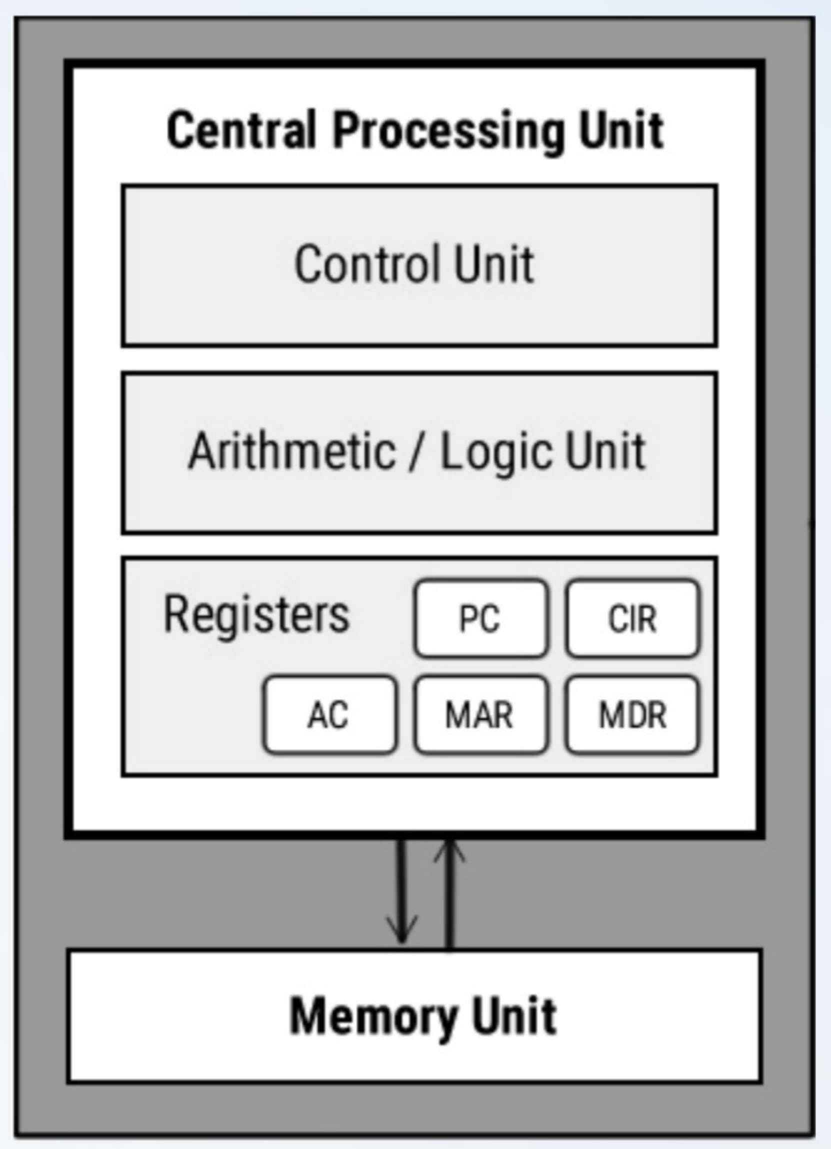
Assembly Programming with

BLOCK

SEMBLER

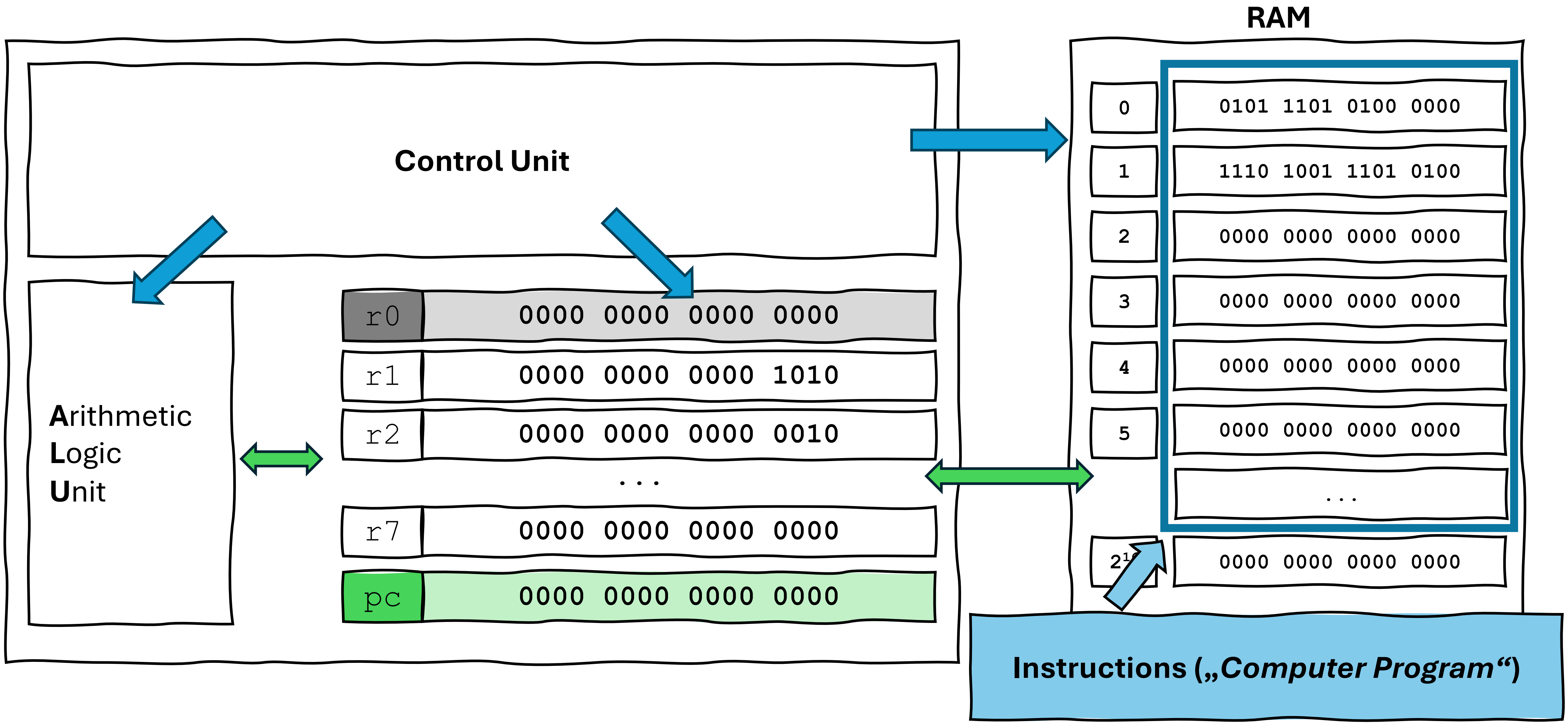
Recap


Central Processing Unit
Control Unit
Arithmetic / Logic Unit
Registers
PC
CIR
AC
MAR
MDR
Memory Unit

ANNA (A New Noncomplex Architecture)

- 16-bit Architecture
- 8 Registers (`r0`, …, `r7`), each storing a 16 bit value
- 16 Instructions
- 2^{16} x Words of RAM

ANNA Architecture (Overview)



r0	0000 0000 0000 0000
r1	0000 0000 0000 1010
r2	0000 0000 0000 0010
...	
r7	0000 0000 0000 0000
pc	0000 0000 0000 0000

0	0101 1101 0100 0000
1	1110 1001 1101 0100
2	0000 0000 0000 0000
3	0000 0000 0000 0000
4	0000 0000 0000 0000
5	0000 0000 0000 0000
	...
2^{16}	0000 0000 0000 0000

ADD Instruction

add r7 r1 r2

Mnemonic

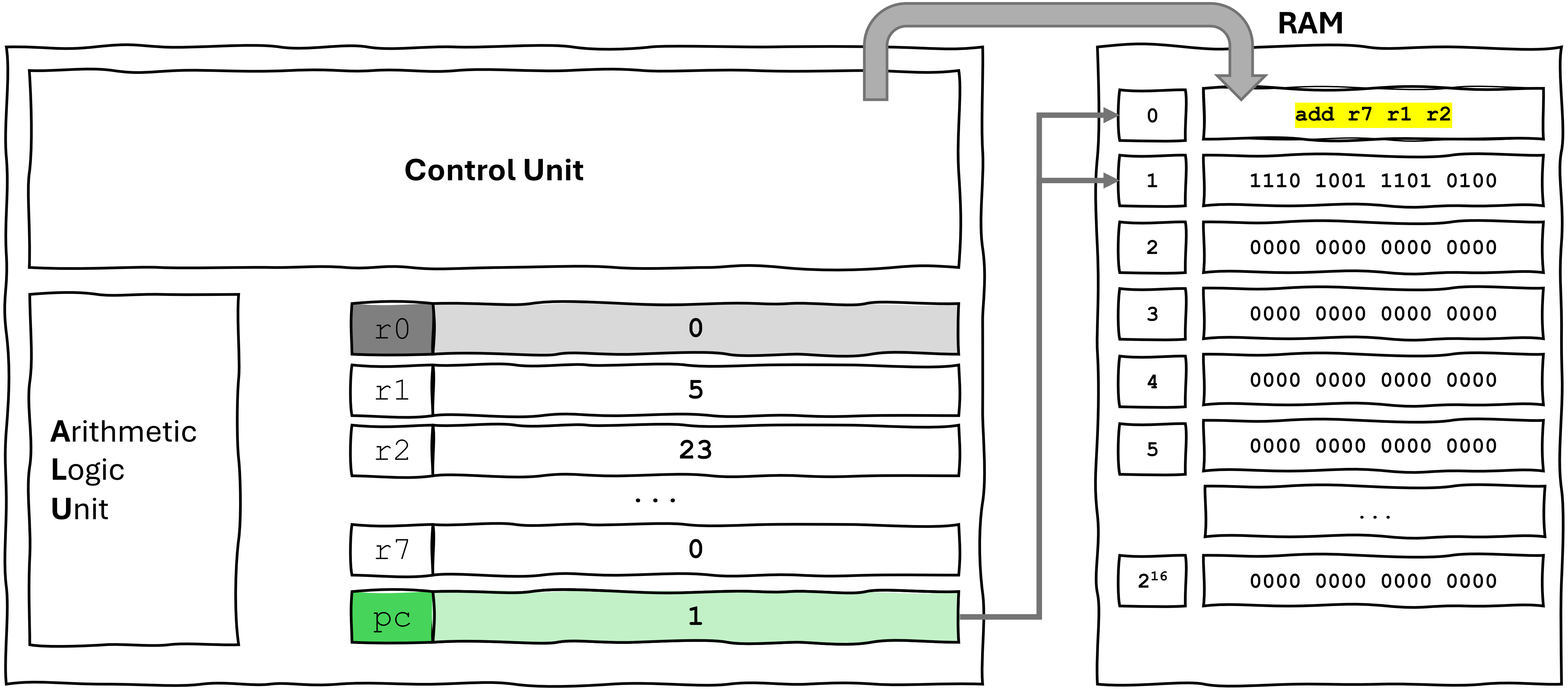
Destination
Rd

Source
Rs1, Rs2

→ r7 = r1 + r2

0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	0
15			12	11		9	8		6	5		3	2		0
Opcode				Rd			Rs_1			Rs_2			Unused		

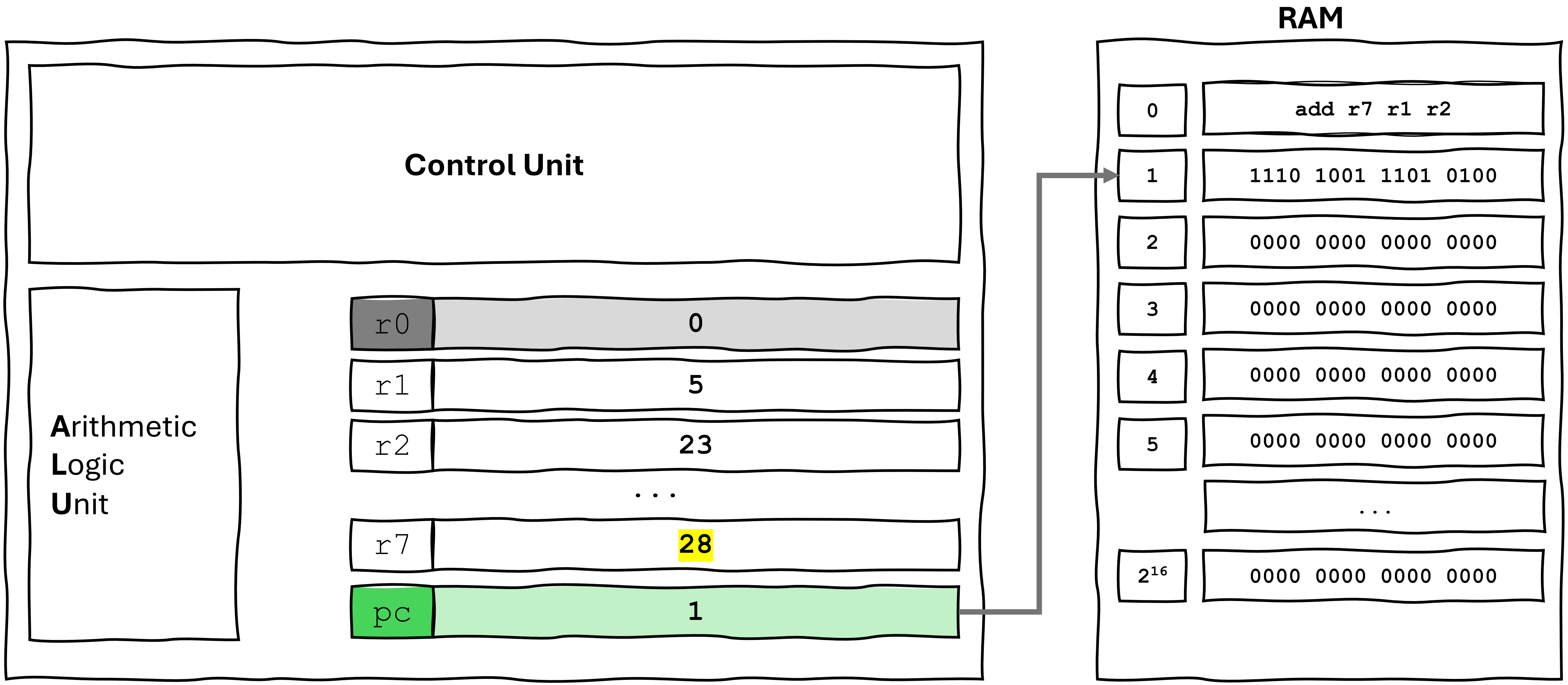
ANNA Architecture (Overview)



r0	0
r1	5
r2	23
...	
r7	0
pc	1

0	add r7 r1 r2
1	1110 1001 1101 0100
2	0000 0000 0000 0000
3	0000 0000 0000 0000
4	0000 0000 0000 0000
5	0000 0000 0000 0000
	...
2^{16}	0000 0000 0000 0000

ANNA Architecture (Overview)



Control Unit

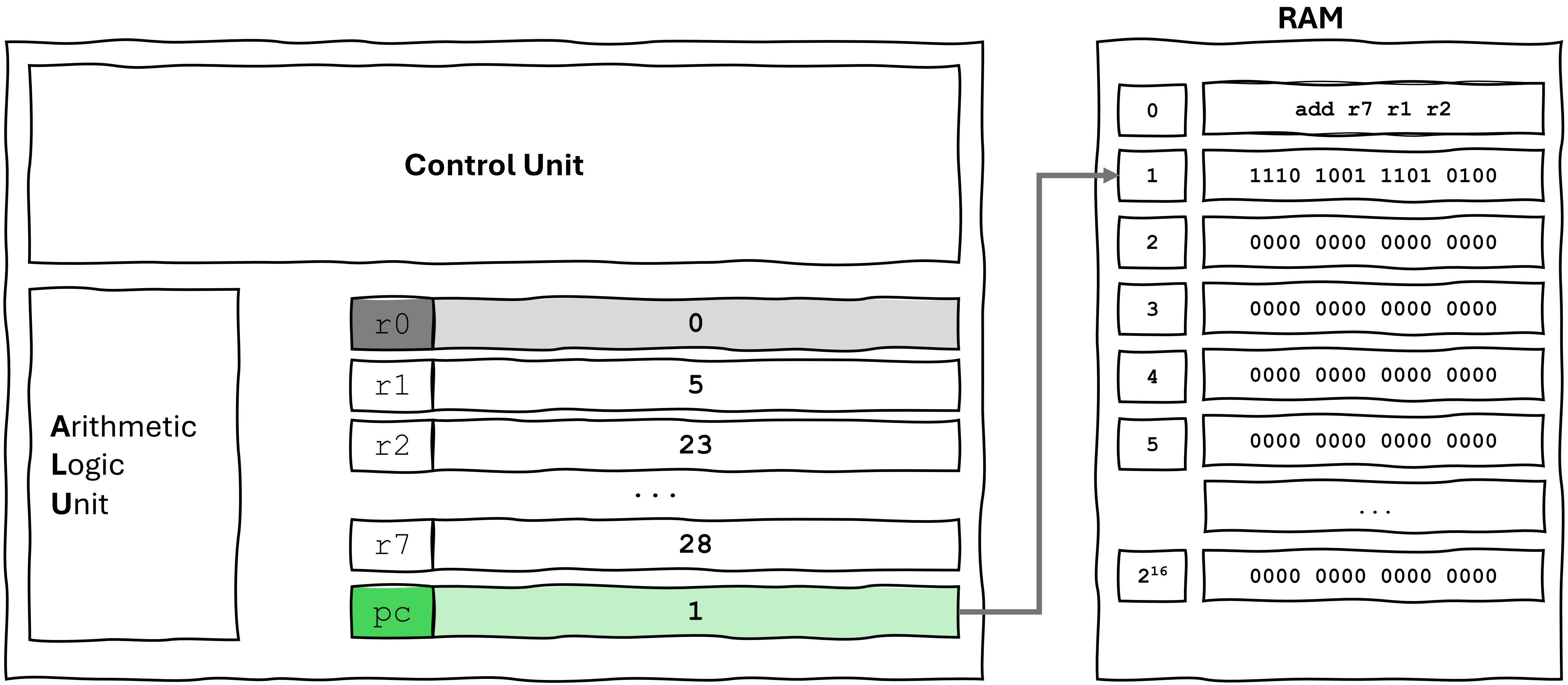
Arithmetic
Logic
Unit

r0	0
r1	5
r2	23
	...
r7	28
pc	1

RAM

0	add r7 r1 r2
1	1110 1001 1101 0100
2	0000 0000 0000 0000
3	0000 0000 0000 0000
4	0000 0000 0000 0000
5	0000 0000 0000 0000
	...
2^{16}	0000 0000 0000 0000

ANNA Architecture (Overview)



Control Unit

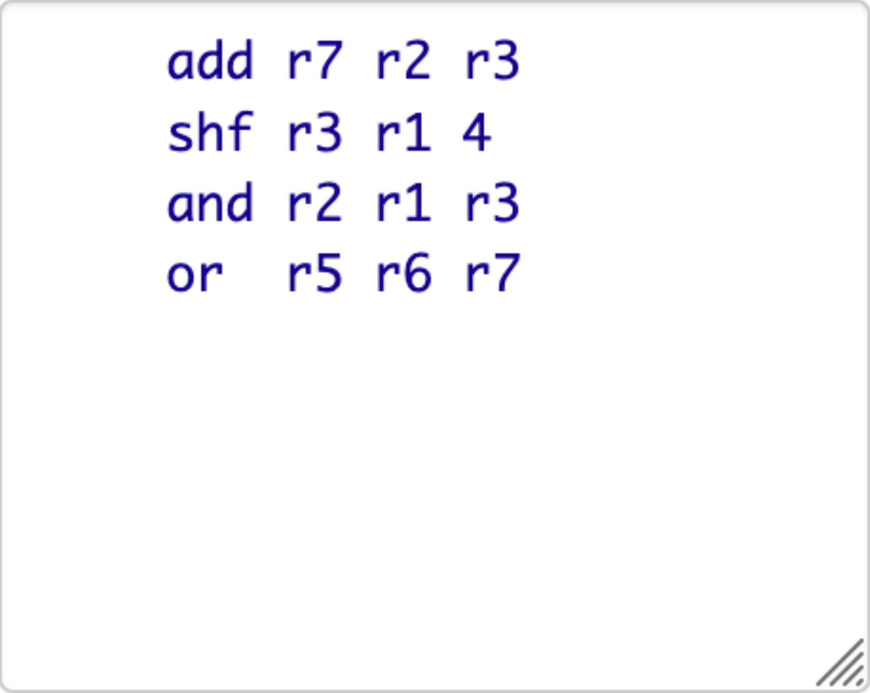
Arithmetic Logic Unit

r0	0
r1	5
r2	23
...	
r7	28
pc	1

RAM

0	add r7 r1 r2
1	1110 1001 1101 0100
2	0000 0000 0000 0000
3	0000 0000 0000 0000
4	0000 0000 0000 0000
5	0000 0000 0000 0000
	...
2^{16}	0000 0000 0000 0000

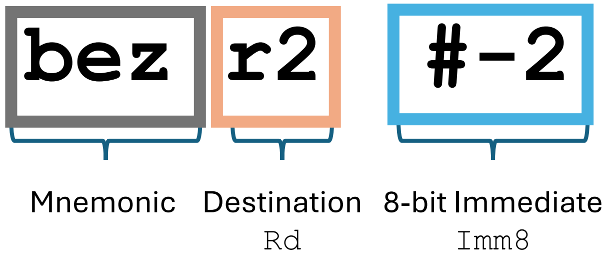
Assembly Code Example



```
add r7 r2 r3
shf r3 r1 4
and r2 r1 r3
or  r5 r6 r7
```

BEZ Instruction

(branch equal zery)



bez r2 #-2

Mnemonic

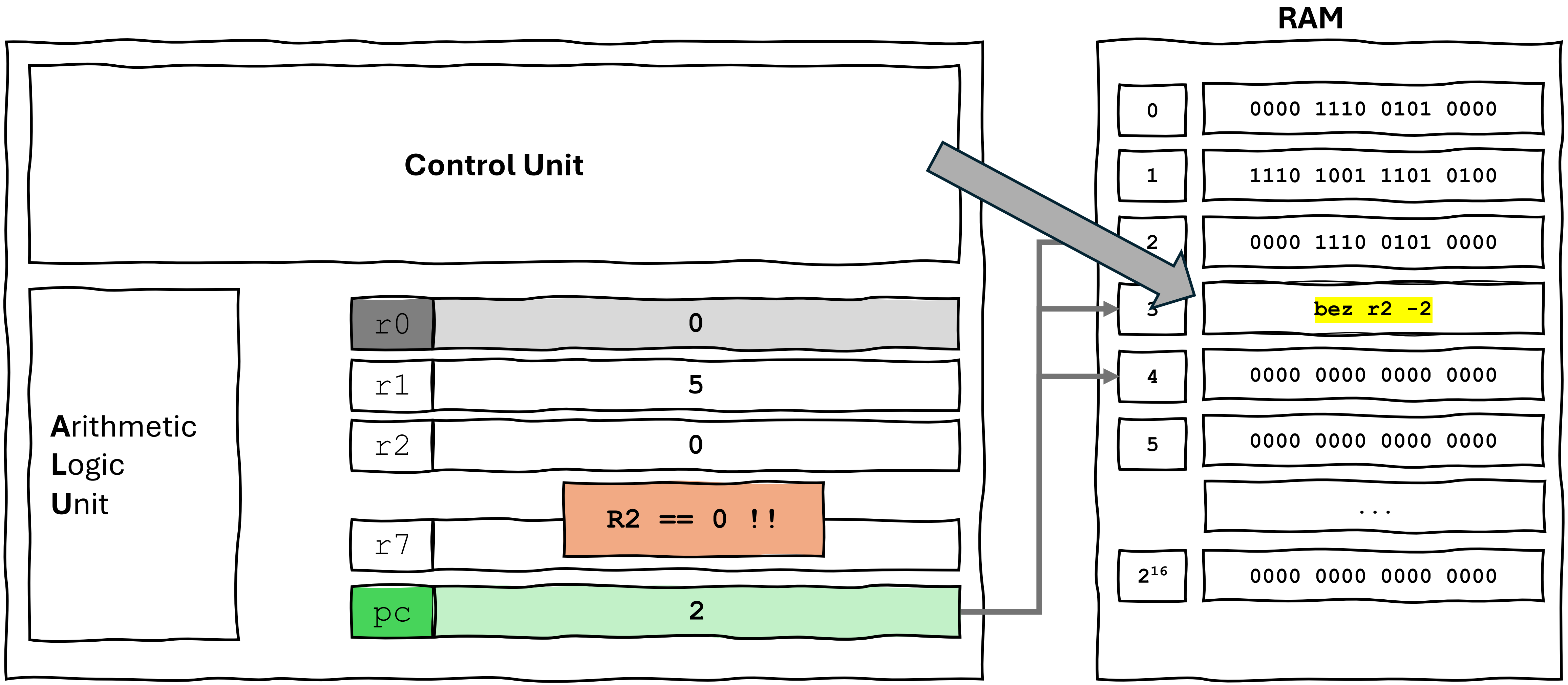
Destination
Rd

8-bit Immediate
Imm8

```
if (r2 == 0){
    pc -= 2;
}
```

1	0	1	0	0	1	0	0	1	1	1	1	1	1	1	0
15			12	11		9	8	7							0
Opcode				Rd			Unused	Imm8							

ANNA Architecture (Overview)



Control Unit

Arithmetic
Logic
Unit

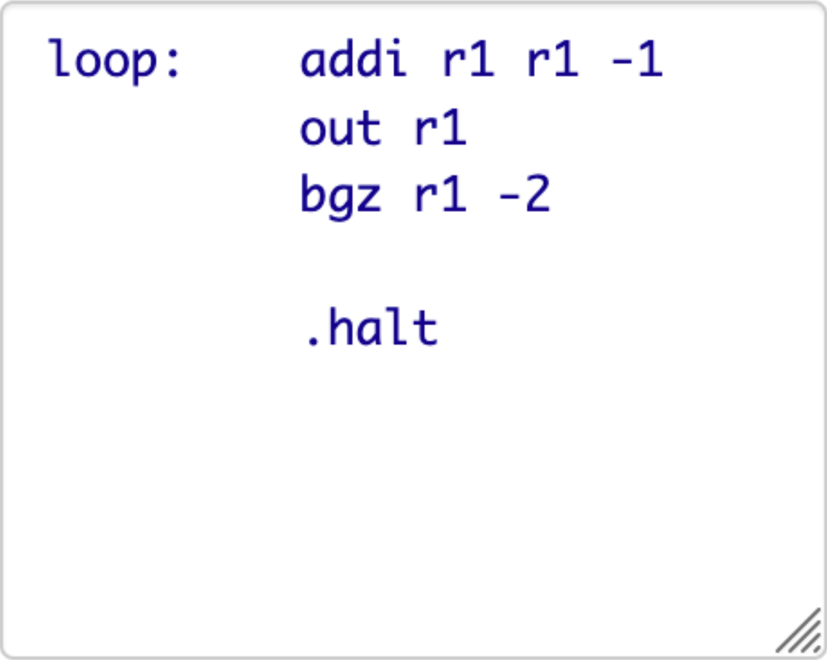
r0	0
r1	5
r2	0
r7	
pc	2

R2 == 0 !!

RAM

0	0000 1110 0101 0000
1	1110 1001 1101 0100
2	0000 1110 0101 0000
3	bez r2 -2
4	0000 0000 0000 0000
5	0000 0000 0000 0000
	...
2^{16}	0000 0000 0000 0000

Jump Example



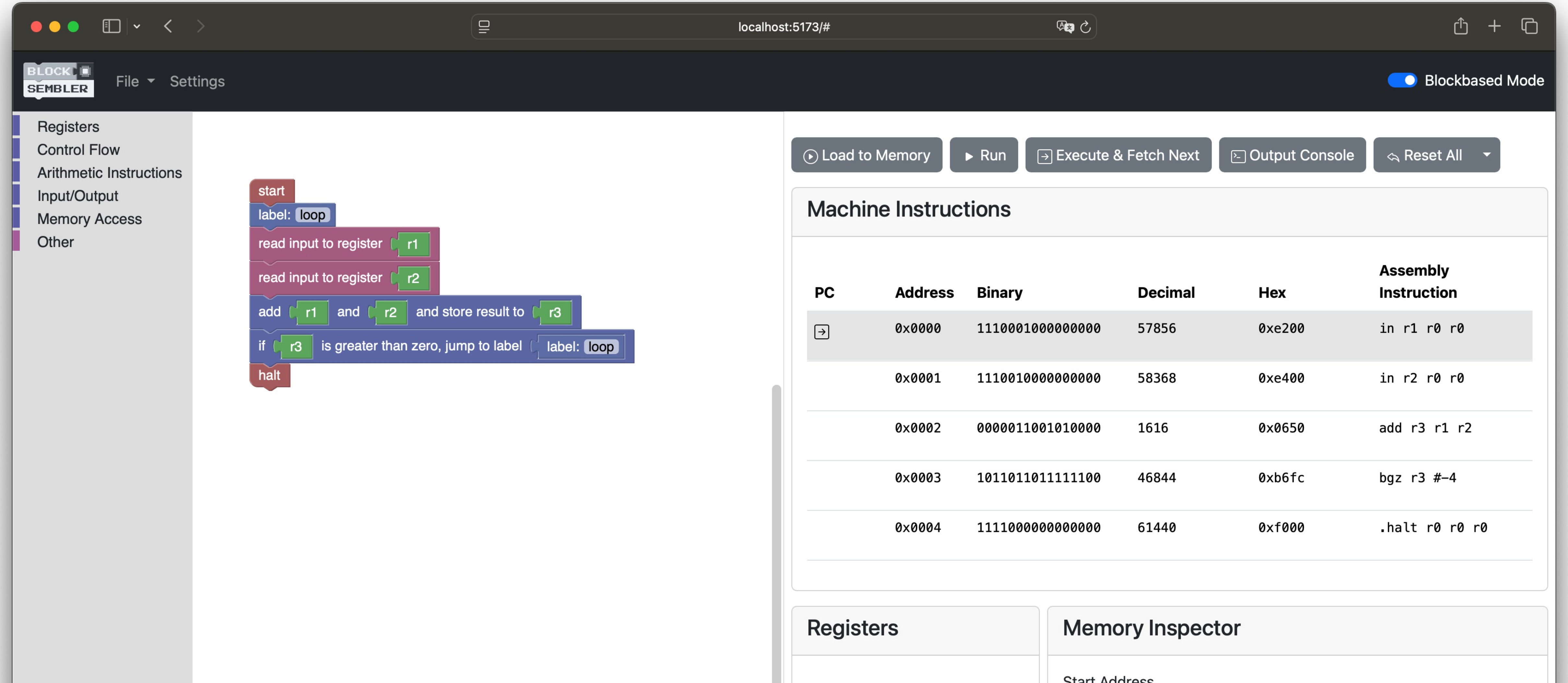
```
loop:    addi r1 r1 -1
         out r1
         bgz r1 -2

         .halt
```

Anna Instruction Set
(Overview)

- `add`
- `sub`
- `and`
- `or`
- `not`
- `shf` *(shift)*
- `lli` *(load lower immediate)*
- `lui` *(load upper immediate)*
- `lw` *(load word)*
- `sw` *(store word)*
- `bez` *(branch equal zero)*
- `bgz` *(branch greater zero)*
- `addi` *(add immediate)*
- `jalr` *(jump and link register)*
- `in`
- `out`

Blocksembler

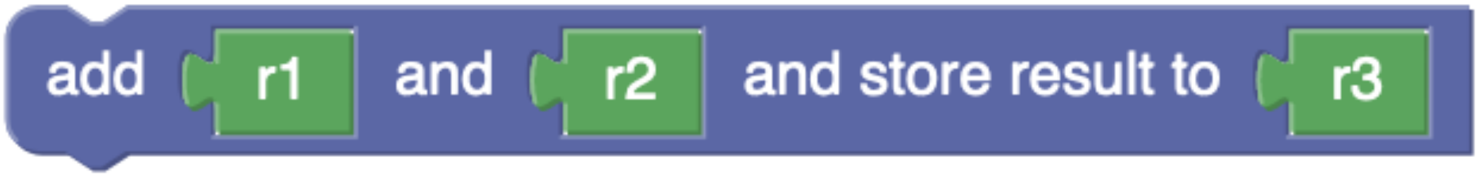


```
start
label: loop
read input to register r1
read input to register r2
add r1 and r2 and store result to r3
if r3 is greater than zero, jump to label label: loop
halt
```

Machine Instructions

PC	Address	Binary	Decimal	Hex	Assembly Instruction
→	0x0000	1110001000000000	57856	0xe200	in r1 r0 r0
	0x0001	1110010000000000	58368	0xe400	in r2 r0 r0
	0x0002	0000011001010000	1616	0x0650	add r3 r1 r2
	0x0003	1011011011111100	46844	0xb6fc	bgz r3 #-4
	0x0004	1111000000000000	61440	0xf000	.halt r0 r0 r0

Blocksembler Instructions



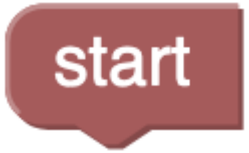
add r1 and r2 and store result to r3

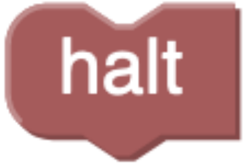
Start/Halt

- Blocksembler programs always begin with a **`start`** block.

- When the Control Unit encounters a **`halt`** instruction (represented by the *Halt* block), the program terminates.

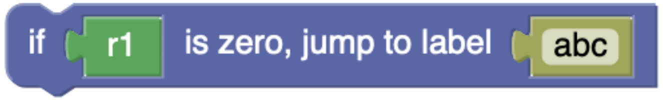
- Using multiple *Halt* blocks within a program is allowed.





JUMP/Branch Instructions

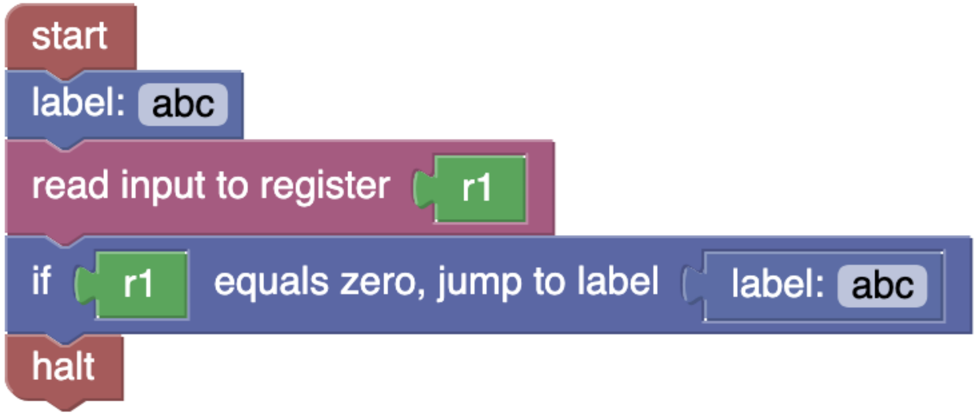
There are instructions that cause the program flow to jump to a specific line in the program when a certain condition is met.



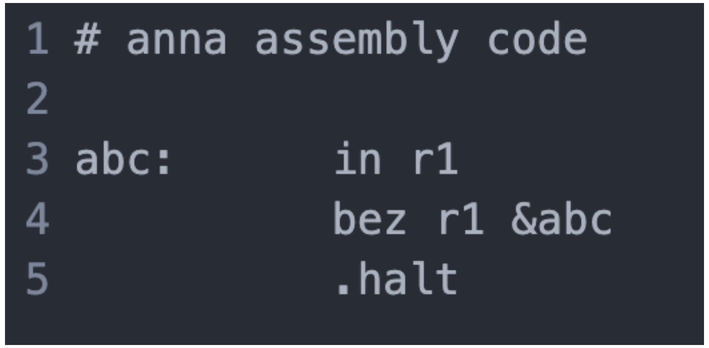
if r1 is zero, jump to label abc

Label

To define the target of such a jump, `labels` are used.



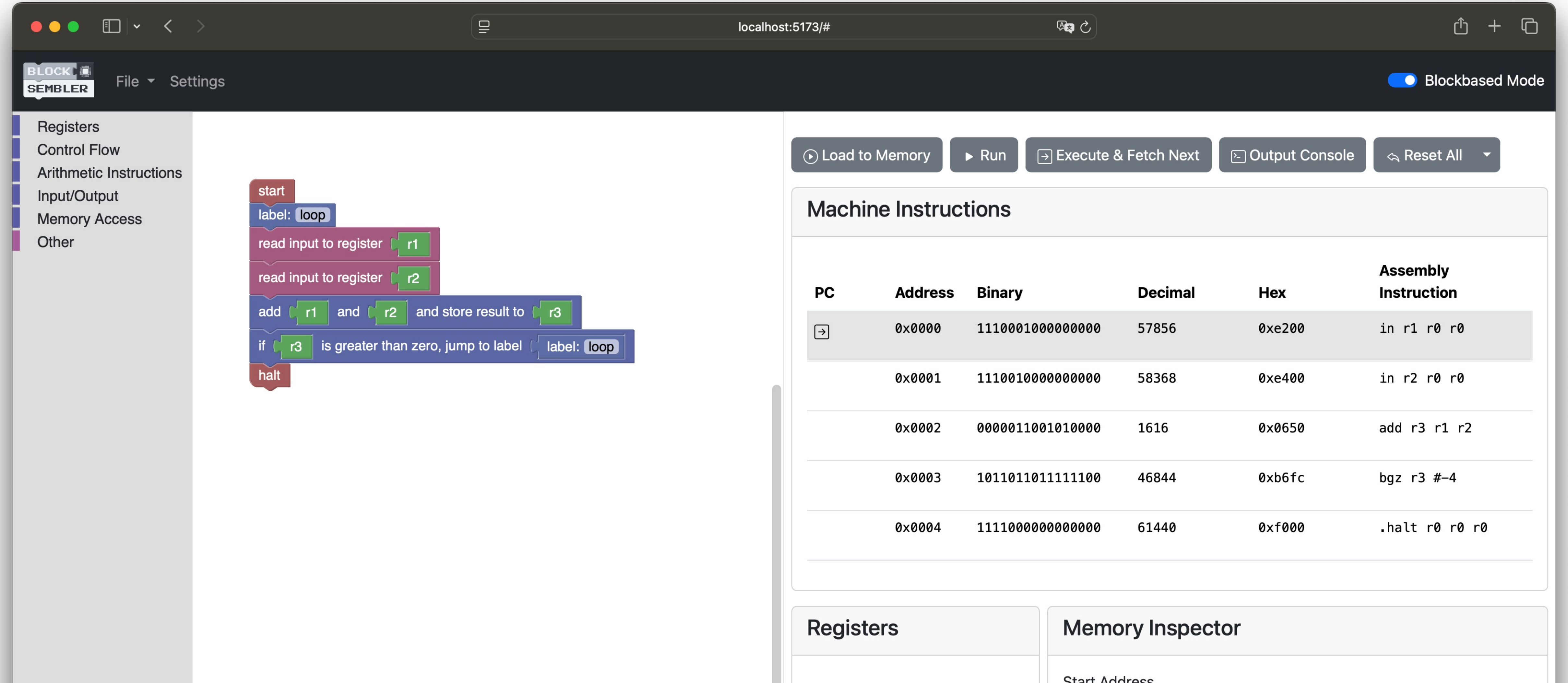
```
start
label: abc
read input to register r1
if r1 equals zero, jump to label label: abc
halt
```



```
# anna assembly code

abc:        in r1
            bez r1 &abc
            .halt
```

Live Demonstration



start
label: loop
read input to register r1
read input to register r2
add r1 and r2 and store result to r3
if r3 is greater than zero, jump to label label: loop
halt

Machine Instructions

PC	Address	Binary	Decimal	Hex	Assembly Instruction
→	0x0000	1110001000000000	57856	0xe200	in r1 r0 r0
	0x0001	1110010000000000	58368	0xe400	in r2 r0 r0
	0x0002	0000011001010000	1616	0x0650	add r3 r1 r2
	0x0003	1011011011111100	46844	0xb6fc	bgz r3 #-4
	0x0004	1111000000000000	61440	0xf000	.halt r0 r0 r0

Registers

Memory Inspector

Start Address

Resources

Presentation:
https://blocksembler.github.io/assets/presentation.pdf

ANNA Documentation:

https://blocksembler.github.io/assets/anna.pdf

Blocksembler:
https://blocksembler.eden.univie.ac.at

mail: florian.woerister@univie.ac.at