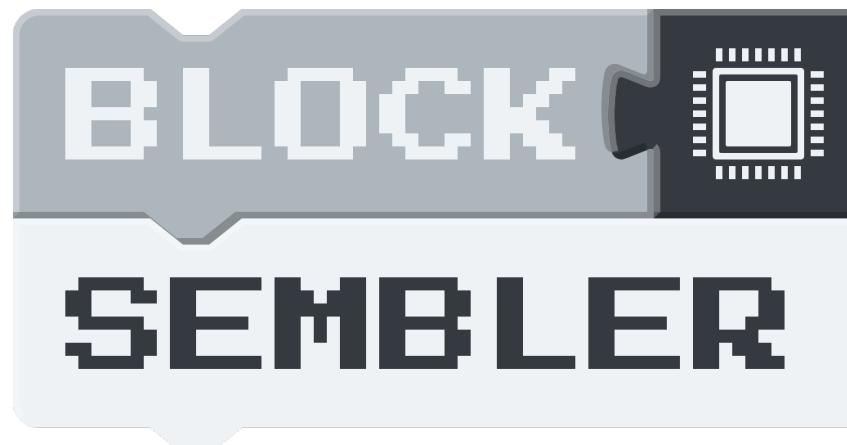
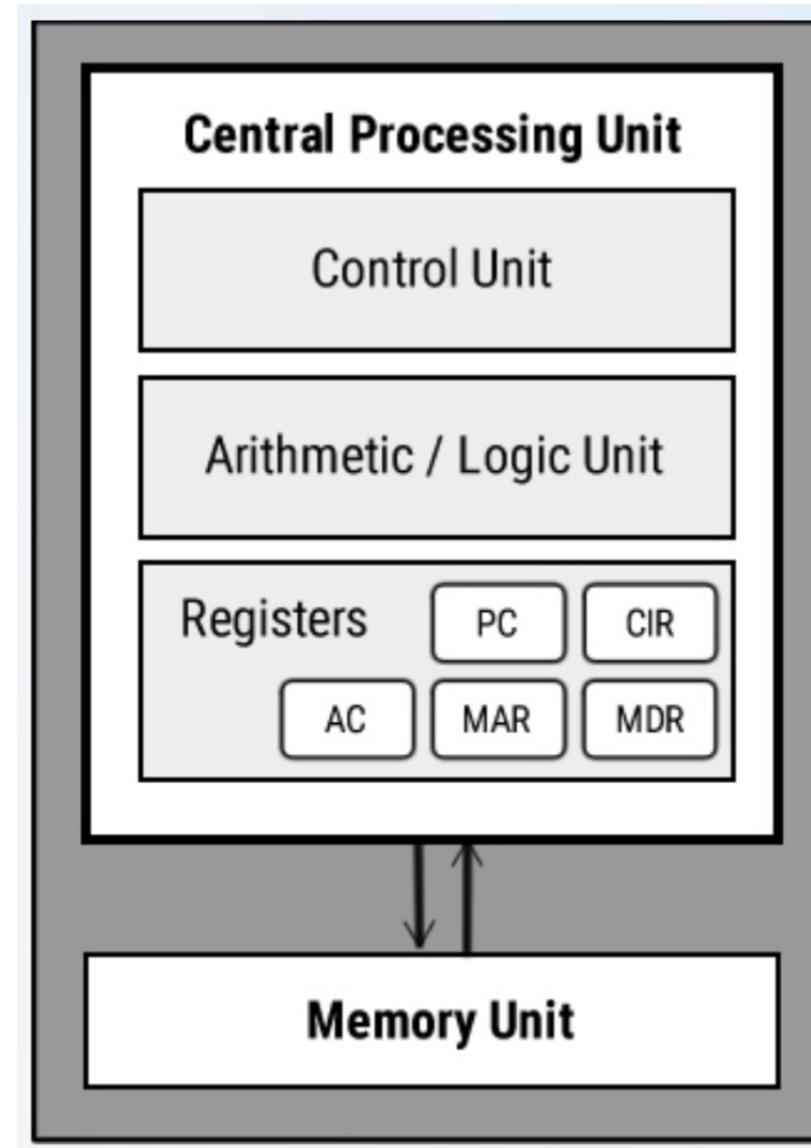


Assembly Programming with



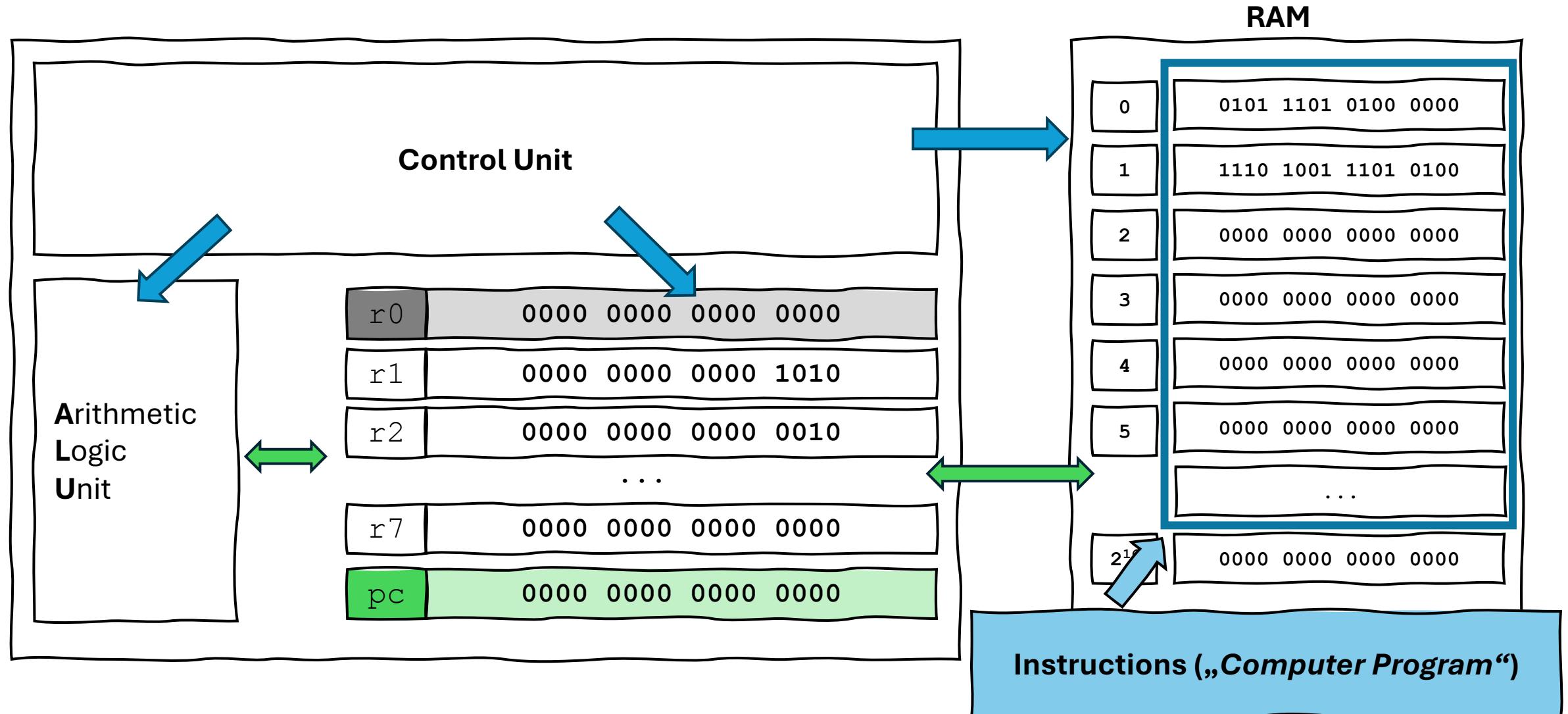
Recap



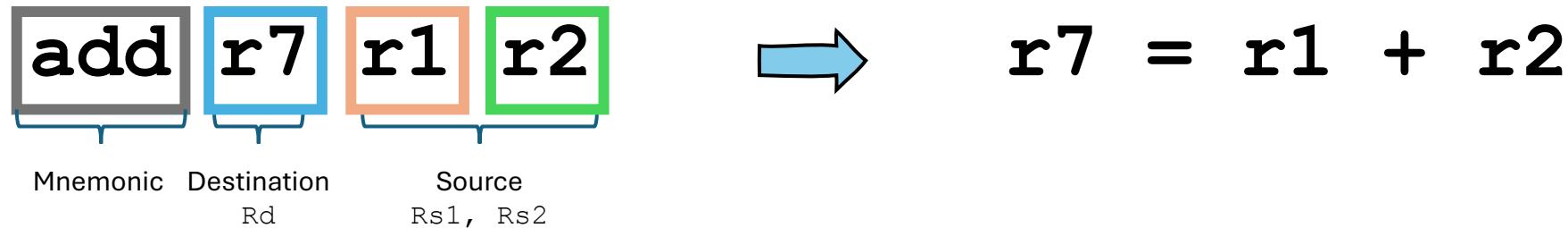
ANNA (A New Noncomplex Architecture)

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

ANNA Architecture (Overview)

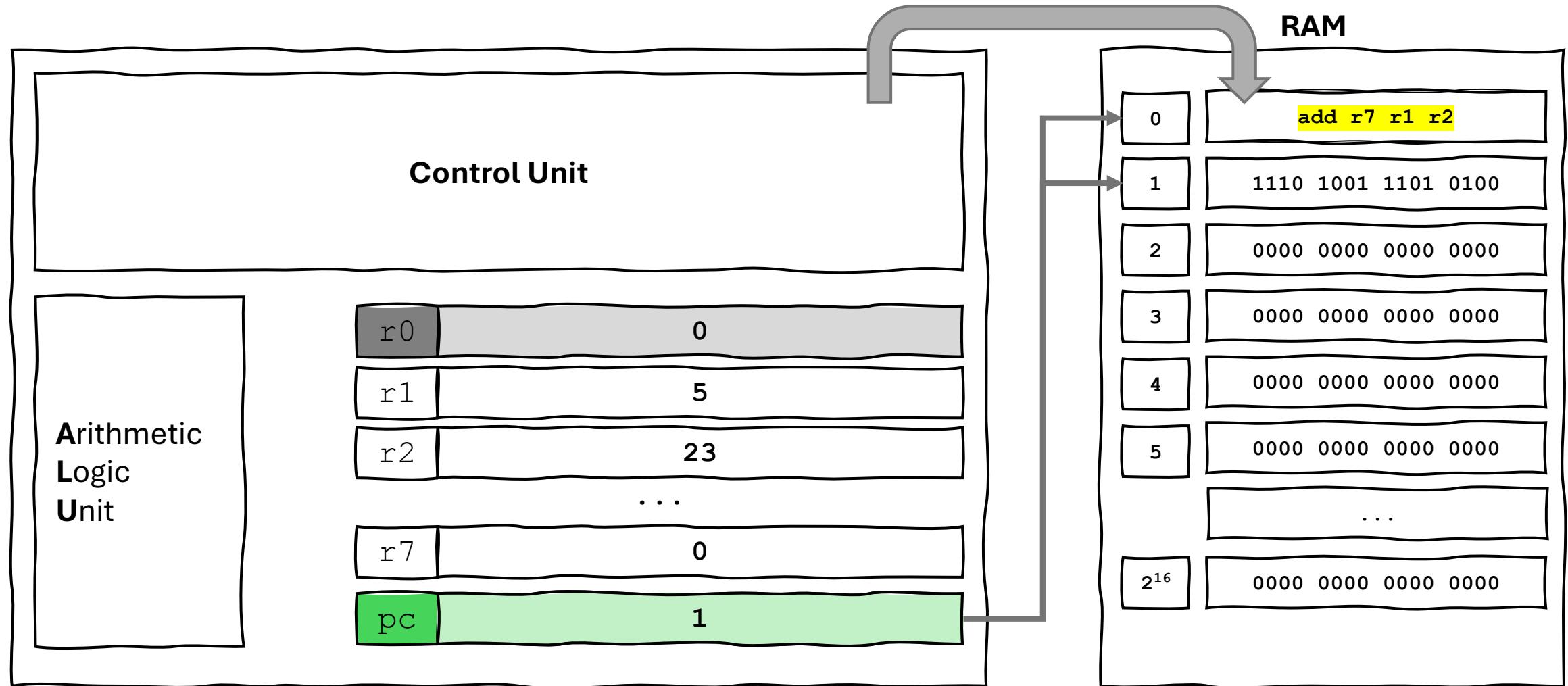


ADD Instruction

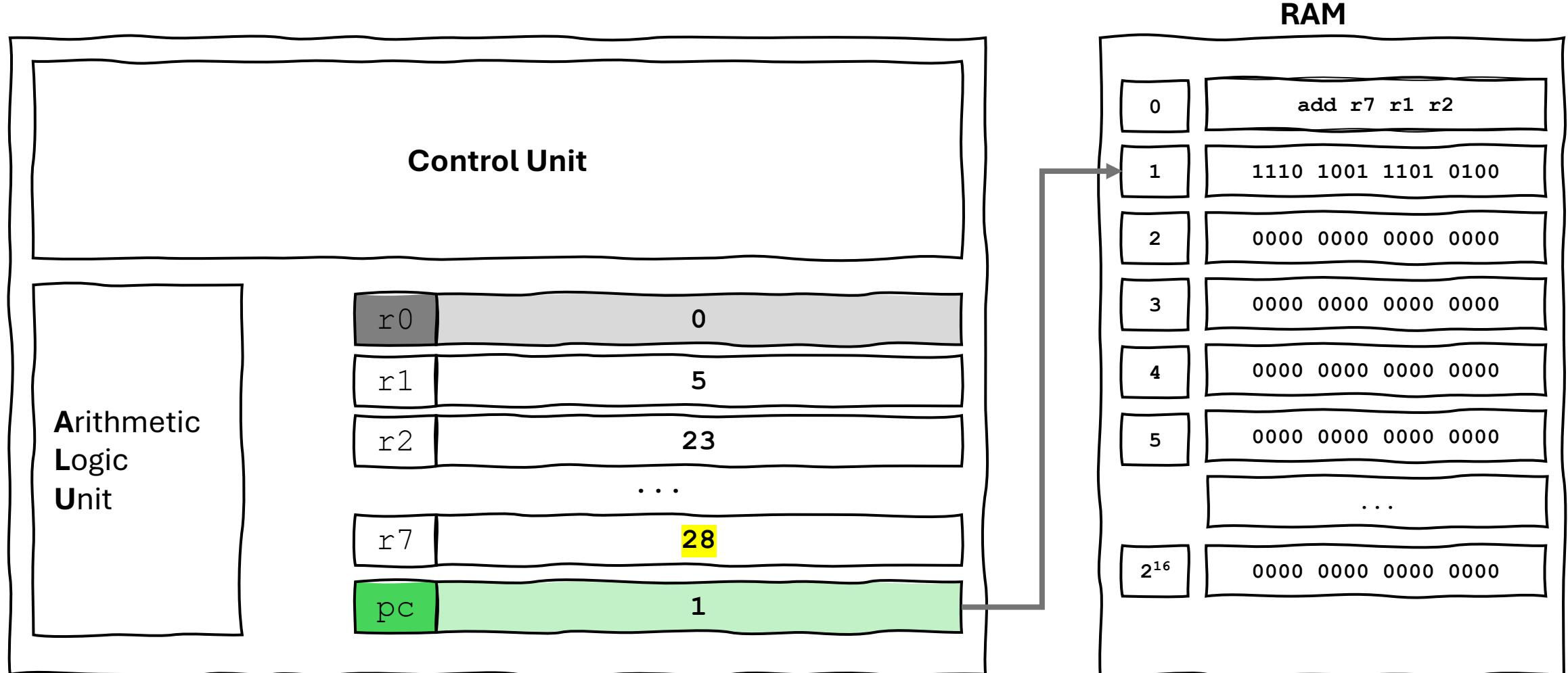


0	0	0	0	1	1	1	0	0	1	0	1	0	0	0
15			12	11		9	8		6	5		3	2	0
Opcode			Rd			Rs ₁			Rs ₂			Unused		

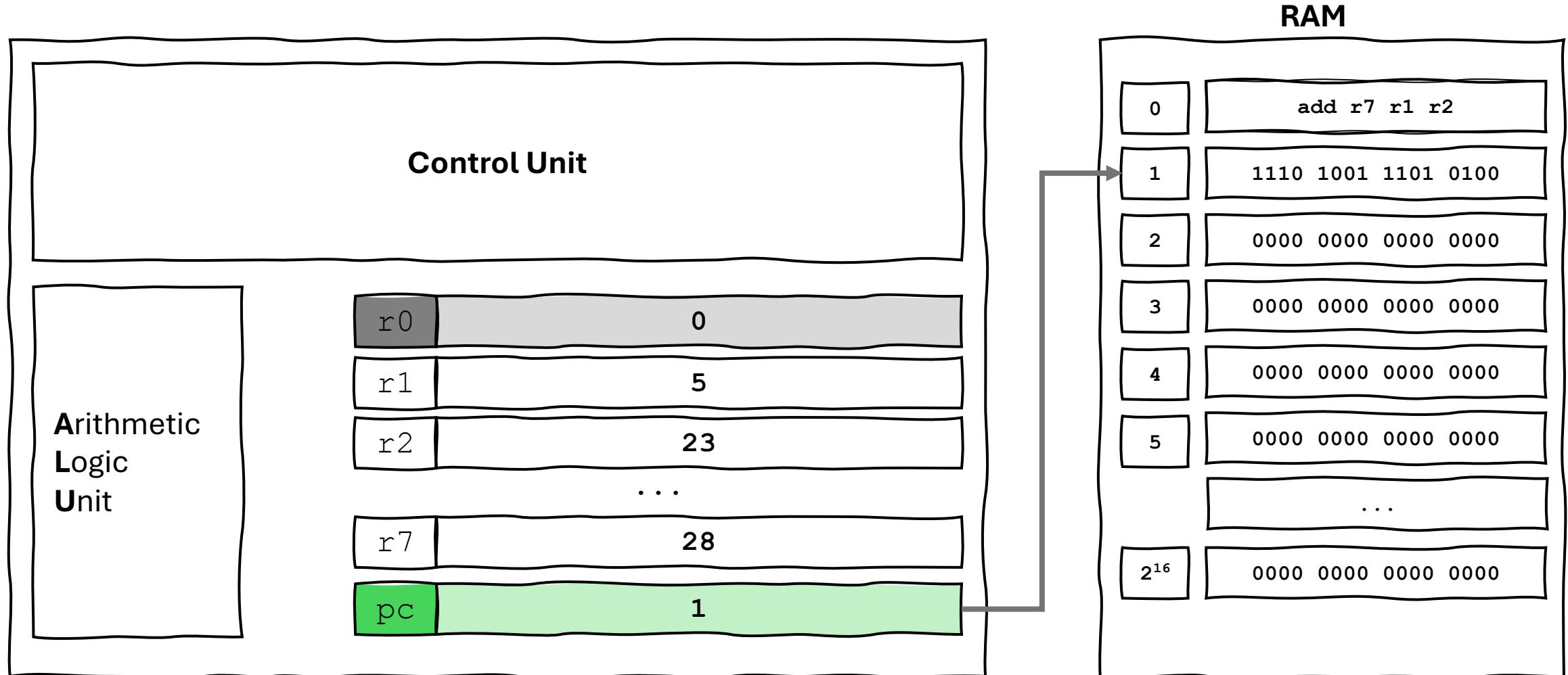
ANNA Architecture (Overview)



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)

Diagram illustrating the BEZR2 instruction format:

- Mnemonic:** bez
- Destination:** r2
- 8-bit Immediate:** #-2

Below the boxes:

- Mnemonic
- Destination
- 8-bit Immediate

Rd

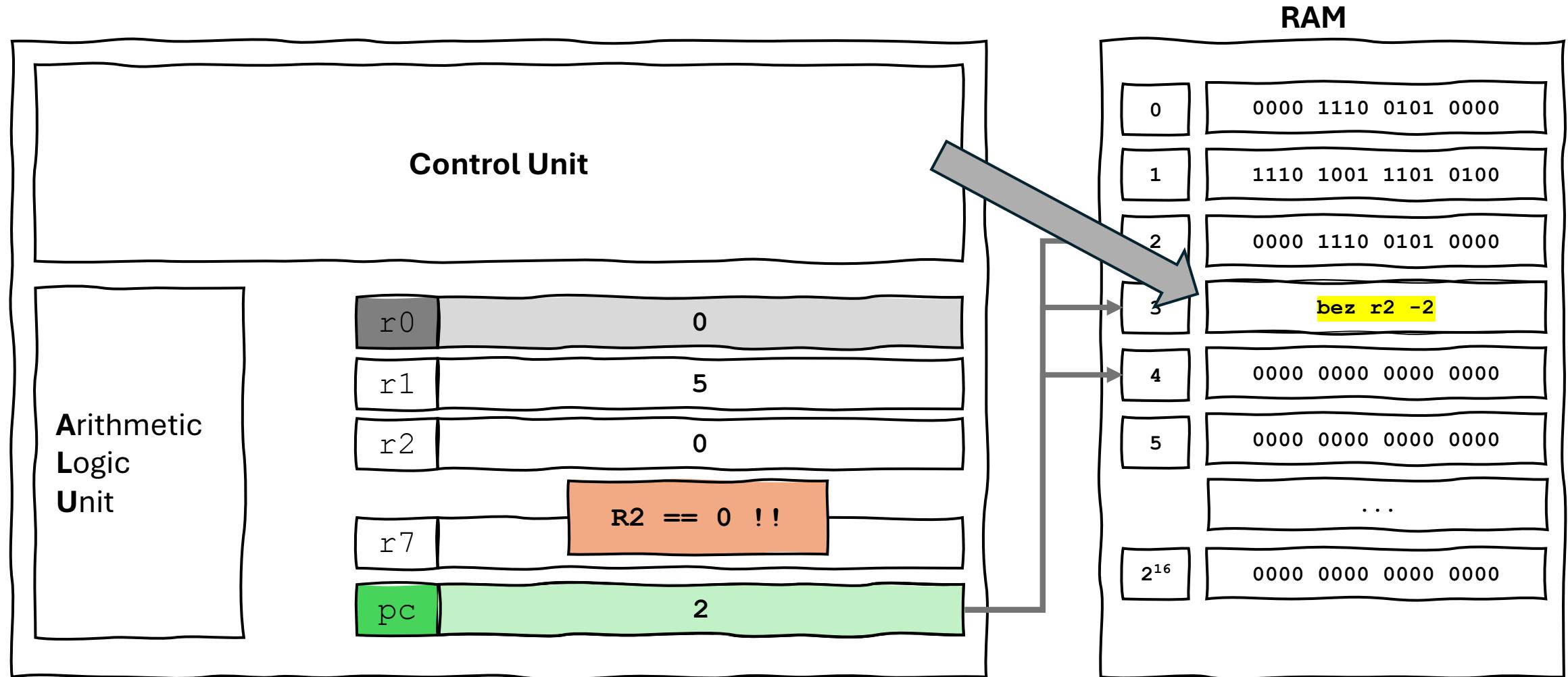
Imm8



```
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 block-based programming environment for a RISC-V like architecture. The interface is divided into several sections:

- Header:** A dark header bar with a back/forward button, a search bar containing "localhost:5173/#", and a file menu.
- Left Sidebar:** A sidebar with a "BLOCKSEMBLER" logo and a "Blockbased Mode" toggle switch. Below the logo are categories: Registers, Control Flow, Arithmetic Instructions, Input/Output, Memory Access, and Other.
- Code Editor:** The main workspace where a block-based program is built. The program starts at a "start" label, reads input to registers r1 and r2, adds r1 and r2 to store in r3, and then checks if r3 is greater than zero, jumping back to the "loop" label if true. It ends with a "halt" block.
- Control Buttons:** A row of buttons at the top right: "Load to Memory", "Run", "Execute & Fetch Next", "Output Console", and "Reset All".
- Machine Instructions Table:** A table showing the assembly output for each instruction. The columns are PC, Address, Binary, Decimal, Hex, and Assembly Instruction.
- Registers Table:** A table showing the current values of registers r0, r1, r2, and r3.
- Memory Inspector:** A table showing memory contents at a specific address.

Machine Instructions Table Data:

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

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 # anna assembly code
2
3 abc:      in r1
4         bez r1 &abc
5         .halt
```

Live Demonstration

The screenshot shows a live demonstration of the Blocksembler interface, which translates block-based code into machine instructions.

Block-based Code:

```
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 loop
halt
```

Machine Instructions:

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

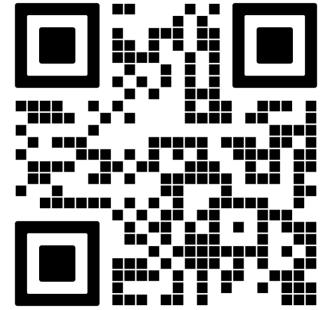
Registers: (Empty table)

Memory Inspector: (Empty table)

Resources

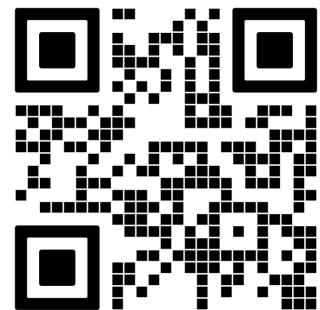
Presentation:

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



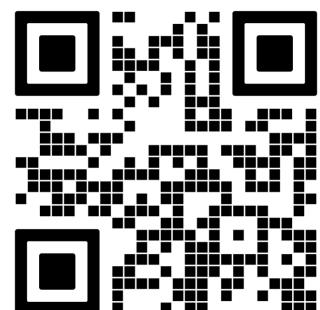
ANNA Documentation:

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



Blocksembler:

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



mail: florian.woerister@univie.ac.at

