CS0 2130

Instruction Set Architecture

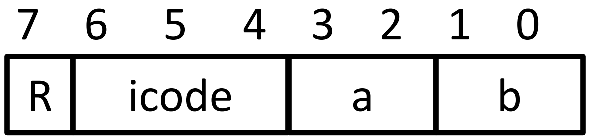
Daniel G. Graham PhD

UNIVERSITY of VIRGINIA | ENGINEERING

REVIEW

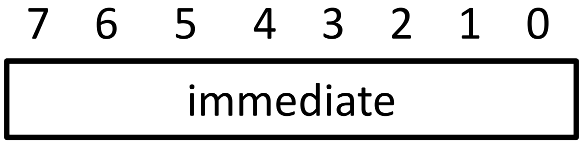
SUBSET OF OUR TOY ISA

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



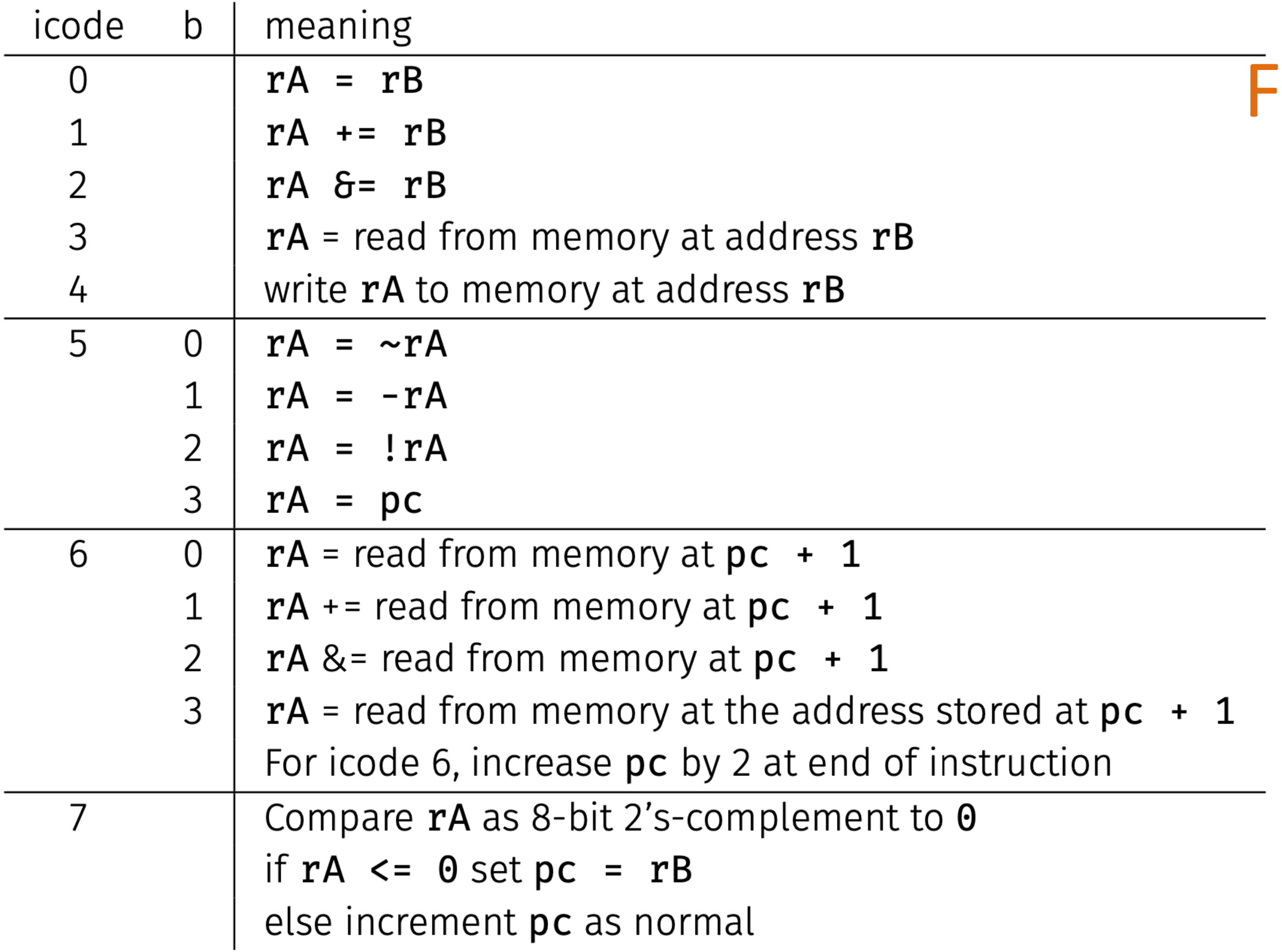
7 6 5 4 3 2 1 0

immediate

Contents

1. Full overview of Toy ISA
2. Some memory Operations with the Toy ISA
3. Loops and Conditionals with Toy ISA
4. Writing and simulating more complex programs with Toy ISA

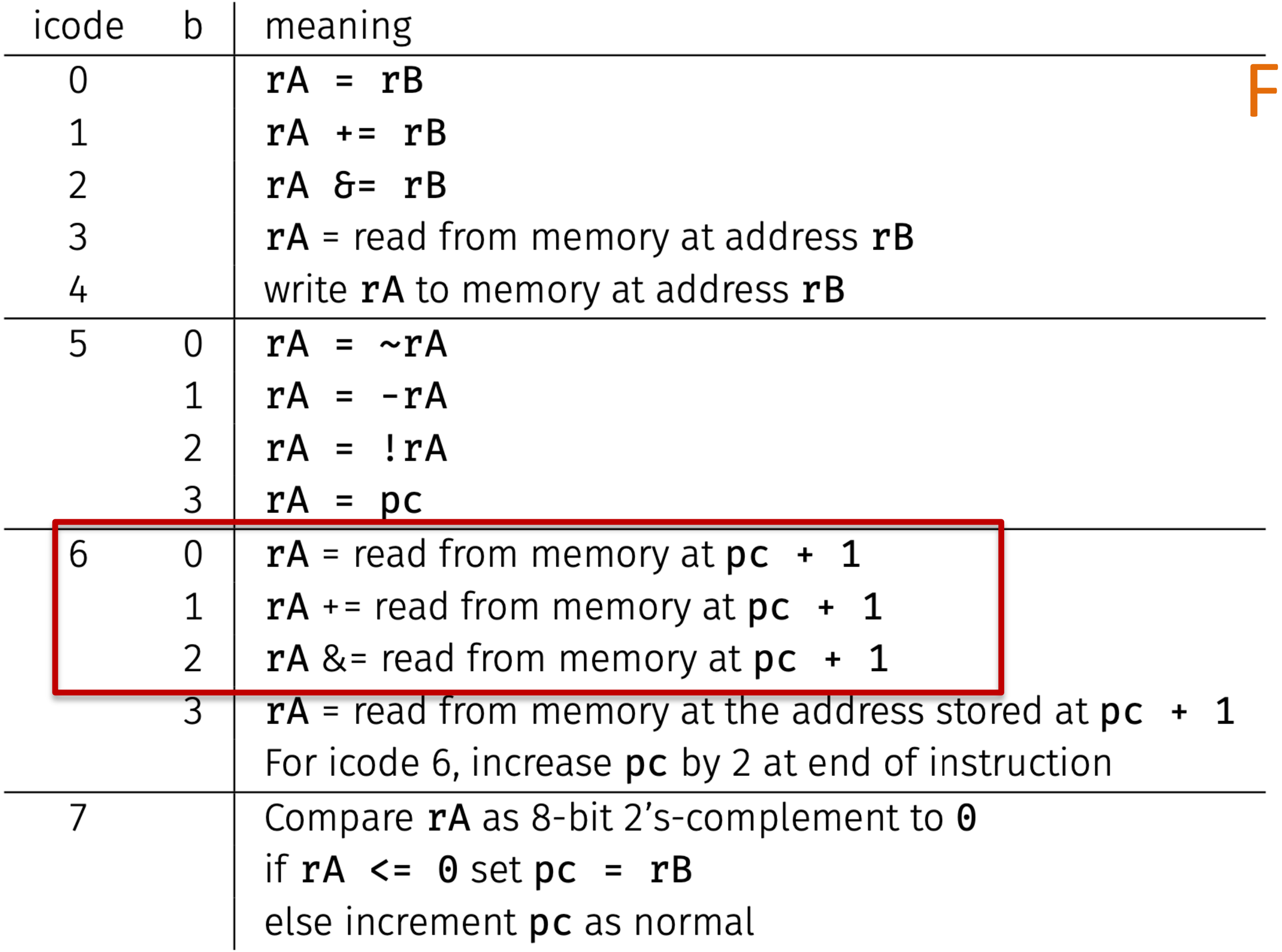
FULL ISA



icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
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

We'll give the full description of ISA at the begin of every exam. In fact this a picture of what we will give you.

FULL ISA



icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		**`rA`** = read from memory at address **`rB`**
4		write **`rA`** to memory at address **`rB`**
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
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

More operations with immediates

MEMORY OPERATIONS

icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`

These instructions are a little tricky. So, let's spend some time on them.

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

7	6	5	4	3	2	1	0
immediate							

READ FROM MEMORY ADDRESS STORED IN RB

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

What are the values of R0 and R1. Once program completes?

Registers

R0	X
R1	X
R2	X
R3	X

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

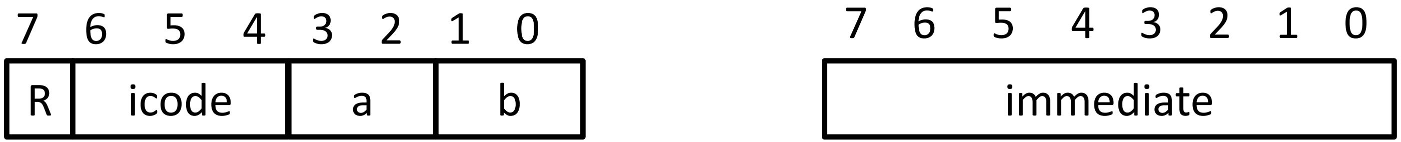
6	0	**rA** = read from memory at **pc + 1**
	1	**rA** += read from memory at **pc + 1**
	2	**rA** &= read from memory at **pc + 1**
	3	**rA** = read from memory at the address stored at **pc + 1**
		For icode 6, increase **pc** by 2 at end of instruction

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

6	0	`rA` = read from memory at `pc + 1`



Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

6	0	**rA** = read from memory at **pc + 1**

7	6 5 4	3 2	1 0
0	1 1 0	01	00

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

Registers

R0	X
R1	23
R2	X
R3	X

PC	02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

6	0	**rA** = read from memory at **pc + 1**

7	6 5 4	3 2	1 0
0	1 1 0	01	00

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

PC Updates to 2 so what instruction will we execute next?

Contents

1. Full overview of Toy ISA
2. Some memory Operations with the Toy ISA
3. Loops and Conditionals with Toy ISA
4. Writing and simulating more complex programs with Toy ISA

Registers

R0	X
R1	23
R2	X
R3	X
PC	02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		**rA** = read from memory at address **rB**
4		write **rA** to memory at address **rB**

Registers

R0	X
R1	23
R2	X
R3	X
PC	02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

3 | `rA` = read from memory at address `rB`

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

7 6 5 4 3 2 1 0
immediate

Registers

R0	X
R1	23
R2	X
R3	X
PC	02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

3	**rA** = read from memory at address **rB**

7	6	5	4	3	2	1	0
0	0 1 1			00		01	

Registers

R0	FF
R1	23
R2	X
R3	X
PC	03

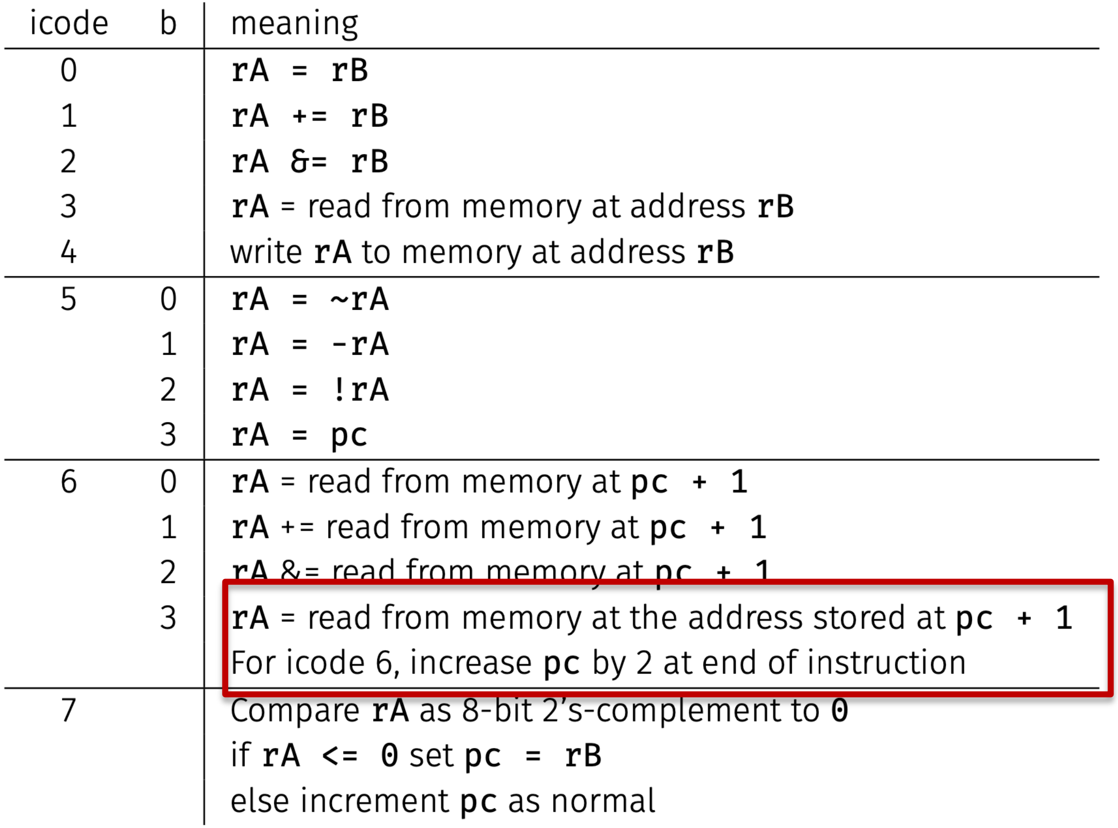
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	23	31													
10																
20				FF												
30																

3	**rA** = read from memory at address **rB**

7	6	5	4	3	2	1	0
0	0 1 1			00		01	

RB is R1 and it stores 0x23.
So, we go location 23 in memory and retrieve the value 0xFF.

STOP. And talk to you neighbor



icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
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

Let's look at this instruction now?

READ FROM MEMORY ADDRESS STORED IN RB

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	37														
10																
20				FF												
30								BB								

What are the values of R0. Once program completes?

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	37														
10																
20				FF												
30								BB								

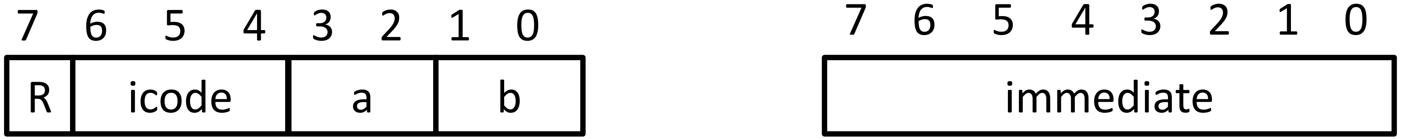
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` += read from memory at `pc + 1`
	2	`rA` &= read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`
		For icode 6, increase `pc` by 2 at end of instruction

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	37														
10																
20				FF												
30								BB								

3	**rA** = read from memory at the address stored at **pc + 1** For icode 6, increase **pc** by 2 at end of instruction


7 6 5 4 3 2 1 0

R | icode | a | b

7 6 5 4 3 2 1 0

immediate

Registers

R0	BB
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	37														
10																
20				FF												
30								BB								

3	**`rA`** = read from memory at the address stored at **`pc + 1`** For icode 6, increase **`pc`** by 2 at end of instruction

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

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

MEMORY WRITES WORK IN A SIMILAR WAY

REGISTER SPILLING

Because we have a limited number of registers, we can’t store all variables in registers, so we must store some in memory and read them into a register when we need them.

Here is the strategy

1. Read the register value to a predetermined location in memory.
2. Use the register
3. Write the register value back to memory, so that it can be used to store something else

Architecture	8 bit	32 bit	64 bit
ARM	X	15	31
Intel x86	X	8	16
Toy ISA	4	X	X

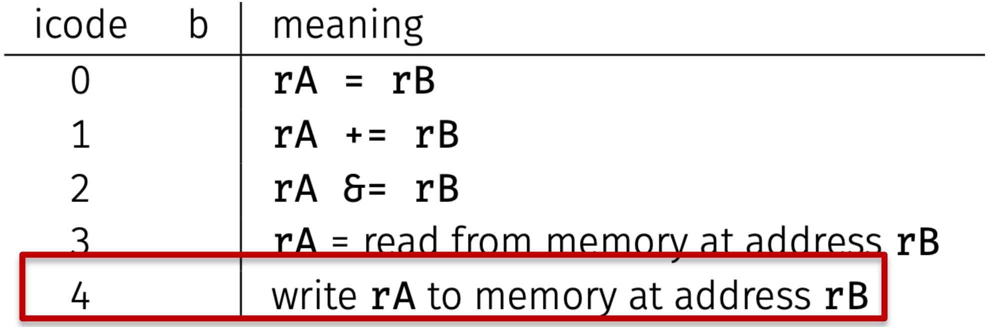
REGISTER SPILLING

```
R0 = M[0x31]
R0 += 2
R1 = M[0x31]
M[R1] = R0
```

After this point R0 can be used for something else

REGISTER SPILLING

```
R0  = M[0x31]
R0 += 2
R1  = M[0x31]
M[R1] = R0
```



icode	b	meaning
0		**rA = rB**
1		**rA += rB**
2		**rA &= rB**
3		**rA** = read from memory at address **rB**
4		write **rA** to memory at address **rB**

M[RB] = RA

```
R0  = M[0x31]        0x63 0x31
R0  += 2             0x61 0x02
R1  = M[0x31]        0x64 0x31
M[R1] = R0           0x41
```

Registers

R0	X
R1	X
R2	X
R3	X
PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	31	61	02	64	31	41									
10																
20																
30		02														

What are the values of R0 and R1. Once program completes?

```
R0  = M[0x31]        0x63 0x31
R0  += 2             0x61 0x02
R1  = M[0x31]        0x64 0x31
M[R1] = R0           0x41
```

Registers

R0	02
R1	X
R2	X
R3	X

PC	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	31	61	02	64	31	41									
10																
20																
30		02														

What are the values of R0 and R1. Once program completes?

```
R0 = M[0x31]        0x63 0x31
R0 += 2             0x61 0x02
R1 = M[0x31]        0x64 0x31
M[R1] = R0          0x41
```

Registers

R0	04
R1	X
R2	X
R3	X
PC	02

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	31	61	02	64	31	41									
10																
20																
30		02														

What are the values of R0 and R1. Once program completes?

```
R0  = M[0x31]        0x63 0x31
R0  += 2             0x61 0x02
R1  = M[0x31]        0x64 0x31
M[R1] = R0           0x41
```

Registers

R0	04
R1	31
R2	X
R3	X

PC	04

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	31	61	02	64	31	41									
10																
20																
30		02														

What are the values of R0 and R1. Once program completes?

```
R0 = M[0x31]      0x63 0x31
R0 += 2           0x61 0x02
R1 = M[0x31]      0x64 0x31
M[R1] = R0        0x41
```

Registers

R0	04
R1	31
R2	X
R3	X

PC	06

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	31	61	02	64	31	41									
10																
20																
30		04														

What are the values of R0 and R1. Once program completes?

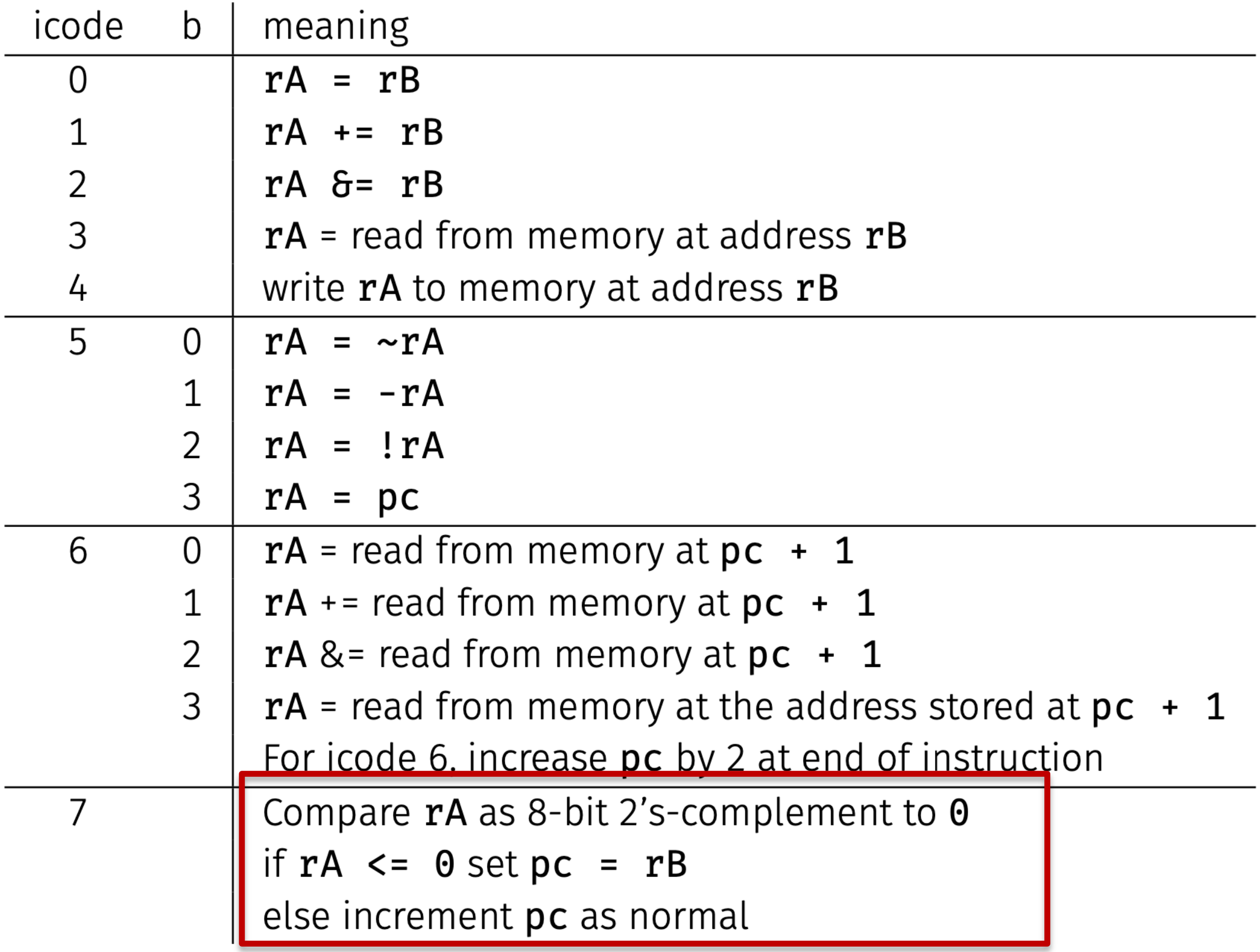
CONDITIONAL IF ELSE

```
x = M[0x0F]
If  x > 0:
    x += 1
Else:
    x &= 7
```

Memory Map IO (Input/output)

Let’s implement this program using our instructions

FULL ISA



icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
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

LET'S ALLOCATE REGISTERS AND PICK INSTRUCTIONS

```
x = M[0x0F]
If  x > 0:
   x += 1
Else:
   x &= 7
```

```
R0 = M[0x20]
R1 =
If R0 <= 0 set PC= R1
R0 += 1
R0 &= 2
```

Let's leave blank for now

LET’S CALCULATE WHERE TO JUMP TO

Memory Address		Size of Instruction
0x00	`R0 = M[0x20]`	2 Bytes
0x02	`R1 =`	2 Bytes
0x04	`If R0 <= 0 set PC= R1`	1 Byte
0x05	`R0 += 1`	2 Bytes
0x07	`R0 &= 2`	2 Bytes

So what address do we want R1 to be?

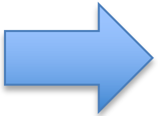
LET’S CALCULATE WHERE TO JUMP TO

Memory Address		Size of Instruction
0x00	`R0 = M[0x20]`	2 Bytes
0x02	`R1 = 0x07`	2 Bytes
0x04	`If R0 <= 0 set PC= R1`	1 Byte
0x05	`R0 += 1`	2 Bytes
0x07	`R0 &= 2`	2 Bytes

So what address do we want R1 to be?

LET’S CALCULATE WHERE TO JUMP TO

```
x = M[0x0F]
If  x > 0:
    x += 1
Else:
    x &= 7
```



Memory Address	
0x00	`R0 = M[0x20]`
0x02	`R1 = 0x07`
0x04	`If R0 <= 0 set PC= R1`
0x05	`R0 += 1`
0x07	`R0 &= 2`

So what address do we want R1 to be?

LOOPS

icode	b	meaning
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
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

WRITE A LOOP

First, rewrite as a do-while loop. (This due to limitation in Toy ISA) reasons will be clear later.

```
x = 2
for (i = 0; i < 5; i++){
    x+=1
}
```

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
```

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
```

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
R2 = PC
```

Store the memory address of the beginning of the loop

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
R2 = PC
R0 += 1
```

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
R2 = PC
R0 += 1
R1 += 1
```

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
R2 = PC
R0 += 1
R1 += 1
R3 = R1
R3+= -5
if R3 <=0 then PC = R2
```

But wait is that correct?

SEE IF YOU CAN ENCODE THIS AND RUN IT IN THE SIMULATOR

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2
R1 = 0
R2 = PC
R0 += 1
R1 += 1
R3 = R1
R3+= -5
if R3 <=0 then PC = R2
```

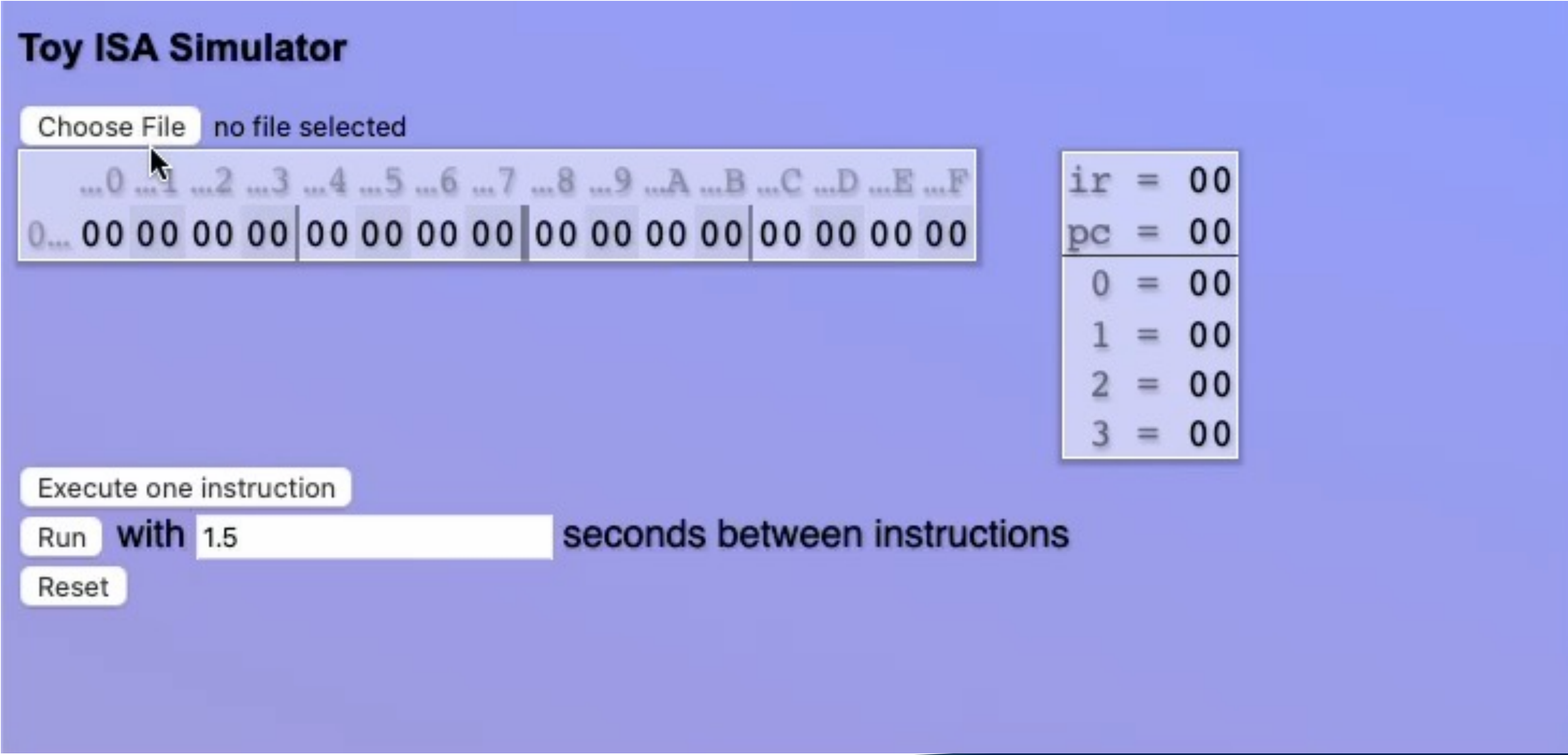
But wait is that correct? Translating the condition can be tricky

WRITE A LOOP

```
x = 2
i = 0
do{
    x+=1
    i++
}while(i<5)
```

```
R0 = 2                        0x60 02
R1 = 0                        0x64 0x00
R2 = PC                       0x5B
R0 += 1                       0x61 0x01
R1 += 1                       0x65 0x01
R3 = R1                        0x0D
R3+= -4                        0x6D 0xFC
if R3 <=0 then PC = R2               0x7E
```

-3 , -2, -1, 0, 1 (five times)



Toy ISA Simulator

no file selected

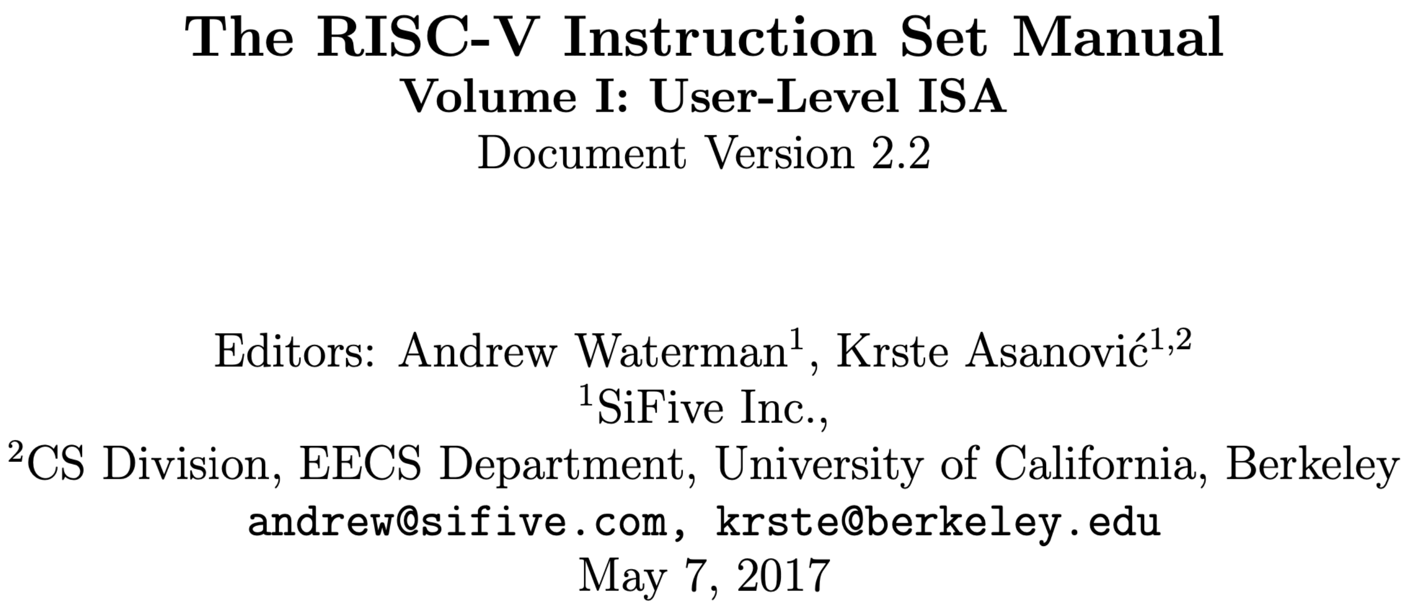
	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

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

Run with 1.5 seconds between instructions

FROM TOY ISA TO RISC-V

SOME PERPECTIVE (RISC-V)



The RISC-V Instruction Set Manual

Volume I: User-Level ISA

Document Version 2.2

Editors: Andrew Waterman[1], Krste Asanović[1,2]

[1]SiFive Inc.,

[2]CS Division, EECS Department, University of California, Berkeley

andrew@sifive.com, krste@berkeley.edu

May 7, 2017

Available at: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]				rd	opcode	U-type

R-Format: instructions using 3 register inputs

I-Format: instructions with immediates, loads

S-Format: store instruction

U-Format: instructions with upper immediates

Detailed Data Sheet: https://www.elsevier.com/__data/assets/pdf_file/0011/297533/RISC-V-Reference-Data.pdf

RISC VS CISC

RISC-V ADD

https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#addi

X86 Add

https://www.felixcloutier.com/x86/add