# Logisim CPU background information

This document contains some information for understanding the Logisim CPU simulation, which is also given in a video. Note that the CPU simulation aims to demonstrate some of the fundamental principles of computers and how a programmable computer can be created from simple components. There are, however, a lot of simplifications made in the Logisim CPU simulation.

The Logisim CPU simulation has two registers, the program counter (PC) and the accumulator (ACC). The program counter gives the address of the instruction that is to be run the next time the clock changes to 1.

The ALU always takes one input from the accumulator and the other input according to the instruction given in the program. The output from the ALU always goes to the accumulator.

The Logisim CPU simulation has separate storage for programs and data. Each item in the data memory is 8 bits (2 hex digits) and so can hold numbers from 0 to 255 when using unsigned integers or -128 to 127 with signed integers. The address for the data memory is 8 bits (so can store 256 items of data with addresses from 00 to ff).

To run the simulation, a program has to be loaded into the program memory. Each instruction in the program memory is 16 bits (4 hex digits) and this memory also has an 8-bit address like the data memory (so can store 256 items)

The program instructions are structured as follows with the least significant bit being bit 0

* bits 12-15 give the operation for the CPU to carry out
* bits 8-11 give the operation for the ALU to carry out
* bits 0-7 give an address (to be used for the data memory) or an immediate value (to be used directly)

This information, and the relation to hex digits is summarised in the table below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CPU operation | | | | ALU Operation | | | | Address or immediate value | | | | | | | |
| Hex digit | | | | Hex digit | | | | Hex digit | | | | Hex digit | | | |

1 0 0 0

2 1 0 0

The CPU operations are as follows:

0 no operation

1 use immediate value as input to ALU

2 use address and pass the value stored at that address in the data memory in to the ALU

3 copy value from accumulator (ACC) to data memory at address given in instruction

4 branch always to address in program memory given in the instruction

5 branch to program address when ACC value is not equal to zero

f halt

Note that CPU operations 4, 5 and f do not work in the starter CPU circuit provided (they should be completed as tasks). There is flexibility to add more CPU operations as hex digits 6 to e do not have an operation assigned.

The ALU operations are as follows:

0 don't use (pass the new input to ACC)

1 add the two inputs

2 bitwise xor of the two inputs

3 multiply the two inputs

4 bitwise not on the input from the ACC

Note that ALU operations 3 and 4 do not work in the starter circuit provided (they should be completed as tasks). Again, there is flexibility to add operations.

## Looking at a demo program

To look at the workings, we will look at a program that stores two numbers in the data memory and then adds and multiplies those numbers and stores the results and then the program stops. Not all these features work on the starter version of the CPU simulation. The video talks through running this program on a version of the CPU where multiplication has been implemented in the CPU.

The program to load into the program memory reads as follows:

v2.0 raw

100f 3004 100a 3005 2104 3000 2004 2305 3001 f000

Looking at each instruction in turn

|  |  |  |  |
| --- | --- | --- | --- |
| CPU operation | ALU operation | Address or immediate | Result |
| 1 | 0 | 0f | Pass immediate (hex) value of 0f through the ALU to the accumulator (decimal value 15) |
| 3 | 0 | 04 | Copy the current value in the accumulator to the data memory given by the address (04) |
| 1 | 0 | 0a | Pass immediate (hex) value of 0a through the ALU to the accumulator (decimal value 10). This will overwrite the previous value on the accumulator. |
| 3 | 0 | 05 | Copy the current value in the accumulator to the data memory given by the address (05) |
| 2 | 1 | 04 | Use the ALU **add** operation with the value from the data memory at address 04 (remember that one input will be from the accumulator) (in decimal we are adding 15 + 10 =25, which is 19 in hex). This will overwrite the previous value on the accumulator. |
| 3 | 0 | 00 | Copy the current value in the accumulator to the data memory given by the address (00) (So that will hold hex 19) |
| 2 | 0 | 04 | Pass the value from the data memory at address 4 through the ALU to the accumulator. We stored hex 0f (decimal 15) at address 04 in the data memory, so that will now appear on the ACC. |
| 2 | 3 | 05 | Use the ALU **multiply** operation with the value from the data memory at address 05 (remember that one input will be from the accumulator) (in decimal we are calculating 15 \* 10 = 150, which is 96 in hex). This will overwrite the previous value on the accumulator. |
| 3 | 0 | 01 | Copy the current value in the accumulator to the data memory given by the address (01) (So that will hold hex 96) |
| f | 0 | 00 | Stop the program |