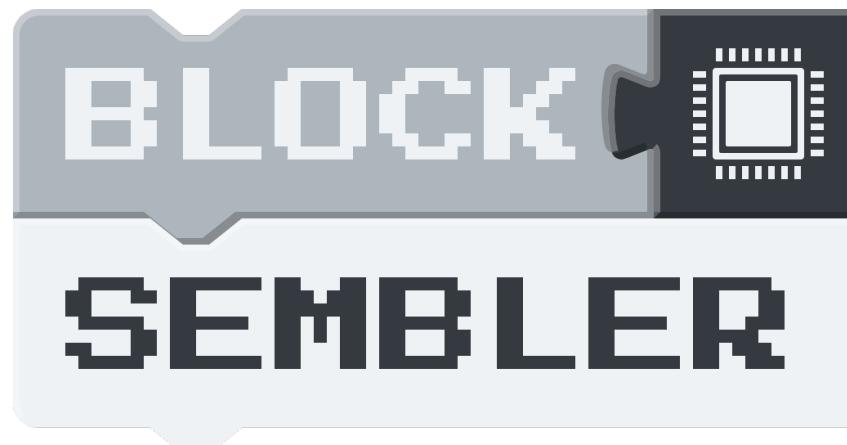
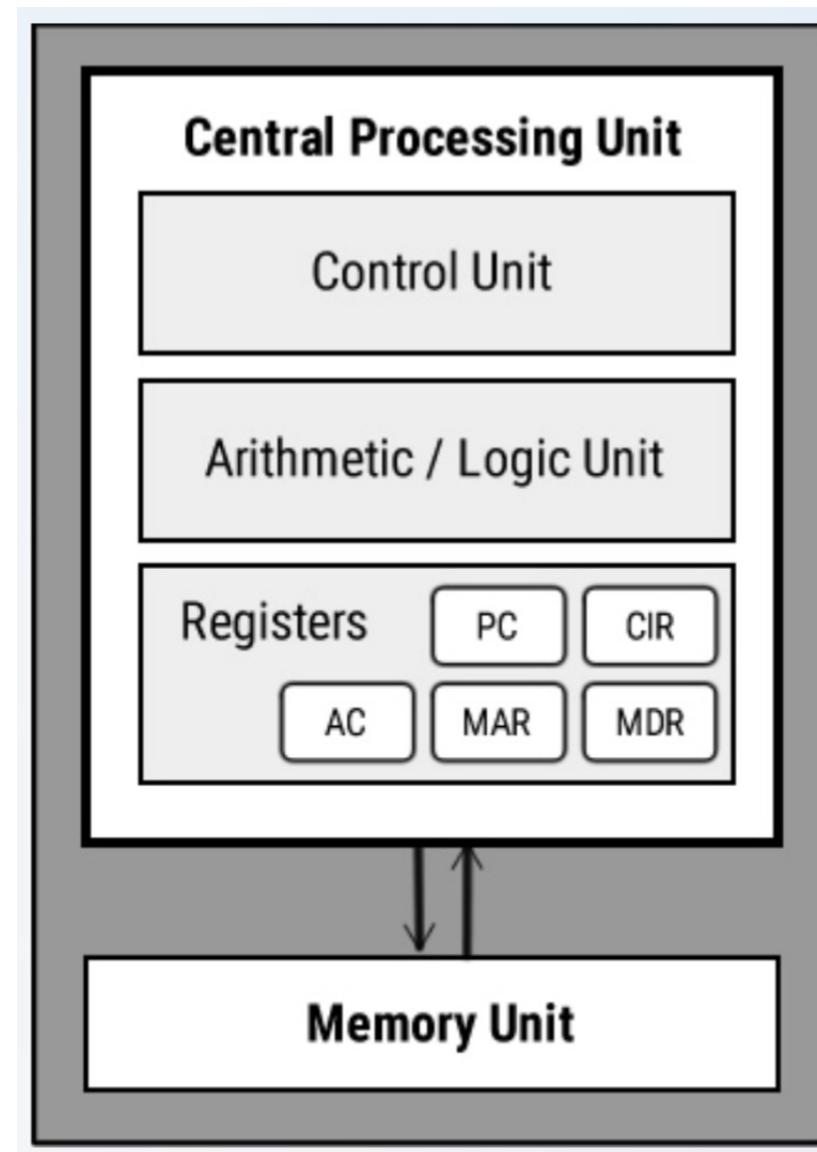


Assembly Programming with



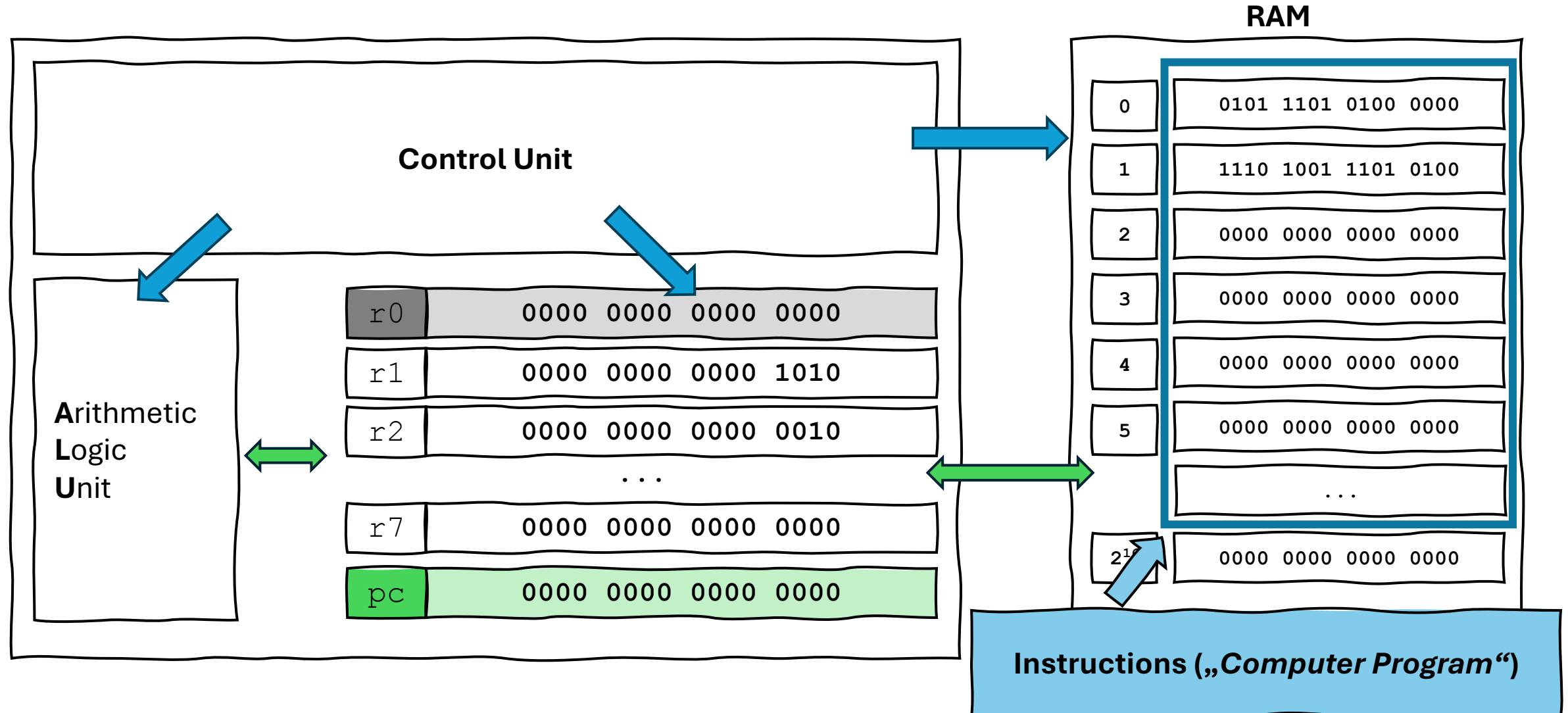
Recap



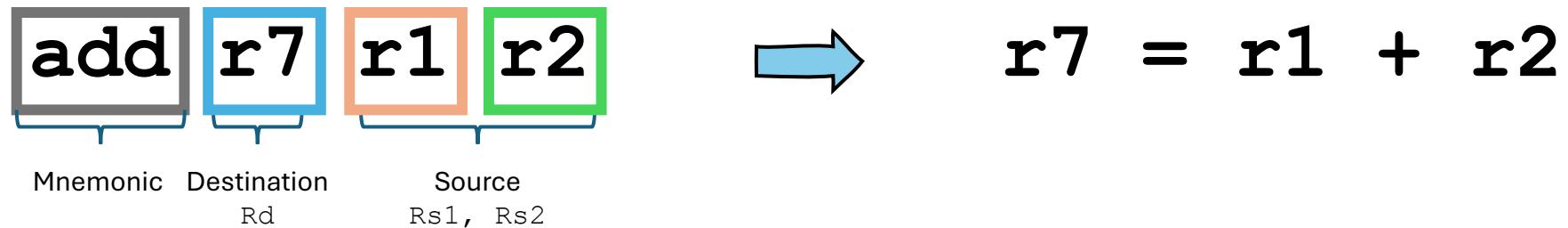
ANNA (A New Noncomplex Architecture)

- 16-bit Architecture
- 8 Registers (r_0, \dots, r_7)
- 16 Instructions
- $2^{16} \times$ Words of RAM

ANNA Architecture (Overview)

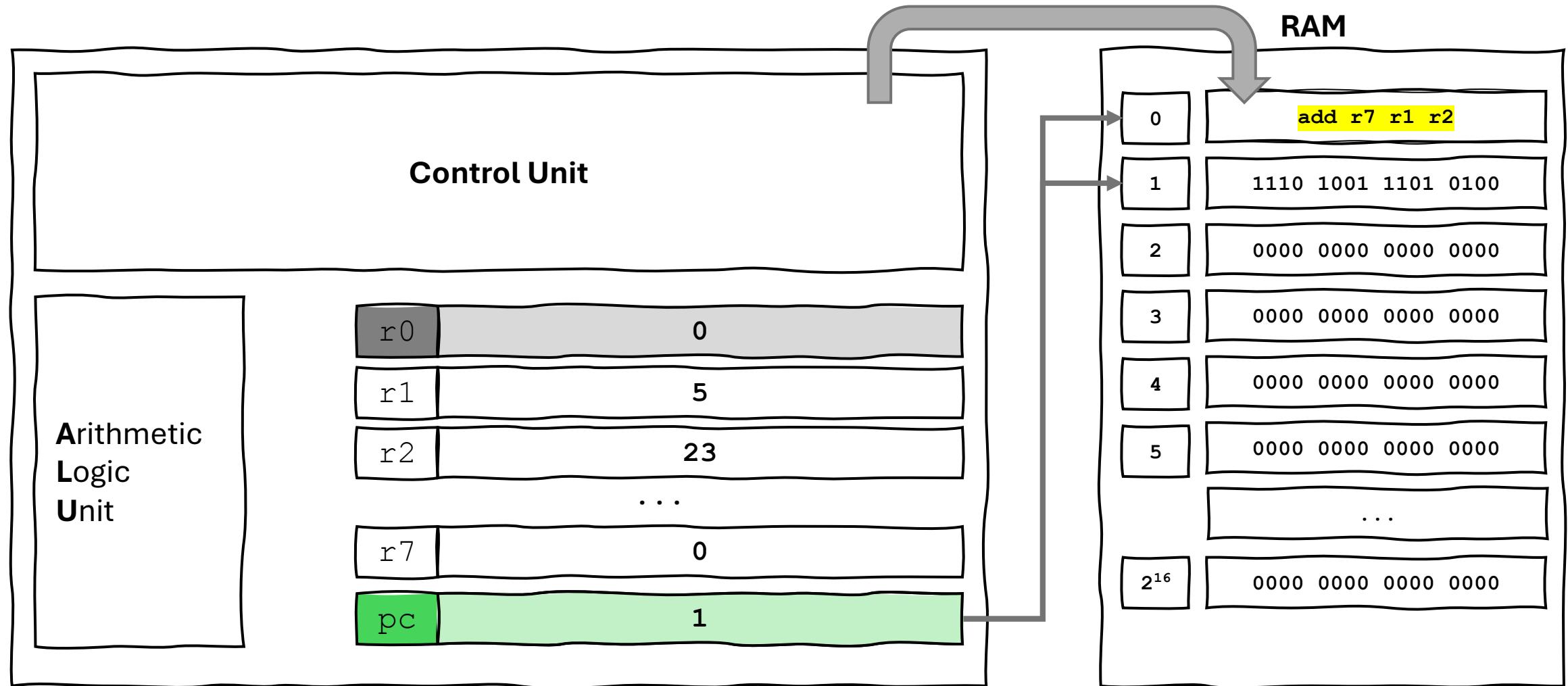


ADD Instruction

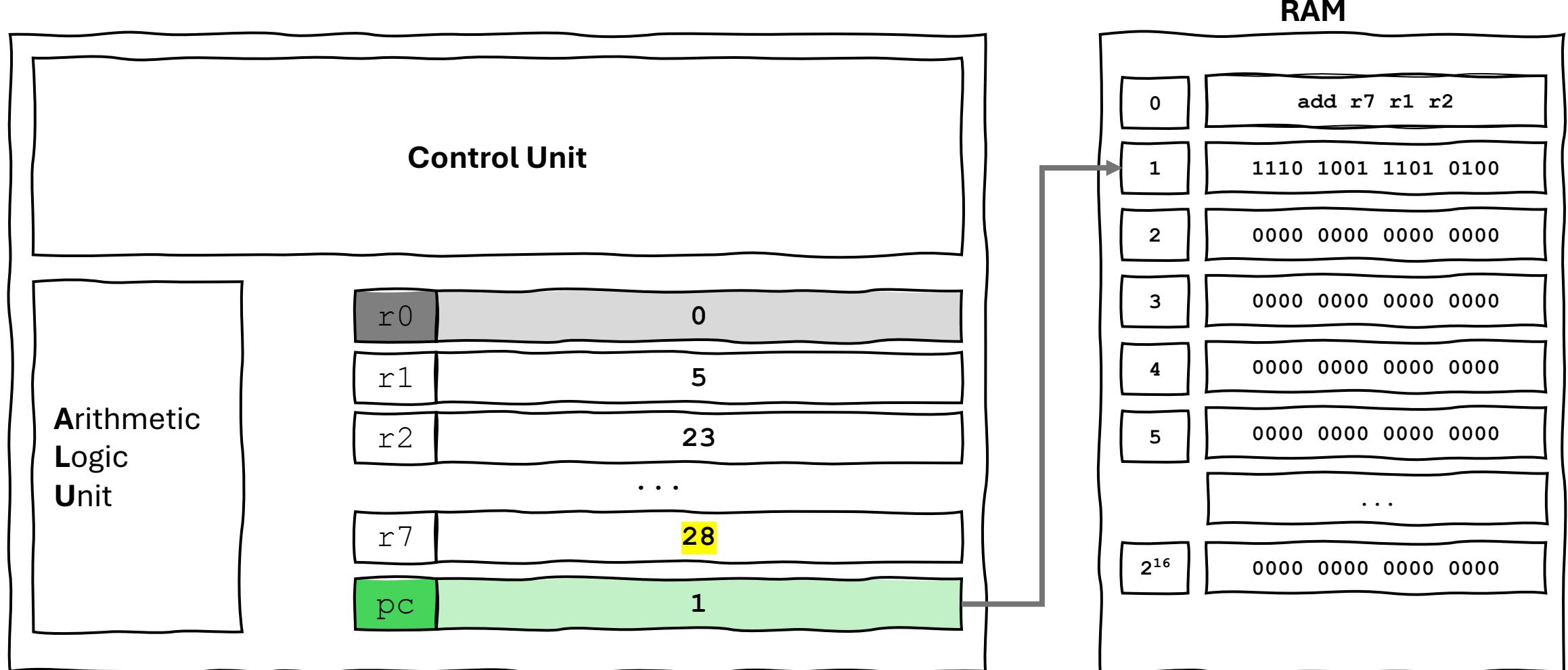


0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	0
15			12	11		9	8		6	5		3	2		0
Opcode				Rd	Rs ₁				Rs ₂	Unused					

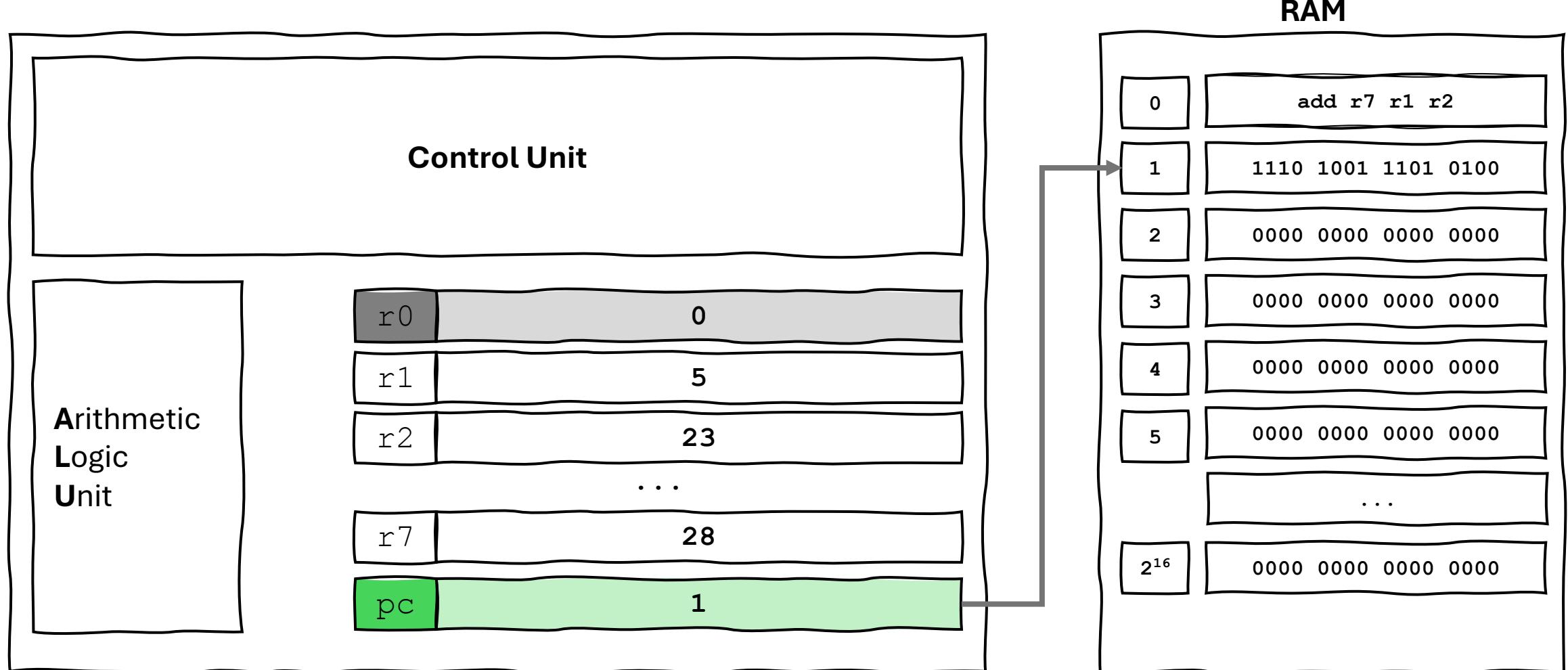
ANNA Architecture (Overview)



ANNA Architecture (Overview)



ANNA Architecture (Overview)



Assembly Code Example

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

BEZ Instruction

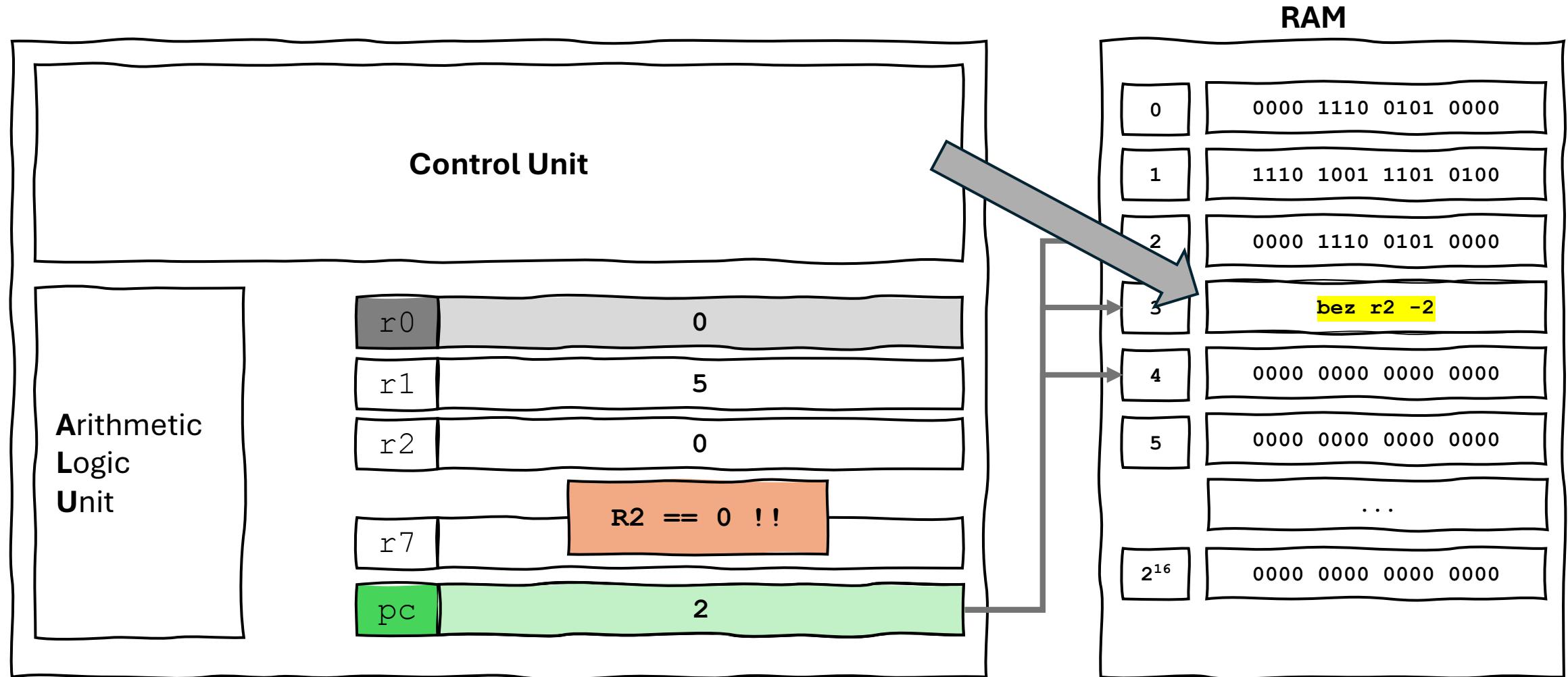
(branch equal zero)

The diagram illustrates the fields of the **BEZ r2 # -2** instruction. It consists of three colored boxes: a grey box for the Mnemonic, an orange box for the Destination register **r2**, and a blue box for the 8-bit Immediate value **# -2**. Below each box is a blue bracket indicating its function: **Mnemonic**, **Destination**, and **8-bit Immediate**.

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

1	0	1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	0
15			12	11		9	8	7									0
Opcode				Rd				Unus ed	Imm8								

ANNA Architecture (Overview)



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

The screenshot shows the Blocksembler interface, a graphical programming environment. On the left, a sidebar lists categories: Program Structure, Primitives, System Instructions, Memory Instructions, Arithmetic Instructions, Logic Instructions, and Compare and Branching. The main area is divided into two sections: a flowchart editor on the left and an assembly code viewer on the right.

Flowchart Editor (Left):

- ENTRYPPOINT:** A sequence of blocks:
 - move value hex value: 0x0018 to register Register: \$1
 - Register: \$2 := Register: \$1 decimal value: 1
 - label: @ loop
 - compare Register: \$2 and decimal value: 1 and update status register
 - jump to address Label: > isPrime if last comparison was equal
 - move value Register: \$1 to register Register: \$3
 - label: @ innerLoop
 - compare Register: \$3 and decimal value: 0 and update status register
 - jump to address Label: > exitInner if last comparison was less or equal
 - Register: \$3 := Register: \$3 Register: \$2
 - jump to address Label: > innerLoop
 - label: @ exitInner
 - jump to address Label: > noPrime if last comparison was equal
 - Register: \$2 := Register: \$2 decimal value: 1

Assembly Code (Right):

```
1  mov $1, 0x0018
2  sub $2, $1, 1
3
4  @loop:
5  cmp $2, 1
6  beq >isPrime
7  mov $3, $1
8
9  @innerLoop:
10  cmp $3, 0
11  ble >exitInner
12  sub $3, $3, $2
13  jmp >innerLoop
14
15 @exitInner:
16  beq >noPrime
17  sub $2, $2, 1
18  jmp >loop
19
20 @noPrime:
21  mov $4, 1
22  jmp >exit
23
24 @isPrime:
25  mov $4, 0
```

Blocksembler Instructions

add  and  and store result to 

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.



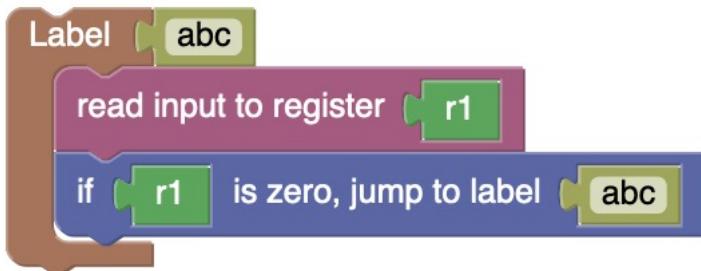
JUMP/Branch Instructions

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



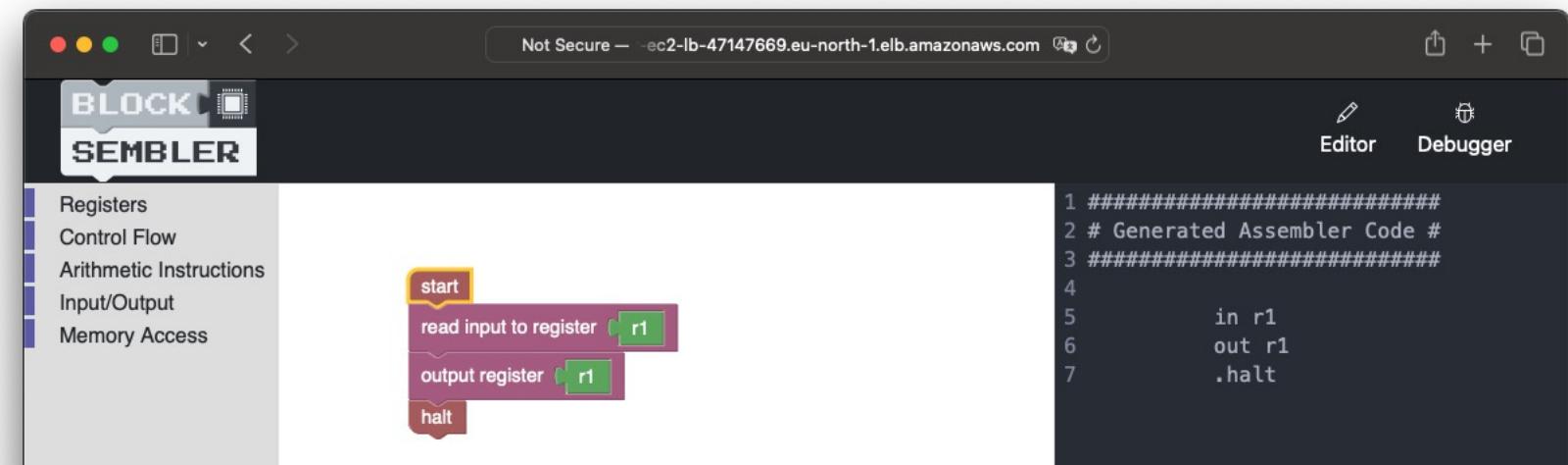
Label

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



```
1 #####  
2 # Generated Assembler Code #  
3 #####  
4  
5  
6 abc:      in r1  
7          bez r1 &abc
```

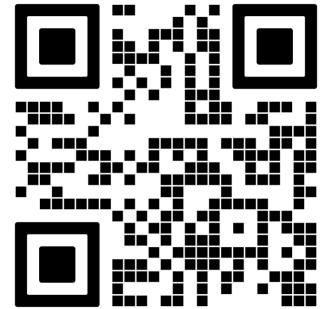
Live Demonstration



Resources

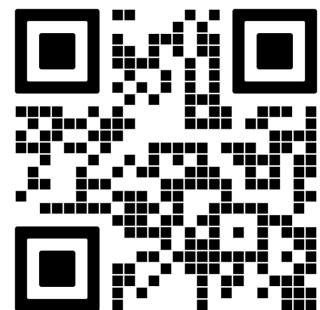
Presentation:

<https://blocksembler.github.io/presentation.pdf>



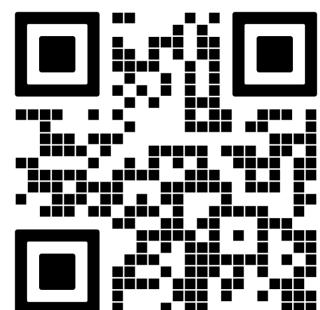
ANNA Documentation:

<https://blocksembler.github.io/anna.pdf>



Blocksembler:

<https://blocksembler.eden.univie.ac.at>



mail: florian.woerister@univie.ac.at