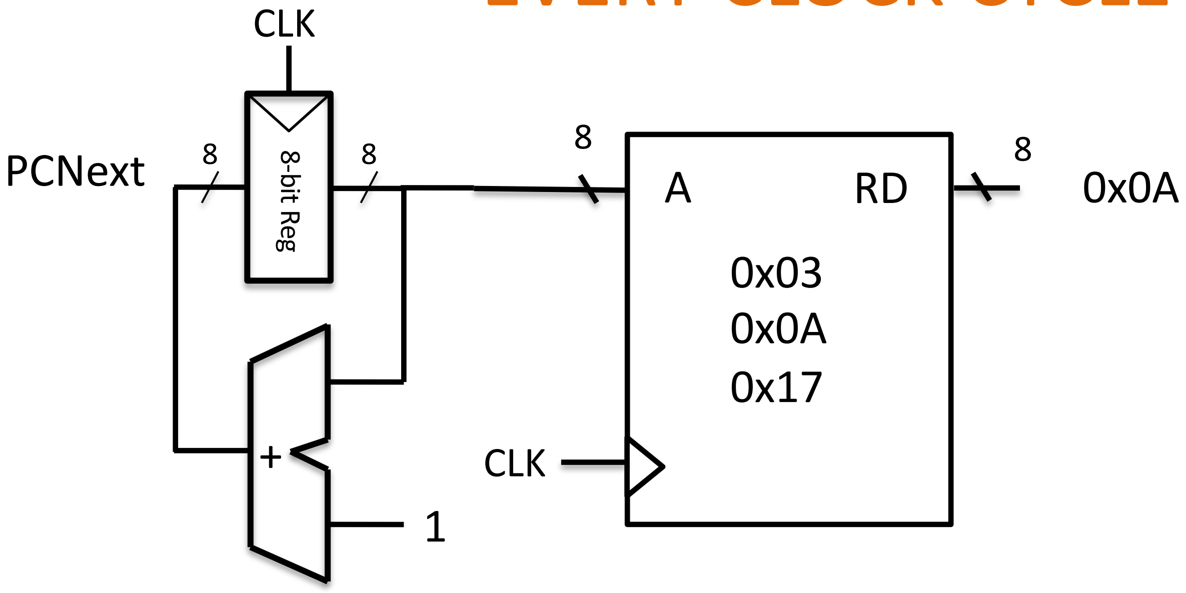
WE HAVE BEEN MANUALLY HOW AUTOMATICALLY CHANGE THE ADDRESS WITH EVERY CLOCK CYCLE

AUTOMATICALLY FETCH A NEW INSTRUCTION EVERY CLOCK CYCLE



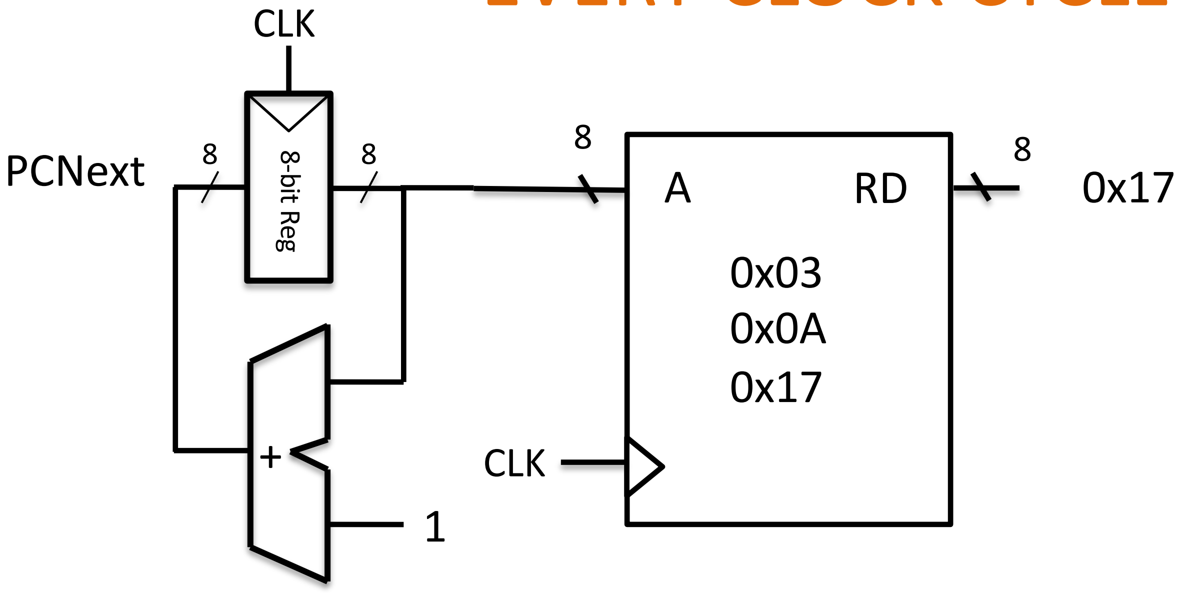
n-bit PC

AUTOMATICALLY FETCH A NEW INSTRUCTION EVERY CLOCK CYCLE



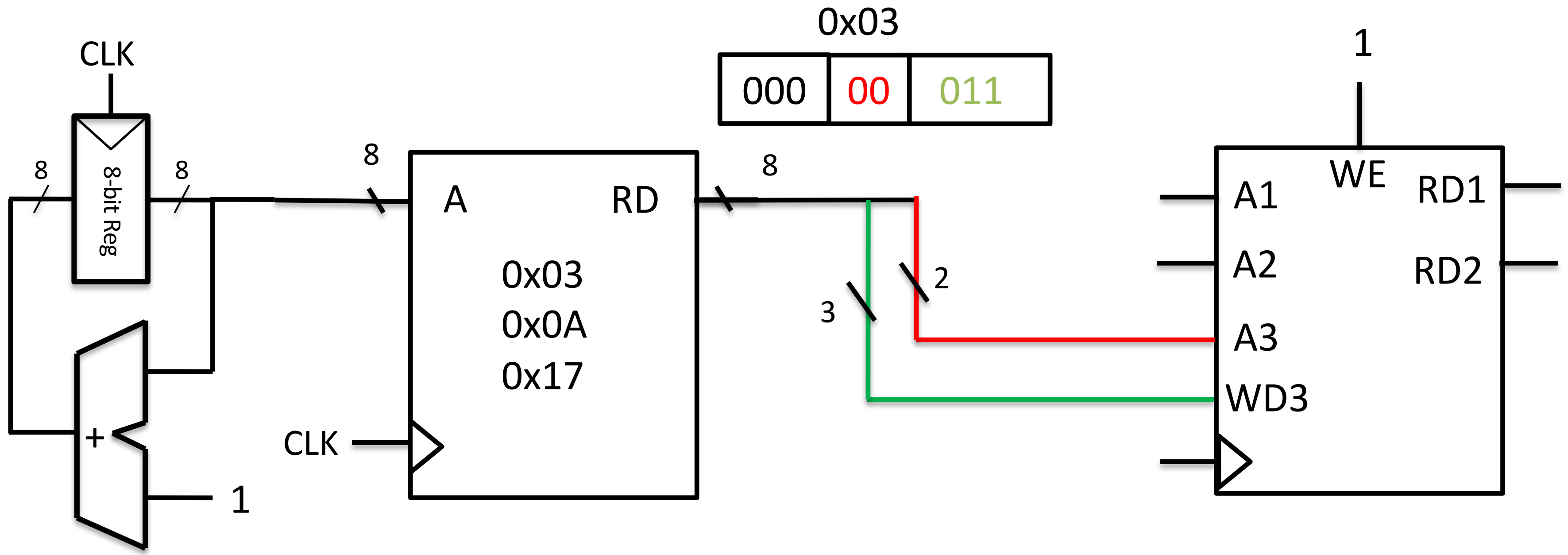
8-bit PC

AUTOMATICALLY FETCH A NEW INSTRUCTION EVERY CLOCK CYCLE

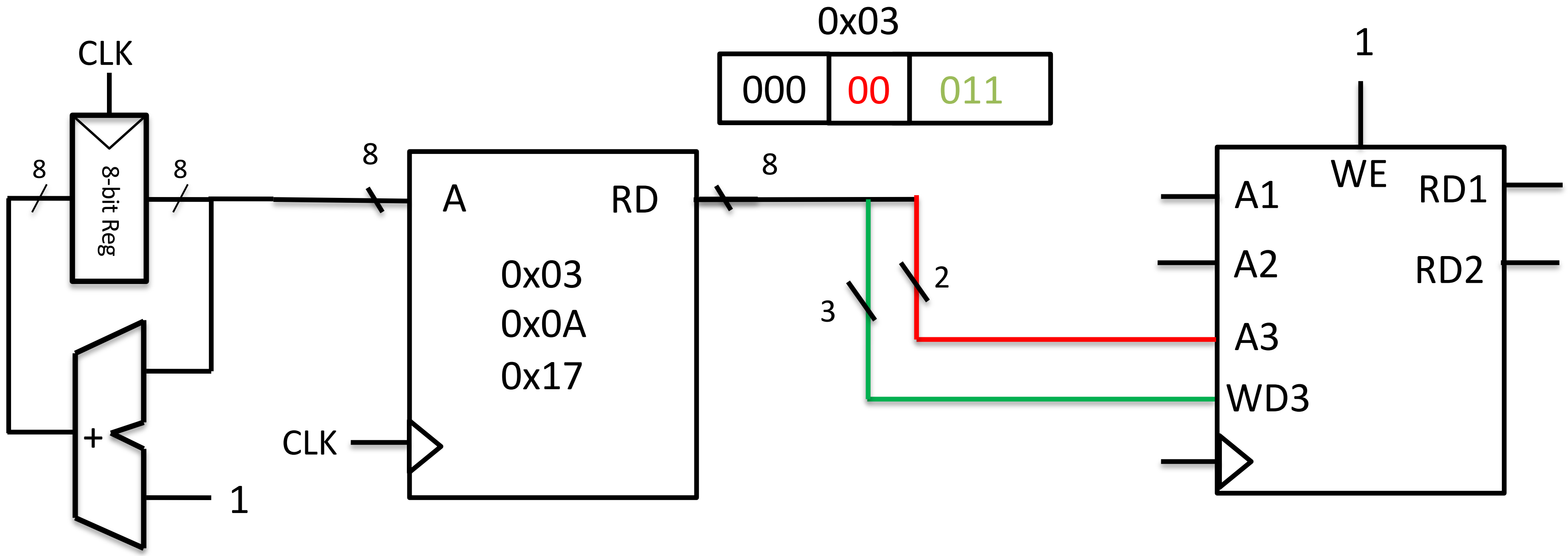


8-bit PC

NOW LET'S ADD OUR REGISTER FILE



8-bit PC



8-bit PC

Our program would have loaded values into the register file

R0 =3
R1 = 2
R2 = -1

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
```

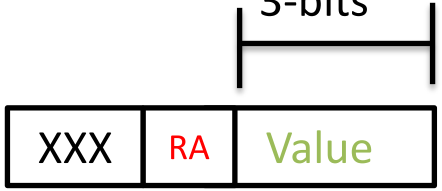
R0 *= R1 ← m = m * x
R0 *= R2 ← m = m * b

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

Multiply **Registers**

`y = m*x*b`

R0 *= R1
R0 *= R2



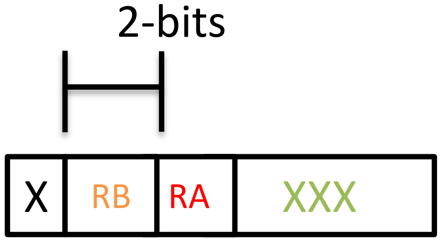
Don't real need the Value bits but we need another register so let's use the unused bits.

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?

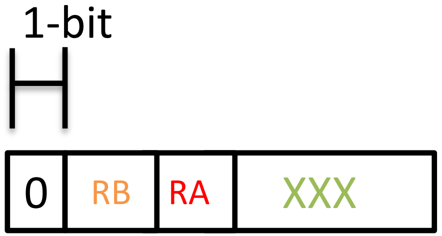
Need to be careful about which one is our destination register
Here the results get written to RA

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

0	RB	RA	XXX

Let's multiply value in **Registers**

y=m*x*b

R0 *= R1

R0 *= R2

0	01	00	000

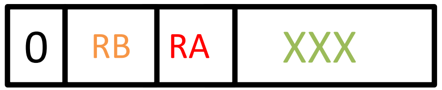
0x20

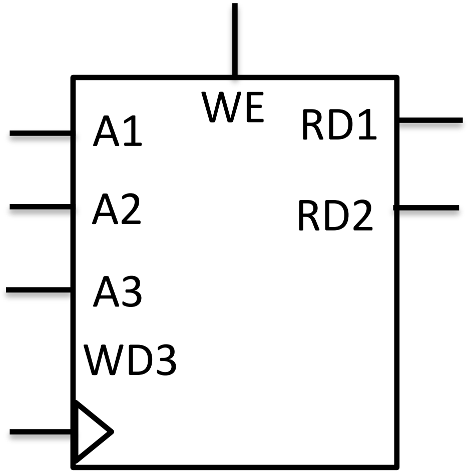
0	10	00	000

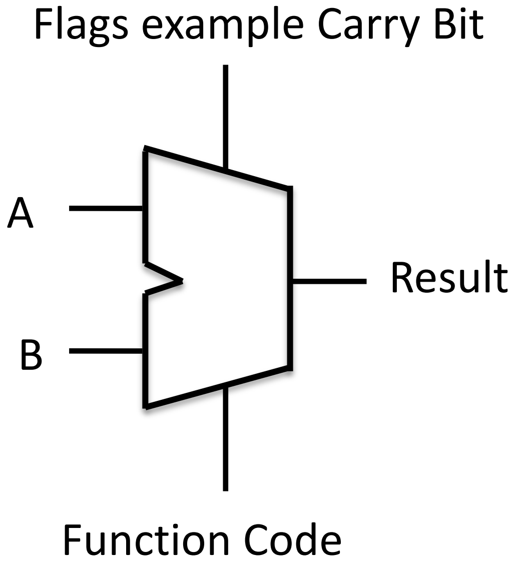
0x40

0x17

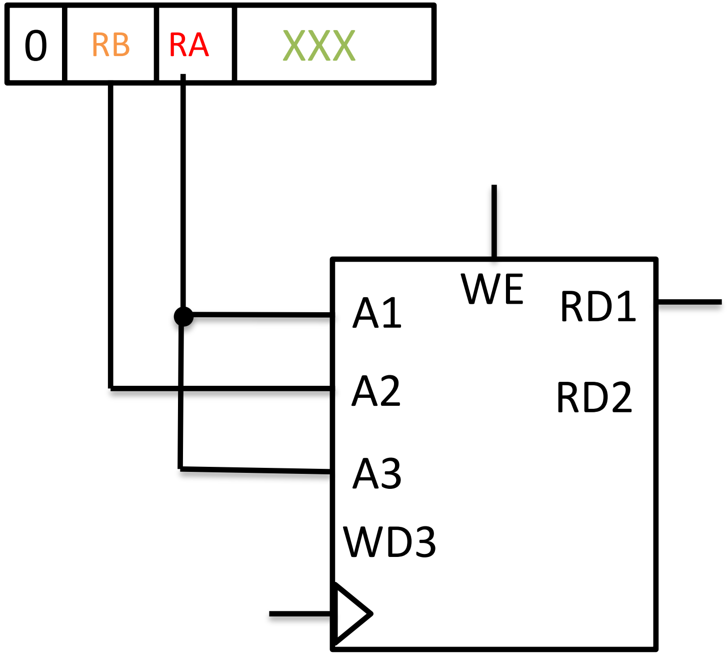
BUILDING MACHINE TO COMPUTE THIS



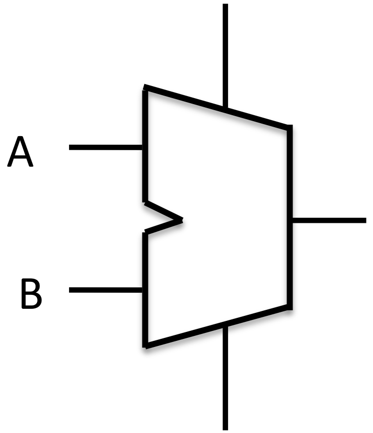




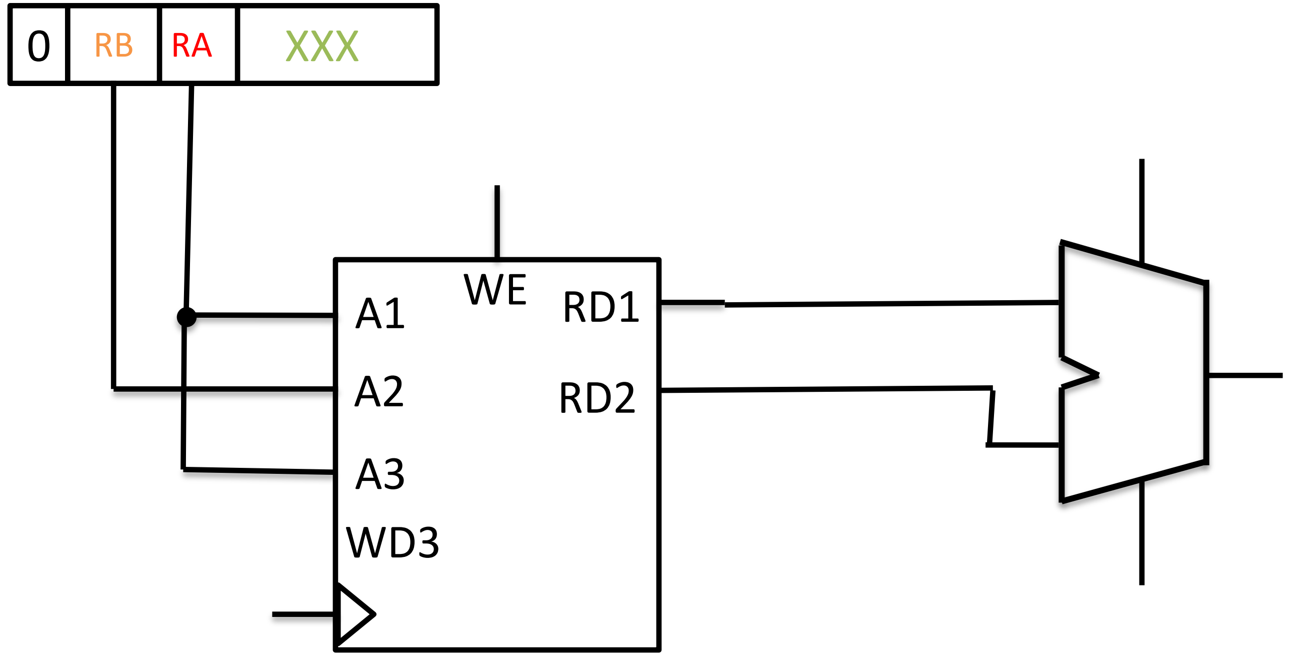
BUILDING MACHINE TO COMPUTE THIS



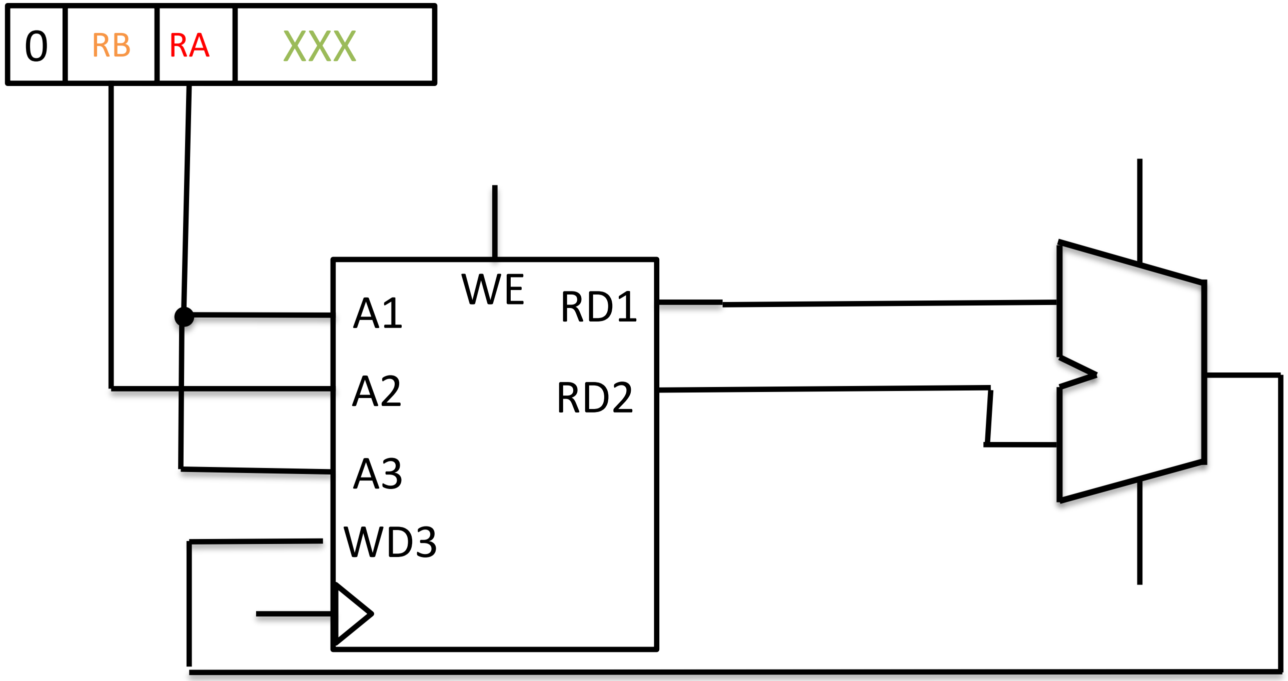
Flags example Carry Bit



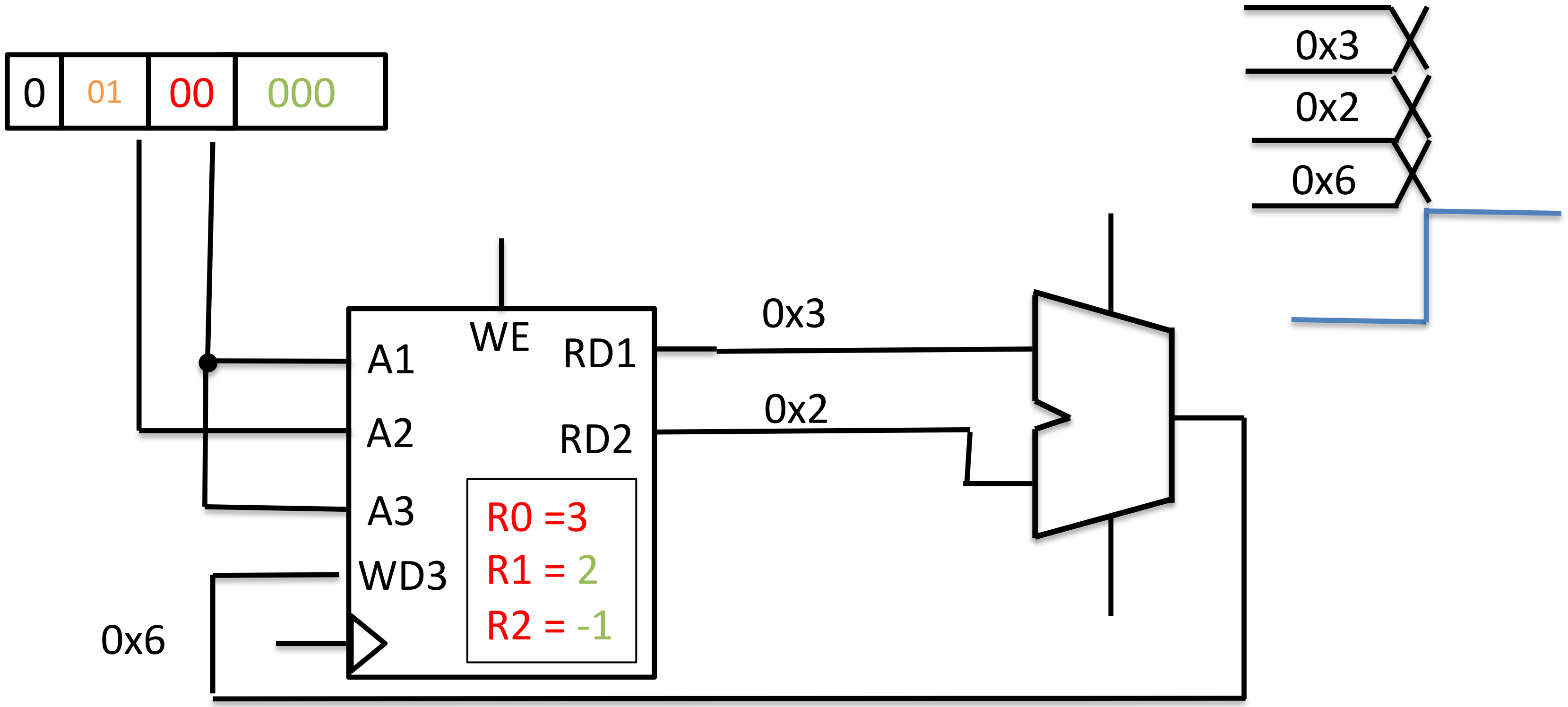
BUILDING MACHINE TO COMPUTE THIS



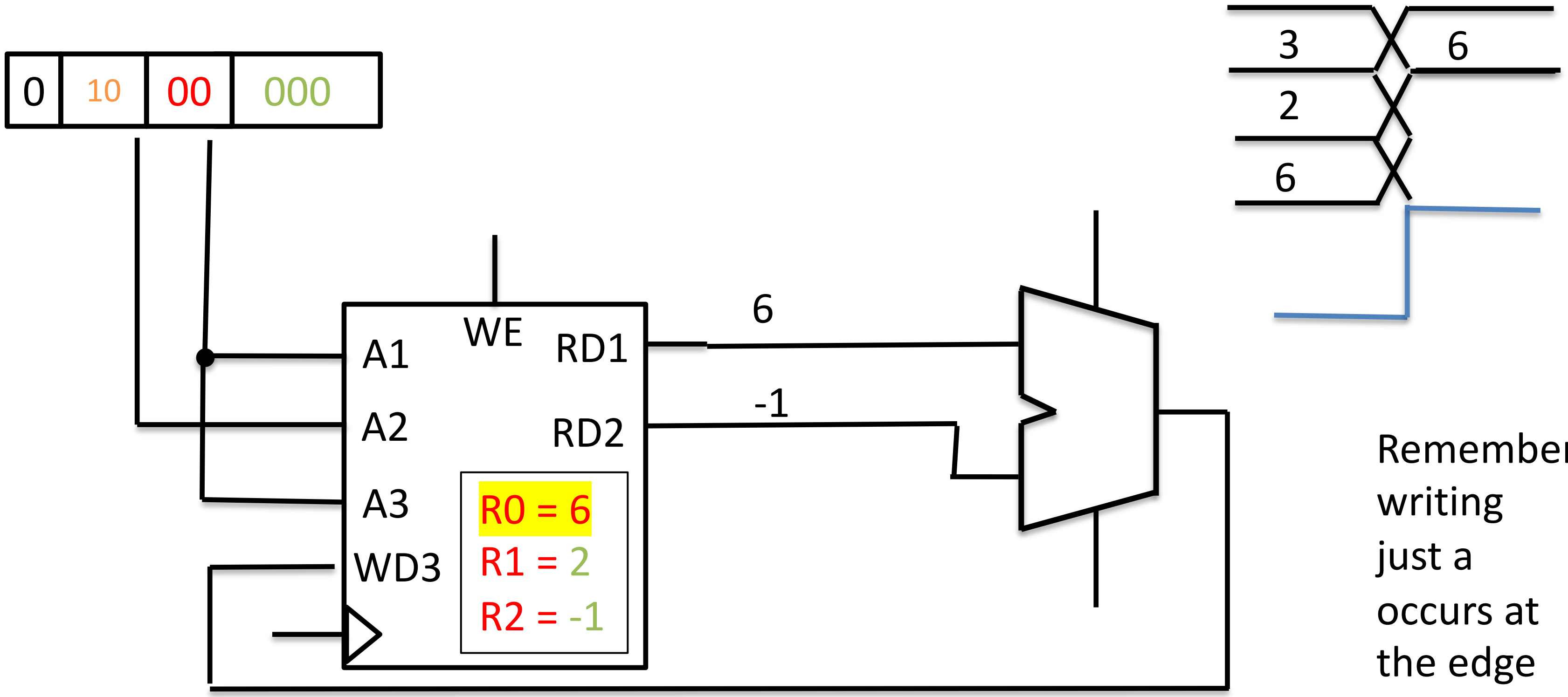
BUILDING MACHINE TO COMPUTE THIS



BUILDING MACHINE TO COMPUTE THIS

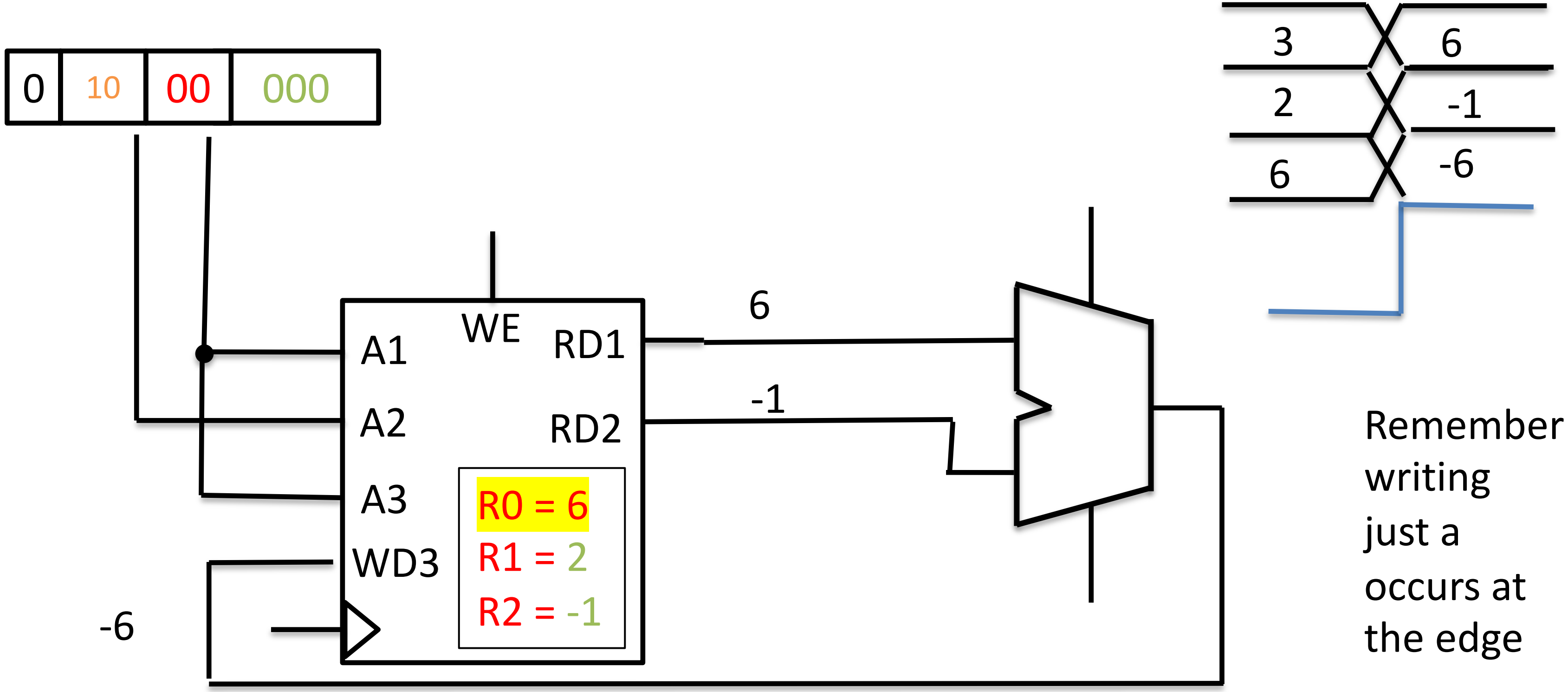


BUILDING MACHINE TO COMPUTE THIS



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 **<u>Registers</u>**

1	RB	RA	Value

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

INSTEAD GOING INSTRUCTION BY INSTRUCTION LET’S DESIGN THE ISA AND THE MACHINE

ENGINEERING