| | 1st Byte | Instruction | Example | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 2nd Byte | 3rd Byte | 4th Byte | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| None Immediate | | | | 0 | 0 | | | | | | | Address | Address | Target Address | | | | |
| 2nd Immediate | | i | addji 1 A B | 1 | 0 | | | | | | | Literal | Address | Target Address | | | | |
| 3rd Immediate | | j | subij A 1 A | 0 | 1 | | | | | | | Address | Literal | Target Address | | | | |
| | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Operation | | | | | | 0 | 0 | | | | | | | | | | | |
| | Maths | | | | | 0 | 0 | 0 | | | | | | | | | | |
| | Add | add | add A B C | | | 0 | 0 | 0 | 0 | 0 | 0 | Address | Address | Target Address | | | | |
| | Subtract | sub | sub A B C | | | 0 | 0 | 0 | 0 | 0 | 1 | Address | Address | Target Address | | | | |
| | Multiply | mul | mul A B C | | | 0 | 0 | 0 | 0 | 1 | 0 | Address | Address | Target Address | | | | |
| | Shift Left | shl | shl A B C | | | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | |
| | Shift Right | shr | shr A B C | | | 0 | 0 | 0 | 1 | 0 | 0 | | | | | | | |
| | Negate | neg | neg A _ C | | | 0 | 0 | 0 | 1 | 0 | 1 | | | | | | | |
| | Divide | div | div A B C | | | 0 | 0 | 0 | 1 | 1 | 0 | Address | Address | Target Address | | | | |
| | Modulo | mod | mod A B C | | | 0 | 0 | 0 | 1 | 1 | 1 | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Bitwise | | | | | | 0 | 0 | 1 | | | | | | | | | | |
| | AND | and | and A B C | | | 0 | 0 | 1 | 0 | 0 | 0 | Address | Address | Target Address | | | | |
| | OR | or | or A B C | | | 0 | 0 | 1 | 0 | 0 | 1 | Address | Address | Target Address | | | | |
| | NOT | not | not A _ C | | | 0 | 0 | 1 | 0 | 1 | 0 | Address | _ | Target Address | | | | |
| | XOR | xor | xor A B C | | | 0 | 0 | 1 | 0 | 1 | 1 | Address | Address | Target Address | | | | |
| | NAND | nand | nand A B C | | | 0 | 0 | 1 | 1 | 0 | 0 | | | | | | | |
| | NOR | nor | nor A B C | | | 0 | 0 | 1 | 1 | 0 | 1 | | | | | | | |
| | XNOR | xnor | xnor A B C | | | 0 | 0 | 1 | 1 | 1 | 0 | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Condition | | | | | | 0 | 1 | | | | | | | | | | | |
| | Compare against values | | | | | 0 | 1 | 1 | | | | | | | | | | |
| | Equal | je | je A B label_name | | | 0 | 1 | | 0 | 0 | 1 | Address | Address | Jump Address | | | | |
| | Not Equal | jne | jne A B label_name | | | 0 | 1 | | 1 | 0 | 1 | Address | Address | Jump Address | | | | |
| | Less Than | jl | jl A B label_name | | | 0 | 1 | | 0 | 1 | 0 | Address | Address | Jump Address | | | | |
| | Less Than Or Equal To | jle | jle A B label_name | | | 0 | 1 | | 0 | 1 | 1 | Address | Address | Jump Address | | | | |
| | Greater Than | jg | jg A B label_name | | | 0 | 1 | | 1 | 1 | 1 | Address | Address | Jump Address | | | | |
| | Greater Than Or Equal To | jge | jge A B label_name | | | 0 | 1 | | 1 | 1 | 0 | Address | Address | Jump Address | | | | |
| | *Space for more* | | | | | 0 | 1 | | | | | Address | Address | Jump Address | | | | |
| | Always | jmp | jmp _ _ label_name | | | 0 | 1 | 0 | 1 | 1 | 1 | _ | _ | Jump Address | | | | |
| | | | | | | | | | | | | | | | | | | |
| Moving | | | | 0 | 0 | 1 | 0 | | | 0 | 0 | Address | _ | Target Address | | | | |
| | Copy | cpy | cpy A _ B | | | 1 | 0 | 0 | 0 | | | | | | | | | |
| | Move (equivalent to copy) | mov | mov A _ B | | | 1 | 0 | 0 | 1 | | | | | | | | | |
| | Stack Pop | pop | pop _ _ A | | | 1 | 0 | 1 | 0 | | | | | | | | | |
| | Stack Push | psh | psh A _ _ | | | 1 | 0 | 1 | 1 | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Function | | | | | | 1 | 0 | | | 1 | | | | | | | | |
| | call | call | call label_name _ _ | | | 1 | 0 | 0 | 0 | 1 | 0 | | | | | | | |
| | return | ret | ret _ _ _ | | | 1 | 0 | 0 | 0 | 1 | 1 | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| RAM | | | | | ↓ | 1 | 1 | | | | | | Only 1 of these is required | | | | | |
| | Read | read | read 24 _ A | 0 | 0 | 1 | 1 | | | | | RAM Address | Read Target Address | Target Address | eg. | read\|(ramNo.) idx to A | read\|(ramNo.) _ A B ---- read (idx in address A) to (address B) | |
| | Write | writ | writ A _ 24 | 0 | 1 | 1 | 1 | | | | | Address | Read Target Address | RAM Address | | writ\|(ramNo.) A to idx | writ\|(ramNo.) A B _ ---- write (value in address A) to (idx in address B) | |
| | RAM 0 | | read\|0 24 _ A | | | 1 | 1 | 0 | 0 | 0 | 0 | | | | | | | |
| | RAM 1 | | writ\|1 A _ 24 | | | 1 | 1 | 0 | 0 | 0 | 1 | | | | | | | |
| | RAM 2 | | writ\|C io A _ | | | 1 | 1 | 0 | 0 | 1 | 1 | | | | | | | |
| | RAM 3 | | | | | 1 | 1 | 0 | 0 | 1 | 1 | | | | | | | |
| | RAM 4 | | | | | 1 | 1 | 0 | 1 | 0 | 0 | | | | | | | |
| | RAM 5 | | | | | 1 | 1 | 0 | 1 | 0 | 1 | | | | | | | |
| | RAM 6 | | | | | 1 | 1 | 0 | 1 | 1 | 1 | | | | | | | |
| | RAM 7 | | | | | 1 | 1 | 0 | 1 | 1 | 1 | | | | | | | |
| | RAM 8 | | | | | | | 1 | 0 | 0 | 0 | | | | | | | |
| | | | | | | | | 1 | 0 | 0 | 1 | | | | | | | |
| | | | | | | | | 1 | 0 | 1 | 1 | | | | | | | |
| | | | | | | | | 1 | 0 | 1 | 1 | | | | | | | |
| | | | | | | | | 1 | 1 | 0 | 0 | | | | | | | |
| | | | | | | | | 1 | 1 | 0 | 1 | | | | | | | |
| | | | | | | | | 1 | 1 | 1 | 1 | | | | | | | |
| | RAM 15 | | | | | | | 1 | 1 | 1 | 1 | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| | Running the Processor | | | | | | | | | | | | | | | | | |

Unfortunately, I was unable to implement 16-bit data, so 255 is the max right now - sorry.

The Input and Output of the processor both have double-dabble implementation
for ease of access when using the processor.

When you open the processor file, the first thing you should do is find the i/o area
and use the poke tool to press the "RESET" pin. This primes the RAM banks.

Next, you should find the ROM component labelled "Program" and edit the contents right-clicking
the component and pressing "Edit Contents". There, you can directly implement your own
program, or you can open a program file from your PC.

Then, you should set your input value. This may need to be changed in the midst of
a program, in which case you should NOT do the next step. Rather, you should hand
tick the clock, by pressing the button directly right of the middle button in the next step

Finally, go to the "Simulate" tab in the top left and press the middle button to begin
pulsing the clock. The default frequency is 0.5hz, but this can be changed in
Simulate/Auto-Tick Frequency in the top ribbon.

Your program is now running! Now just wait for it to finish executing...

| 1st Byte | Instruction | Example | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | | 2nd Byte | 3rd Byte | 4th Byte |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | | | | | | | | | | | | | | |
| And Presto! | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| Programming / Using the Replit program | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| Each instruction in your program will use 4 byte of data in the ROM. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| The first byte you type will be the opcode. | | | | | | | | | | | | | | |
| The second byte you type will be its first argument. | | | | | | | | | | | | | | |
| The third byte you type will be its second argument. | | | | | | | | | | | | | | |
| The fourth byte you type will be the address that the result is sent to. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| When you see an "_" in an instruction's example, it means that that byte is inherently empty. | | | | | | | | | | | | | | |
| To add these bytes, you should fill it in with the hex number "20". | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| Here is an example of what an instruction might look like: | | | | | | | | | | | | | | |
| add\|\| r0 24 r1 ---> 64 0 24 1 --> 40001801 | | * here, "\|\|" simply means that the second | | | | | | | | | | | | |
| ↑ this means bitwise OR the two values (0 OR 64 = 64) | | argument is an immediate value, i.e. not an address. | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| Converting the binary instructions to hex to form instructions is hella tedious, huh? | | | | | | | | | | | | | | |
| That's why I've created a python program so you don't have to! | | | | | | | | | | | | | | |
| Find it here at this link: https://replit.com/@LucasChambers/LEG-Code-Generation?v=1 | | | | | | | | | | | | | | |
| The program follows the same syntax as described in the examples in this guide. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| To type comments, type in a '#' at the end of an instruction. E.g. mov\|\| 27 to r1 **# move 27 to r1** | | | | | | | | | | | | | | |
| You can also just leave entire lines as comments with a '#' at the start of the line. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| All opcodes execute in order from left to right, i.e (add a b c) = a + b --> c | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| **Some special cases:** | | | | | | | | | | | | | | |
| To denote what RAM bank you are addressing, type '\|' after read/writ and then the address of the bank | | | | | | | | | | | | | | |
| You can also put a register address, etc. to signify the bank at the value in that address. | | | | | | | | | | | | | | |
| '\|i', '\|j' and '\|ij' can come after any opcode, and denote whether arg 1, 2 or both are immediates, respectively. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| You can use the keyword 'to' as a stand-in for '_' to improve readability - both function as explicit empties. | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |
| When you want to convert your typed code, type 'end' on a new line. | | | | | | | | | | | | | | |

| | Type | | | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Register | | 0 | 0 | 0 | 0 | 0 | | | | | | |
| | IO | | 1 | 0 | 0 | 1 | 0 | | | | | | These can be accessed like any other register |
| | Stack | | 2 | 0 | 1 | 0 | 0 | | | | | | |
| | Program Counter | | 3 | 0 | 1 | 1 | 0 | | | | | | |
| | Explicitly Empty "_" | | | | | | 1 | | | | | | |
| | Console Offset | | | 1 | 0 | 0 | 0 | | | | | | |
| | | | | | | | | | | | | | |
| Register | Register | A...H | 0 | | 0 | 0 | 0 | ? | ? | X | X | X | |
| | | | | | | | | | | | | | |
| IO | Input | io | 1 | | 0 | 1 | 0 | ? | ? | 0 | 0 | 0 | |
| | Output | io | 1 | | 0 | 1 | 0 | ? | ? | 0 | 0 | 0 | |
| | | | | | | | | | | | | | |
| Stack | Stack | stack | 2 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | Function Stack | Not ad | 2 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| | | | | | | | | | | | | | |
| PC | Program Counter | pc | 3 | | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | | | | | |
| RAM | RAM Banks | ram... | | | | | | | | | | | |