## 

## 

## 

## Final Project Documentation

Timing Diagrams and Input Selection Tables

Ariela Mitrani, Jerry Kurtin, Jade Davis, Garrett Guerrero

Halt

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 1; Stops the program from executing |
| Decode Ended/Execute Started | x |
| Execute Ended/Memory Started | x |
| Memory Ended/PC Update Started/Write Back Started | x |
| PC Update Ended | x |
| Write Back Ended/Program Done | x |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | x |
| Wrote rB | x |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | x |
| Wrote valE | x |
| Wrote valM | x |
| Wrote new PC | x |

NOP

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 1 |
| Decode Ended/Execute Started | 3 |
| Execute Ended/Memory Started | 5 |
| Memory Ended/PC Update Started/Write Back Started | 15 |
| PC Update Ended | 15 |
| Write Back Ended/Program Done | 15 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | x |
| Wrote rB | x |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | x |
| Wrote valE | x |
| Wrote valM | x |
| Wrote new PC | x |

rrmovq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 2 |
| Decode Ended/Execute Started | 4 |
| Execute Ended/Memory Started | 6 |
| Memory Ended/PC Update Started/Write Back Started | 16 |
| PC Update Ended | 16 |
| Write Back Ended/Program Done | 16 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | 3 |
| Wrote valB | x |
| Wrote valE | 4 |
| Wrote valM | x |
| Wrote new PC | 16 |

irmovq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 10 |
| Decode Ended/Execute Started | 12 |
| Execute Ended/Memory Started | 14 |
| Memory Ended/PC Update Started/Write Back Started | 24 |
| PC Update Ended | 24 |
| Write Back Ended/Program Done | 24 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | 10 |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | x |
| Wrote valE | 12 |
| Wrote valM | x |
| Wrote new PC | 24 |

rmmovq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 10 |
| Decode Ended/Execute Started | 12 |
| Execute Ended/Memory Started | 14 |
| Memory Ended/PC Update Started/Write Back Started | 24 |
| PC Update Ended | 24 |
| Write Back Ended/Program Done | 24 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | 10 |
| Wrote valP | 1 |
| Wrote valA | 11 |
| Wrote valB | 11 |
| Wrote valE | 12 |
| Wrote valM | x |
| Wrote new PC | 24 |

mrmovq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 10 |
| Decode Ended/Execute Started | 12 |
| Execute Ended/Memory Started | 14 |
| Memory Ended/PC Update Started/Write Back Started | 24 |
| PC Update Ended | 24 |
| Write Back Ended/Program Done | 24 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | 10 |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | 11 |
| Wrote valE | 12 |
| Wrote valM | 24 |
| Wrote new PC | 24 |

opq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 2 |
| Decode Ended/Execute Started | 4 |
| Execute Ended/Memory Started | 6 |
| Memory Ended/PC Update Started/Write Back Started | 16 |
| PC Update Ended | 16 |
| Write Back Ended/Program Done | 16 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | 3 |
| Wrote valB | 3 |
| Wrote valE | 4 |
| Wrote valM | 16 |
| Wrote new PC | 16 |

jxx

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 9 |
| Decode Ended/Execute Started | 11 |
| Execute Ended/Memory Started | 13 |
| Memory Ended/PC Update Started/Write Back Started | 23 |
| PC Update Ended | 23 |
| Write Back Ended/Program Done | 23 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | x |
| Wrote rB | x |
| Wrote valC | 9 |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | x |
| Wrote valE | x |
| Wrote valM | x |
| Wrote new PC | 23 |

call

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 9 |
| Decode Ended/Execute Started | 11 |
| Execute Ended/Memory Started | 13 |
| Memory Ended/PC Update Started/Write Back Started | 23 |
| PC Update Ended | 23 |
| Write Back Ended/Program Done | 23 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | x |
| Wrote rB | x |
| Wrote valC | 9 |
| Wrote valP | 1 |
| Wrote valA | x |
| Wrote valB | 10 |
| Wrote valE | 11 |
| Wrote valM | x |
| Wrote new PC | 23 |

return

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 1 |
| Decode Ended/Execute Started | 3 |
| Execute Ended/Memory Started | 5 |
| Memory Ended/PC Update Started/Write Back Started | 15 |
| PC Update Ended | 15 |
| Write Back Ended/Program Done | 15 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | x |
| Wrote rB | x |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | 2 |
| Wrote valB | 2 |
| Wrote valE | 3 |
| Wrote valM | 15 |
| Wrote new PC | 15 |

pushq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 2 |
| Decode Ended/Execute Started | 4 |
| Execute Ended/Memory Started | 6 |
| Memory Ended/PC Update Started/Write Back Started | 16 |
| PC Update Ended | 16 |
| Write Back Ended/Program Done | 16 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | 3 |
| Wrote valB | 3 |
| Wrote valE | 4 |
| Wrote valM | x |
| Wrote new PC | 16 |

popq

| Event | Clock Cycle |
| --- | --- |
| Fetch Started | 0 |
| Fetch Ended/Decode Started | 2 |
| Decode Ended/Execute Started | 4 |
| Execute Ended/Memory Started | 6 |
| Memory Ended/PC Update Started/Write Back Started | 16 |
| PC Update Ended | 16 |
| Write Back Ended/Program Done | 16 |
| Wrote icode | 1 |
| Wrote ifun | 1 |
| Wrote rA | 2 |
| Wrote rB | 2 |
| Wrote valC | x |
| Wrote valP | 1 |
| Wrote valA | 3 |
| Wrote valB | 3 |
| Wrote valE | 4 |
| Wrote valM | 16 |
| Wrote new PC | 16 |

Fetch

| halt | icode:ifun ← M1[PC]  valP ← PC + 1 |
| --- | --- |
| nop | icode:ifun ← M1[PC]  valP ← PC + 1 |
| Rrmovq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valP ← PC + 2 |
| rmmovq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valC ← M8[PC+2]  valP ← PC + 10 |
| irmovq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valC ← M8[PC+2]  valP ← PC + 10 |
| Mrmovq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valC ← M8[PC+2]  valP ← PC + 10 |
| OPq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valP ← PC + 2 |
| jxx | icode:ifun ← M1[PC] |
| call | icode:ifun ← M1[PC]  valC ← M8[PC+1]  valP ← PC + 9 |
| ret | icode:ifun ← M1[PC]  valP ← PC + 1 |
| pushq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valP ← PC + 2 |
| popq | icode:ifun ← M1[PC]  rA:rB ← M1[PC+1]  valP ← PC + 2 |

Decode

| halt | x |
| --- | --- |
| nop | x |
| Rrmovq | valA ← R[rA] |
| irmovq | x |
| rmmovq | valA ← R[rA]  valB ← R[rB] |
| mrmovq | valB ← R[rB] |
| Opq | valA ← R[rA]  valB ← R[rB] |
| jxx | x |
| call | valB ← R[RSP] |
| ret | valA ← R[RSP]  valB ← R[RSP] |
| pushq | valA ← R[rA]  valB ← R[RSP] |
| popq | valA ← R[RSP]  valB ← R[RSP] |

Execute

| halt | cpu.stat = HLT |
| --- | --- |
| nop | x |
| Rrmovq | valE ← valA |
| irmovq | valE ← valC |
| rmmovq | ​​valE ← valB + valC |
| mrmovq | valE ← valB + valC |
| Opq | valE ← valB OP valA  set flag |
| jxx | Cnd ← Cond(CC,ifun) |
| call | valE ← valB - 8 |
| ret | valE ← valB + 8 |
| pushq | ​​valE ← valB - 8 |
| popq | valE ← valB + 8 |

Memory

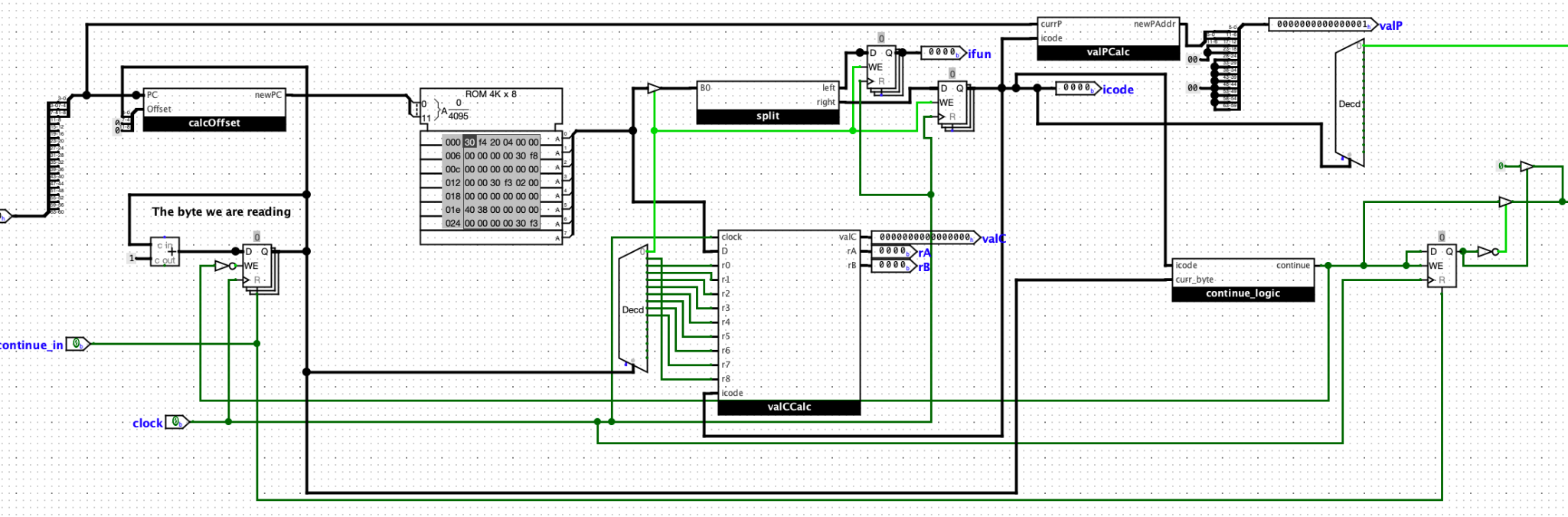
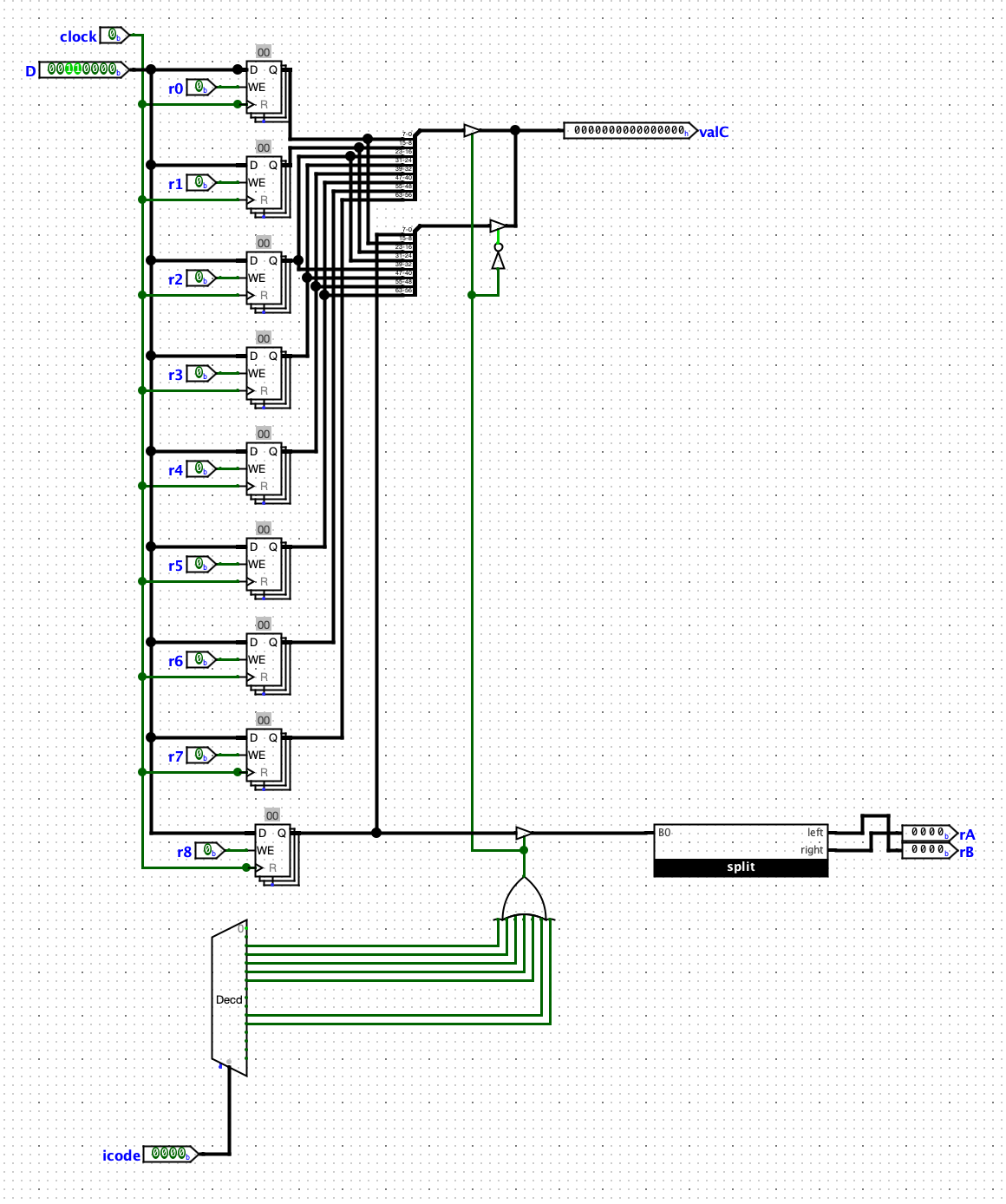
| halt | x |
| --- | --- |
| nop | x |
| rrmovq | x |
| irmovq | x |
| rmmovq | M8[valE] ← valA |
| mrmovq | valM ← M8[valE] |
| Opq | x |
| jxx | x |
| call | M8[valE] ← valP |
| ret | valM ← M8[valA] |
| pushq | M8[valE] ← valA |
| popq | valM ← M8[valA] |

PC Update

| halt | PC ← 0 |
| --- | --- |
| nop | PC ← valP |
| rrmovq | PC ← valP |
| irmovq | PC ← valP |
| rmmovq | PC ← valP |
| mrmovq | PC ← valP |
| opq | PC ← valP |
| jxx | PC ← Cnd ? valC:valP |
| call | PC ← valC |
| ret | PC ← valM |
| pushq | PC ← valP |
| popq | PC ← valP |

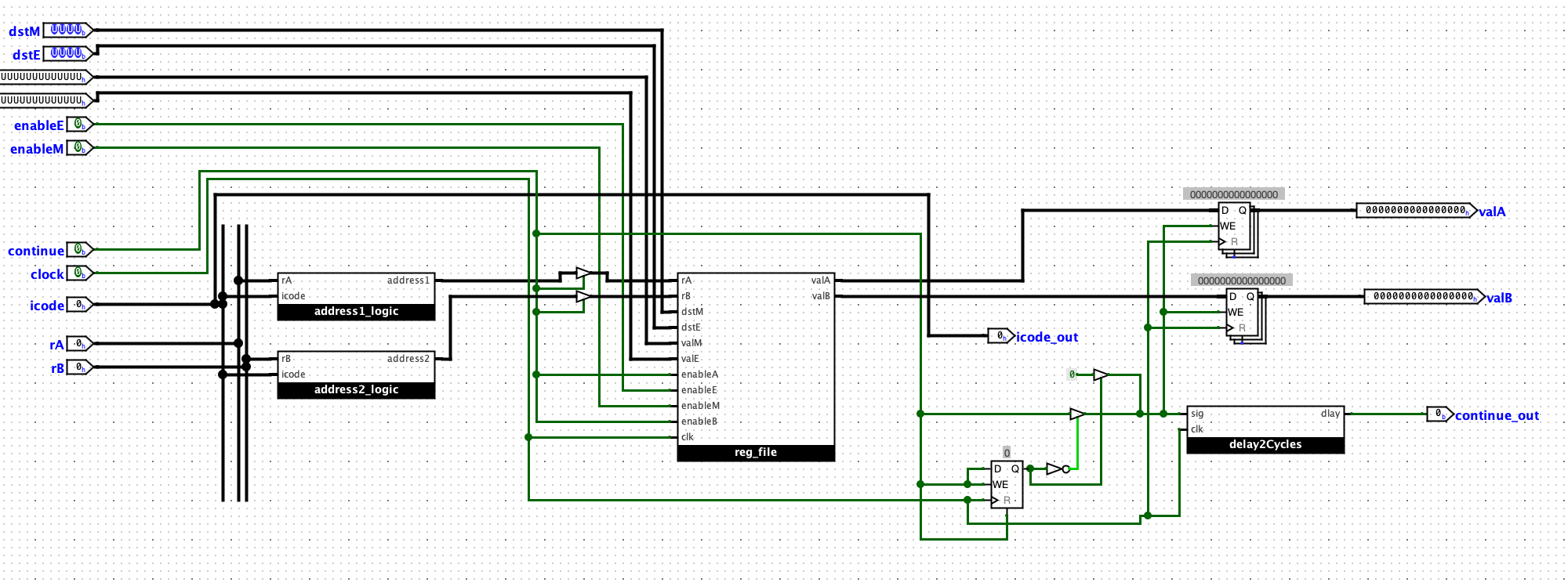
Fetch: Jade

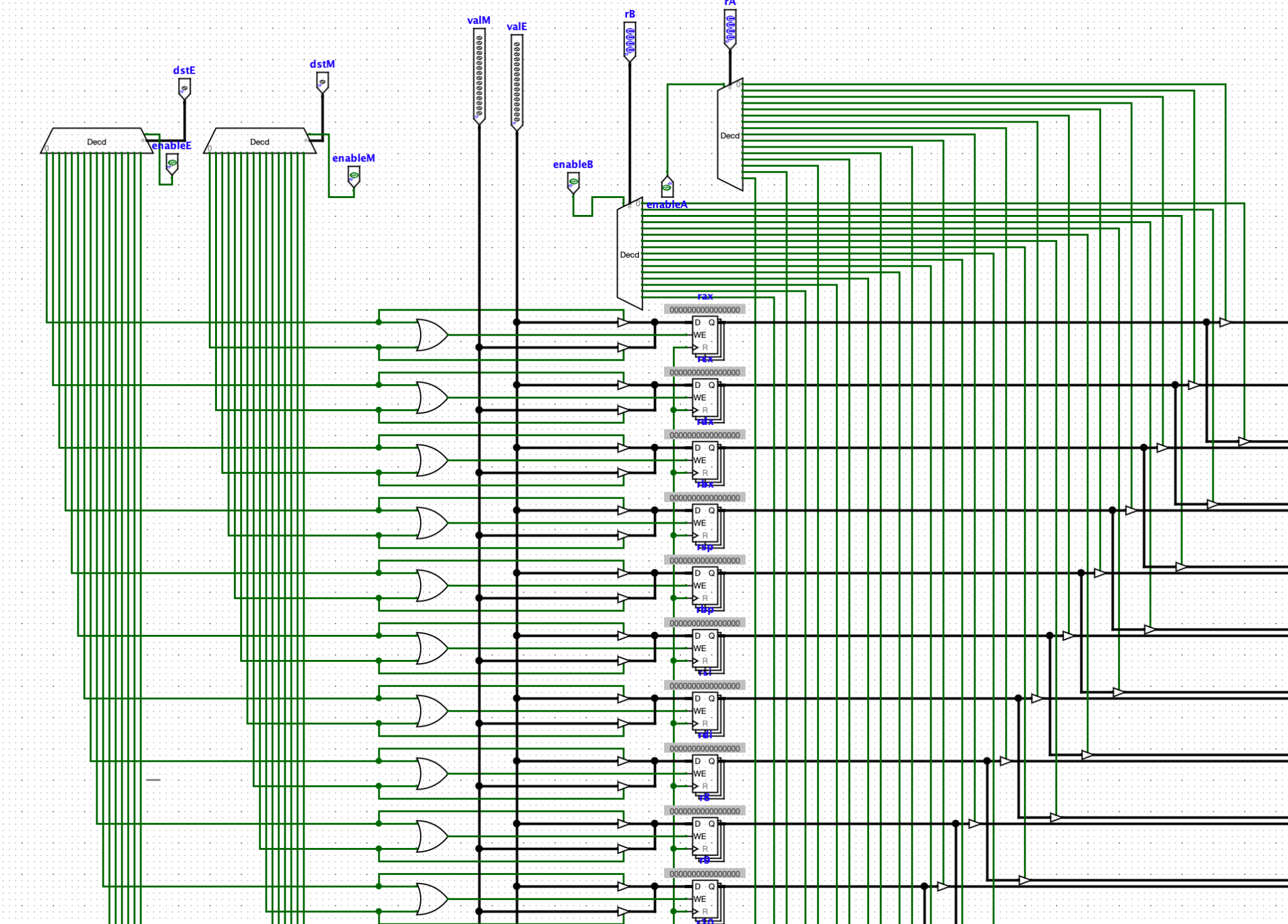
The Fetch cycle retrieves data from the memory and turns it into meaningful instructions that will be used for execution in later steps. The first byte read in is separated into icode (4 bits which are all unique for each instruction) and ifun which will be useful for the next step. Next the byte we are reading in currently is decoded and from that rA, rB and valC are determined (shown below). If we need to continue reading in data the process will continue and it will alter valC. PC is updated, and a continue bit will be 0 once the fetch stage is done operating.



Decode: Garrett

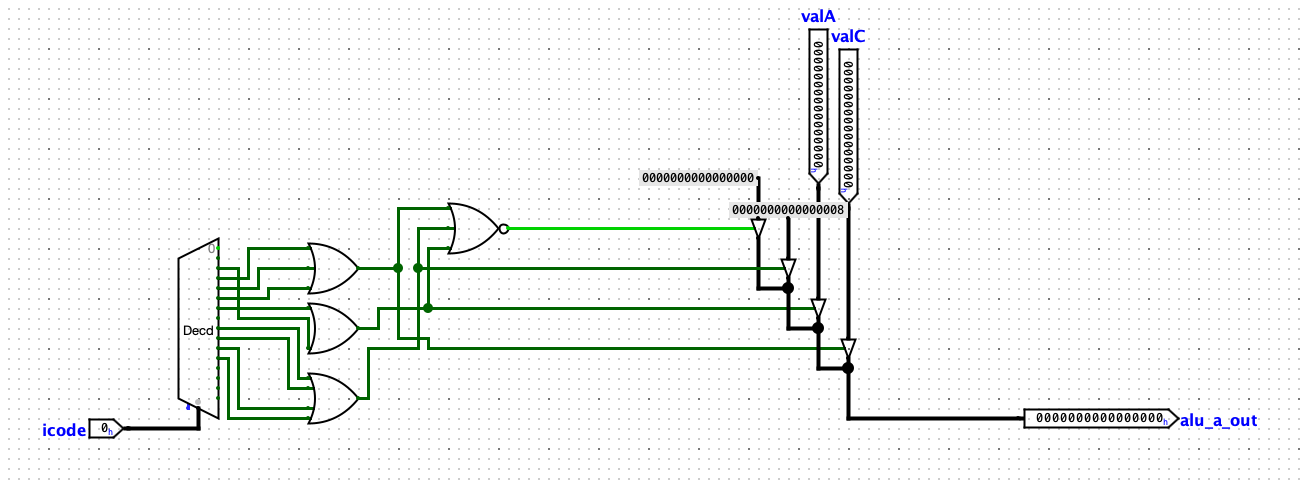
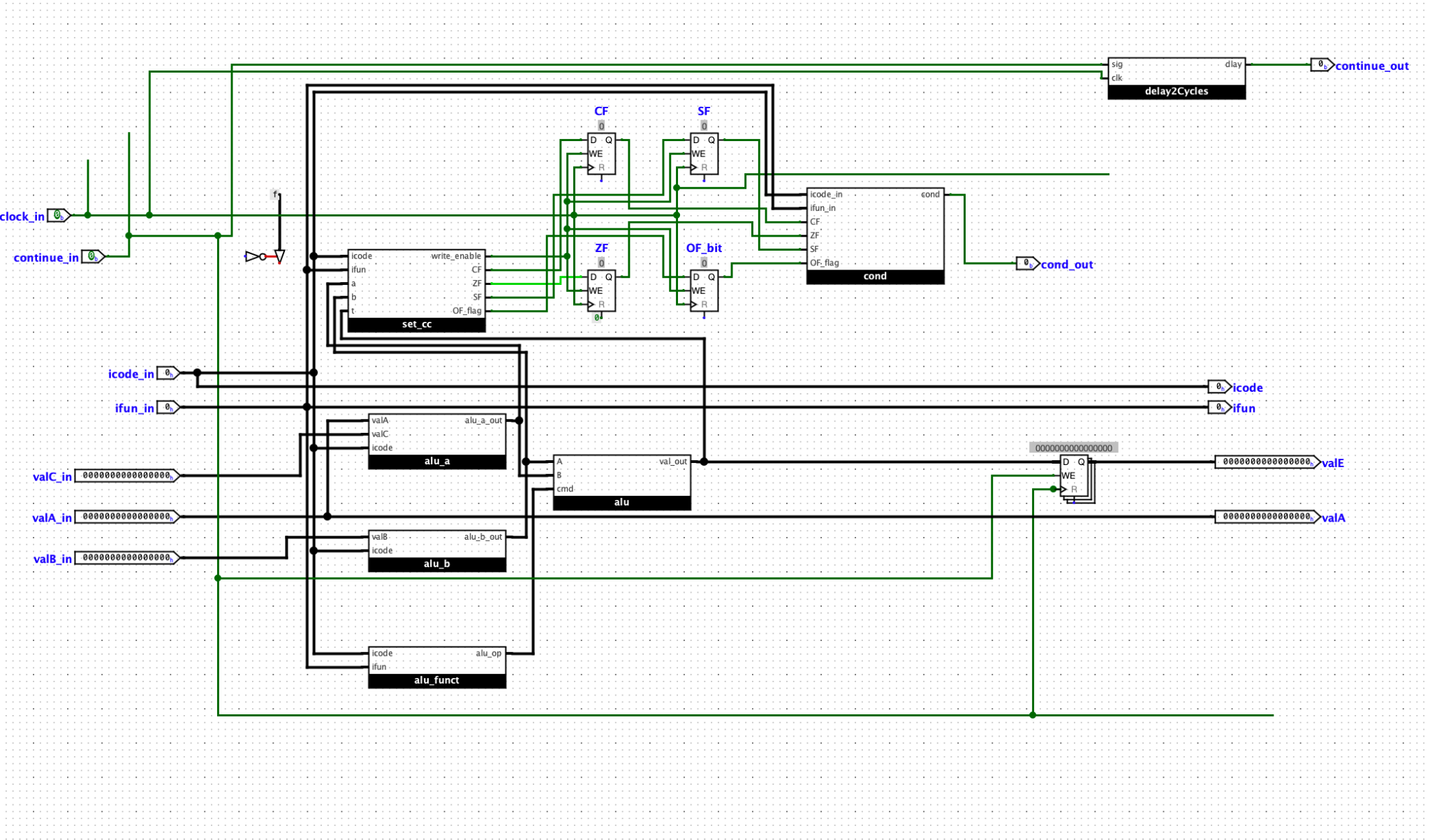
Decode contains a register file that values can be written to and read from. The destinations of these values are determined by the value of icode, which was retrieved from fetch. Initially, the values within registers rA and rB (also from fetch) are read from the register file and written to valA and valB, respectively. During the write-back stage, the register is used again to write new values into the register from the newly updated valM and valE.





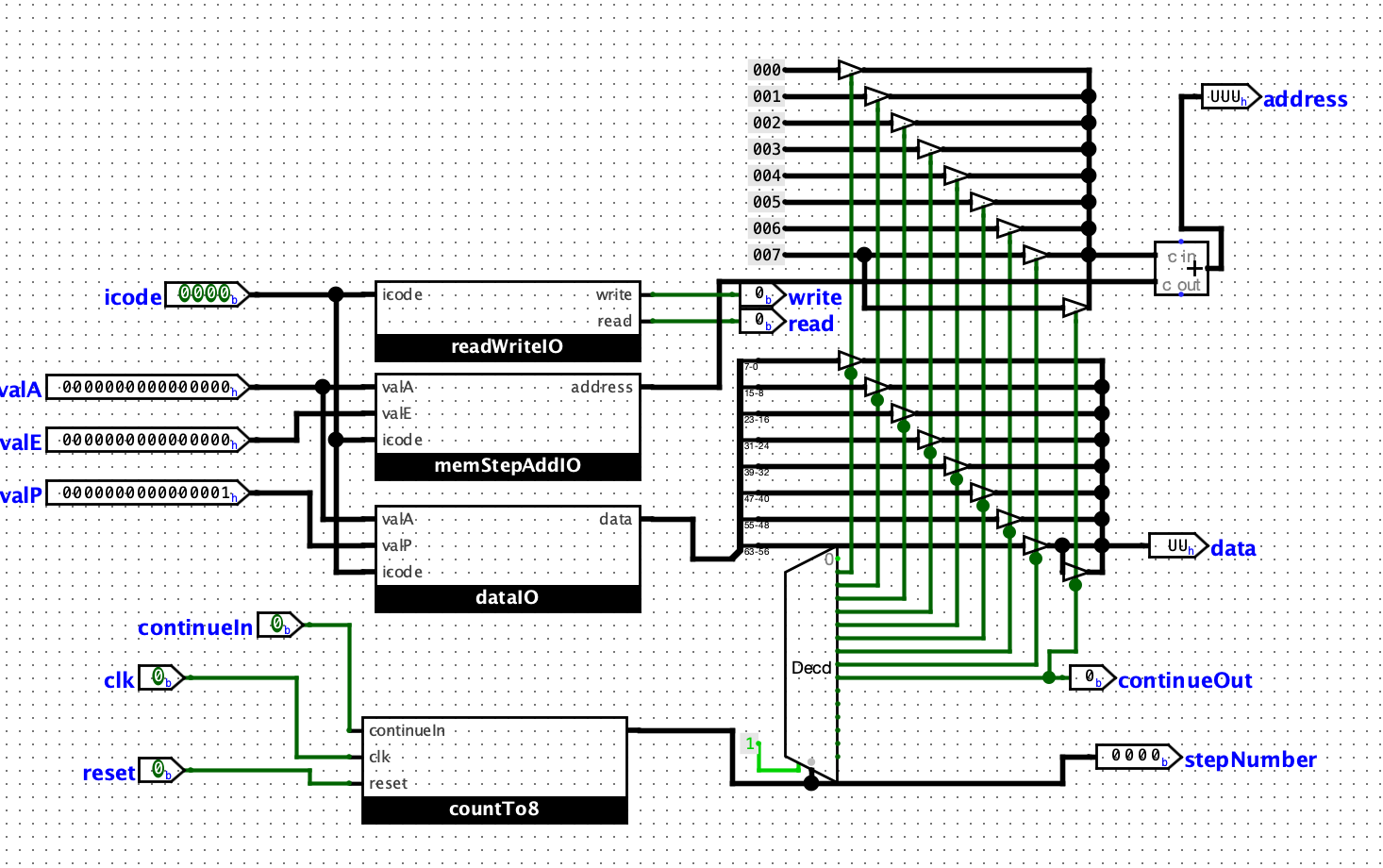
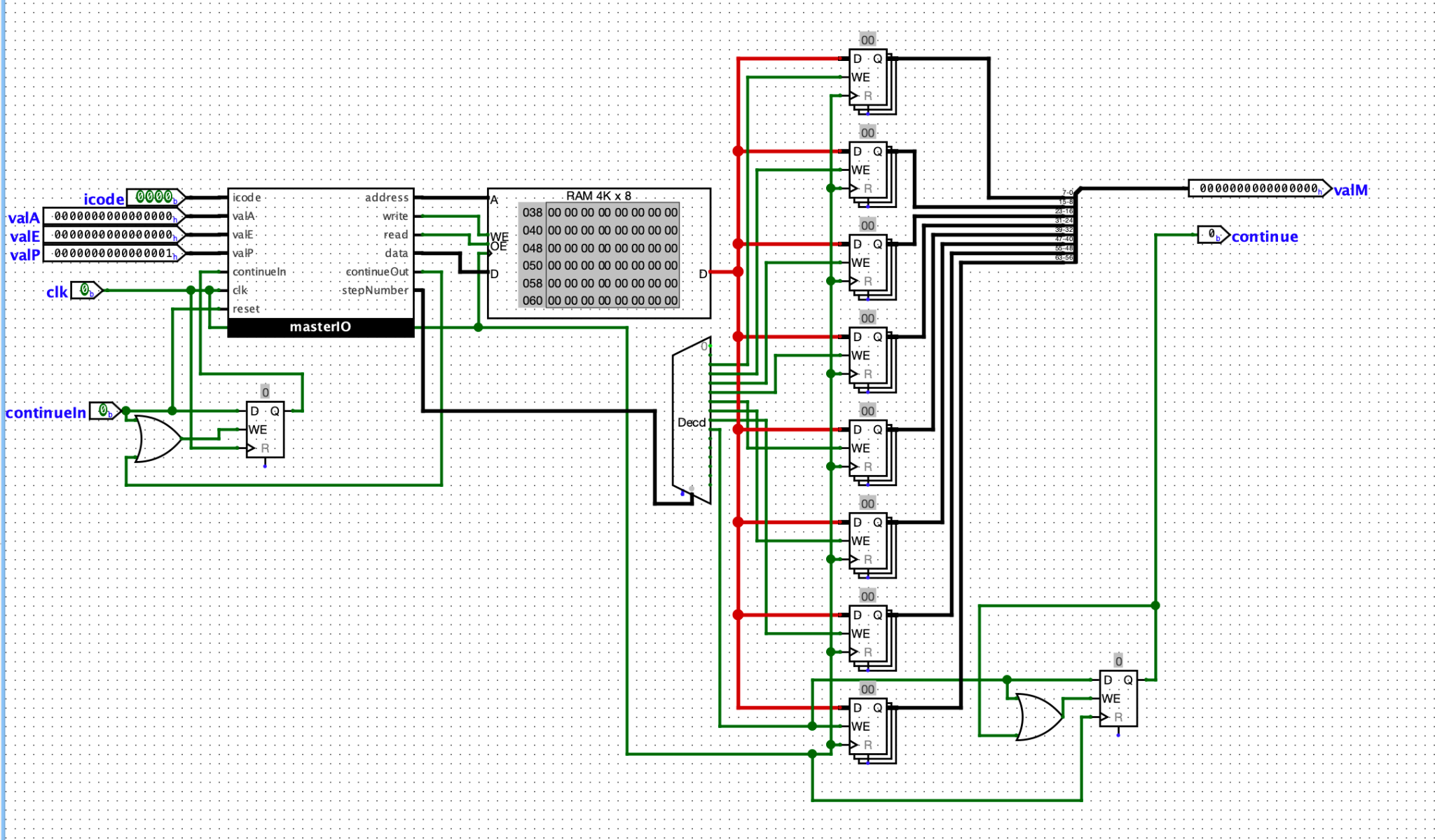
Execute: Jerry

The execute step was responsible for passing values through and doing any necessary computations. This included deciding what operations were to be performed, picking the correct operands, and performing the operations within many separate ALUs. There was also a conditional bit that had to be set here as a flag for future steps, such as the jump step. Below is a diagram of the overall structure of the execute step, as well as a screenshot of one of the deciding logic units to pick operands that were later used to compute valE.



Memory: Ariela

The memory stage of our processor is responsible for taking in the values it receives from the prior stage, and with that information, deciding whether to read or write the RAM. For read and write operations, we had to process one bit at a time from the register because the RAM cannot handle 64-bit data being passed in. For the address, we just used the 12 least significant bits. Below is a screenshot of the overall layout of the memory step, as well as some of the top-level abstractions we used to decide what to do and which values to use.



Write-Back and PC Update: Everyone

The last step of this project was connecting everything together and implementing the

PC update and write back. Write back was a fairly simple process since the register file was already built in decode, and we just had to pick the correct value(s) to route back to the register file and update. PC update was probably the easiest step of the process, because it just used a mux to select between three different values for PC and passed it as an output to a register we had at the beginning of our processor.

