### Part A. ISA Intro

##### 0. Introduction

**Name:**  KISS, Keep It Short and Sweet

**Overall philosophy:**

Using just 7-bit length machine code, we try to achieve the full function of both program1 and program2, we also try to keep our ISA and program code as succinct as possible.

**Specific goals strived for and achieved:**

We achieved all the functions required for both program1 and program2.

We use a lot of skills both in software and hardware, such as, we use single instruction to calculate the number of same bits between two registers, as well as to calculate the product of 6 and register using shift. Also, we don't need 'set less than' instruction, all the branches are handled by our 'jpu' instruction. In addition, we store all the immediate number we will use into MUX, so even the length of machine code is limited, we can still use some big immediate number for our 'init' and 'jpu'.

##### 1. Instruction list

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction** | **PC** | **Coding** | **Functionality** | **Example** | |
| init Rx, imm | PC++ | 000 xx ii | Rx = MUX[imm]  Imm will go into a MUX to select specific hardwrited number (0,1,6,108) | init R1, 6 | 000 01 10 |
| ld Rx, Ry | PC++ | 001 xx yy | Rx = Mem[Ry] | ld R3, R2 | 001 11 10 |
| st Rx, Ry | PC++ | 010 xx yy | Mem[Ry] = Rx | st R0, R3 | 010 00 11 |
| add Rx, Ry | PC++ | 011 xx yy | Rx = Rx + Ry | add R1, R1 | 011 01 01 |
| jpu1 Rx,Ry,imm | if Rx < Ry:  PC = MUX(imm)  else:  PC++ | 100 x y ii | x = 0 for R0  x = 1 for R1  y = 0 for R2  y = 1 for R1  Imm number will go into a MUX to select specific jumps | jpu1 R0, R2, 6 | 100 0 0 01 |
| jpu2 Rx,Ry,imm | if Rx < Ry:  PC = MUX(imm)  else:  PC++ | 101 x y ii | x = 0 for R2  x = 1 for R3  y = 0 for R0  y = 1 for R1 | jpu2, R1, R3, 8 | 101 1 1 01 |
| subR3 Rx | PC++ | 11100 xx | R3 = R3 - Rx | subR3 R2 | 11100 10 |
| inc Rx | PC++ | 11101 xx | Rx = Rx + 1 | Inc R3 | 11101 11 |
| R3x6 | PC++ | 1111110 | R3 = R3 \* 6 | R3x6 | 11111110 |
| score | PC++ | 1111111 | R3 = the match score of R3 and R1.  This function is done using logic circuit. | Score | 1111111 |

##### 2. Register Design

|  |  |
| --- | --- |
| Register Name | Number |
| R0 | 00 |
| R1 | 01 |
| R2 | 10 |
| R3 | 11 |

##### 3. Control Flow

Since there are total 7 branches used in our Program1 and Program2, the instruction address of all these branches are constant. We save all these address into MUXs, and use the immediate number from machine code to directly select values from MUX.

Accordingly, there is no need for us to calculate the target addresses.

##### 4. Memory Model

#### 4.1 Data Memory

* 16-bit double-byte addressable
* 128memory units in total
* using 7-bit address.

|  |  |
| --- | --- |
| Address | Memory |
| 000 0000 | Mem[0] |
| 000 0001 | Mem[1] |
| ... | ... |
| 111 1111 | Mem[127] |

#### 4.2 Instruction Memory

* 8-bit byte addressable, PC is initialized at 0
* 64 memory units in total
* using 6-bit address.

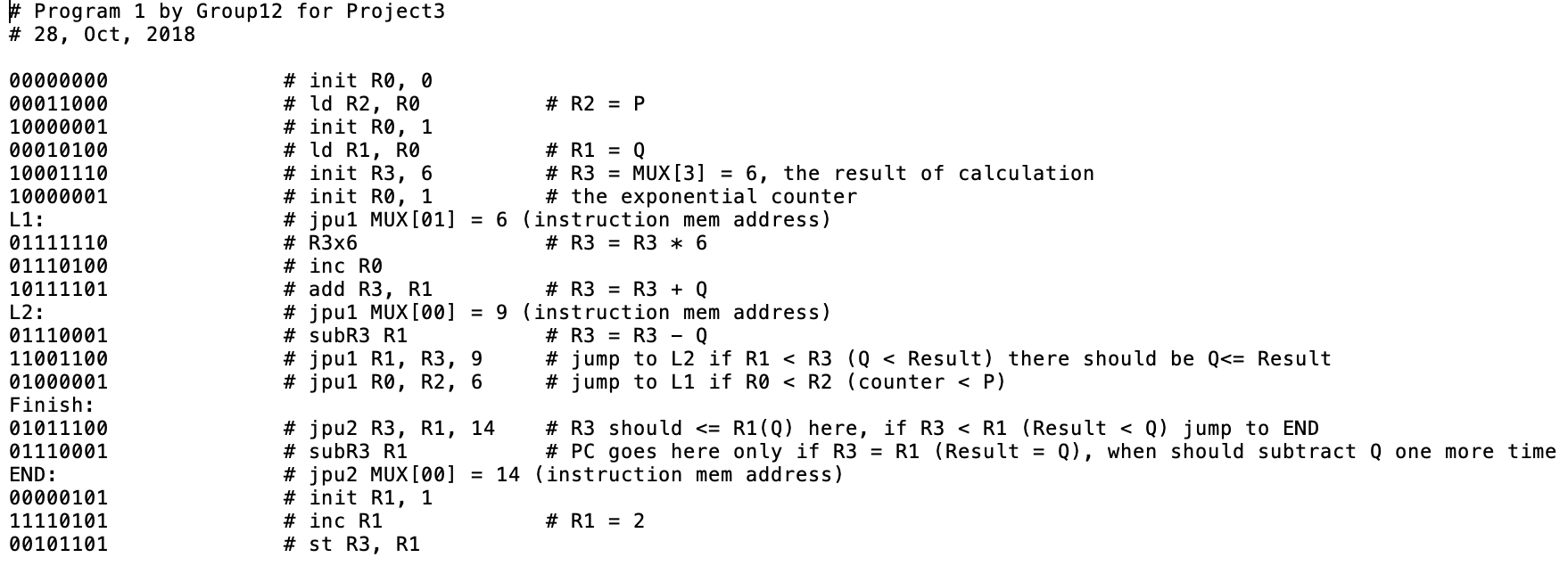
|  |  |
| --- | --- |
| Address | Memory |
| 00 0000 | Mem[0] |
| 00 0001 | Mem[1] |
| ... | ... |
| 11 1111 | Mem[63] |

### Part B. Answers to Questions

### Part C. Software Package

### Part D. ISA package

##### 1. Algorithms & Machine Code



##### 

##### 2. Pattern C & Pattern D

### 

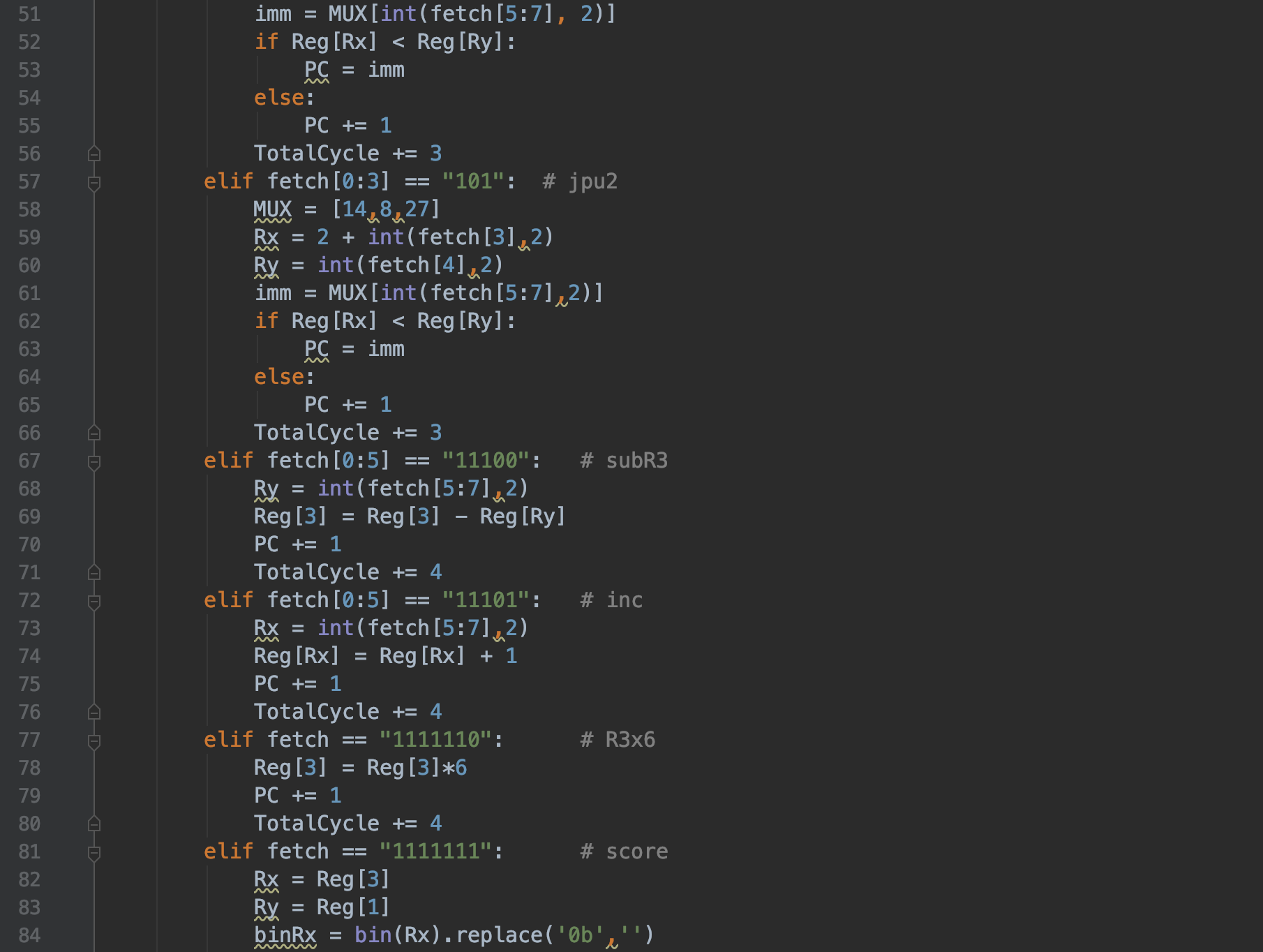
Pattern C:

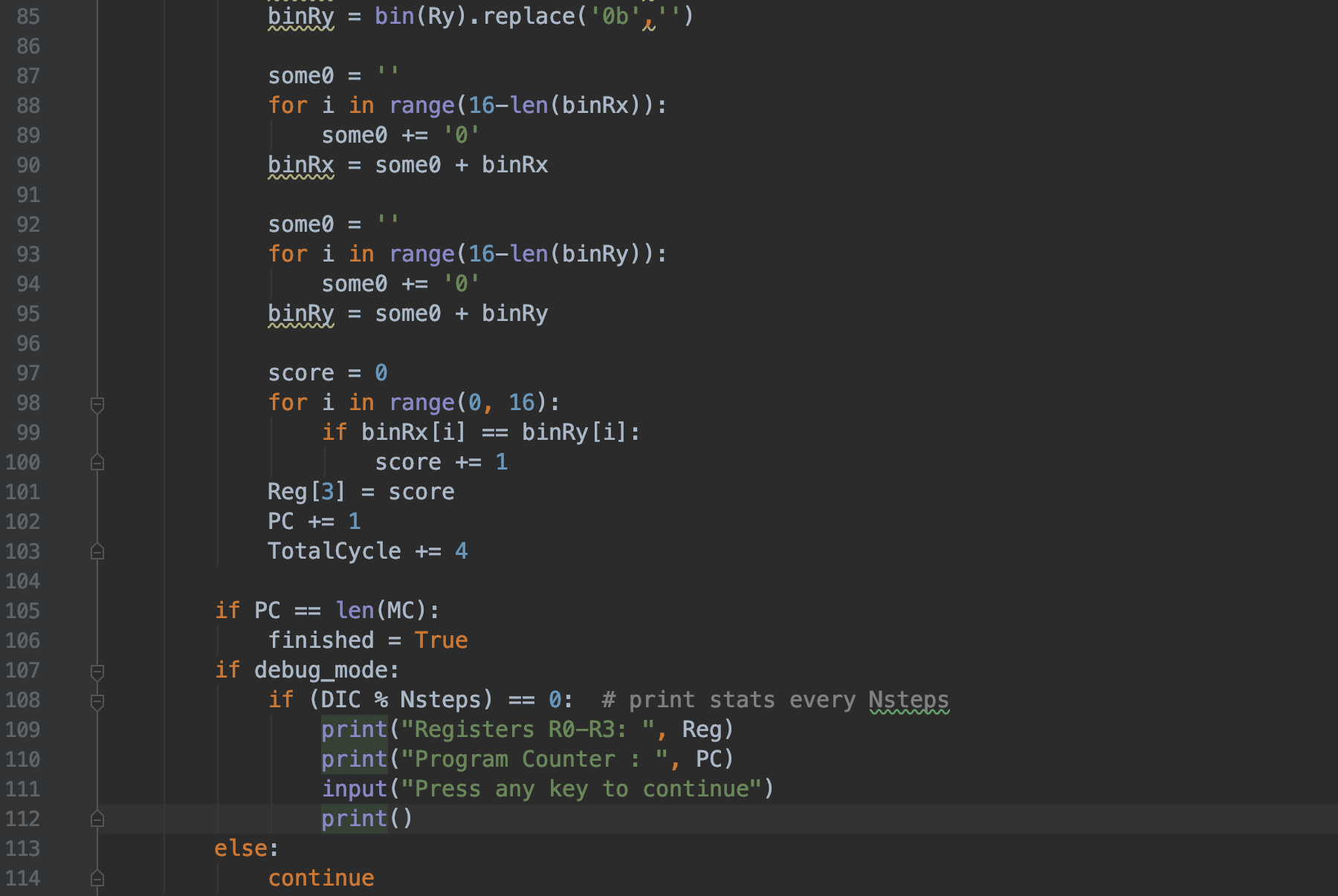
* For program1, Pattern C makes sure that in 6^P % Q, even if the remainder is 0, our program can still get the answer correctly.
* For program2, target pattern is all 0, the patterns need to be graded are random numbers produce by a Binomial distribution with p = 0.5.

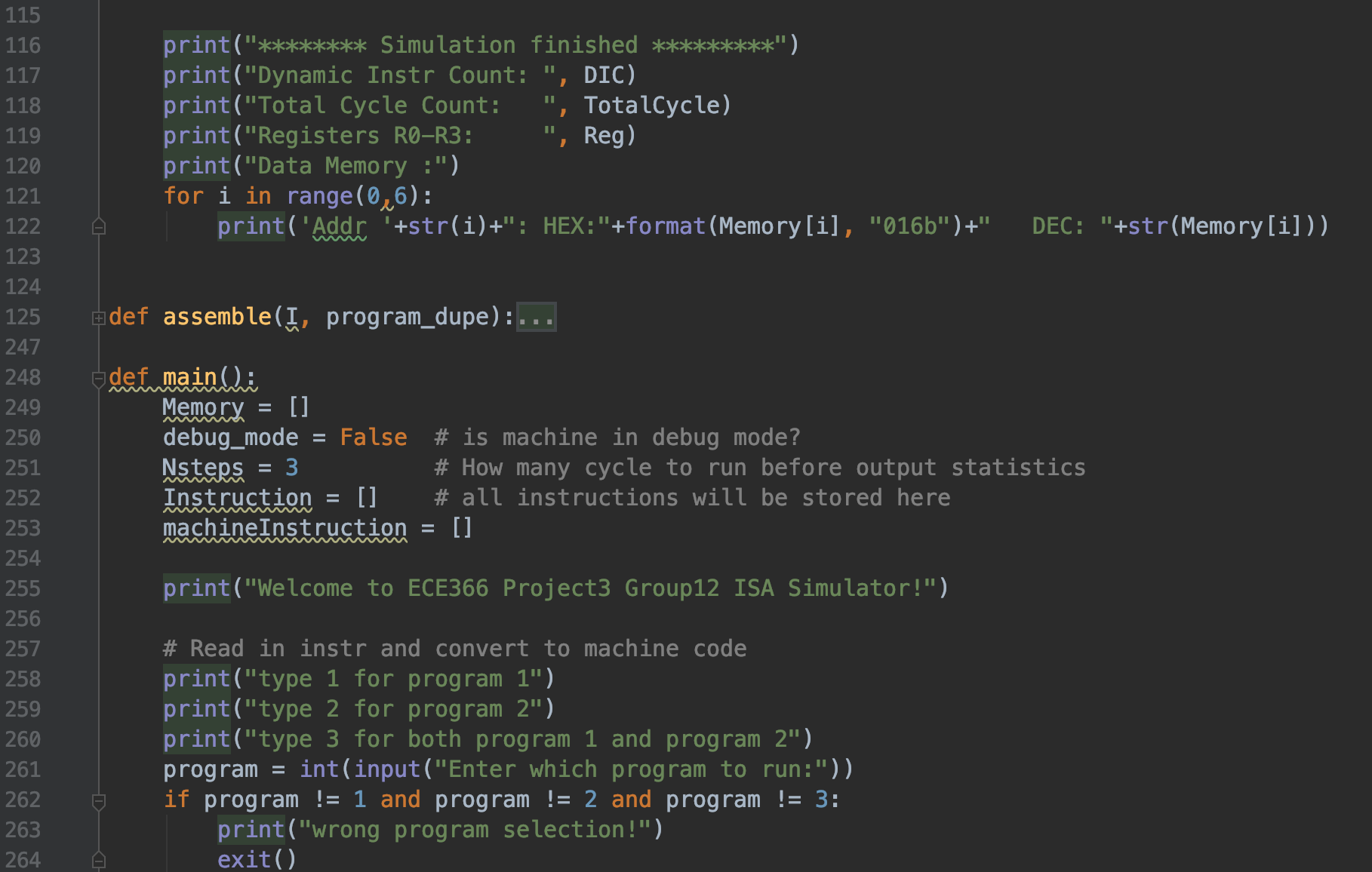
Pattern D:

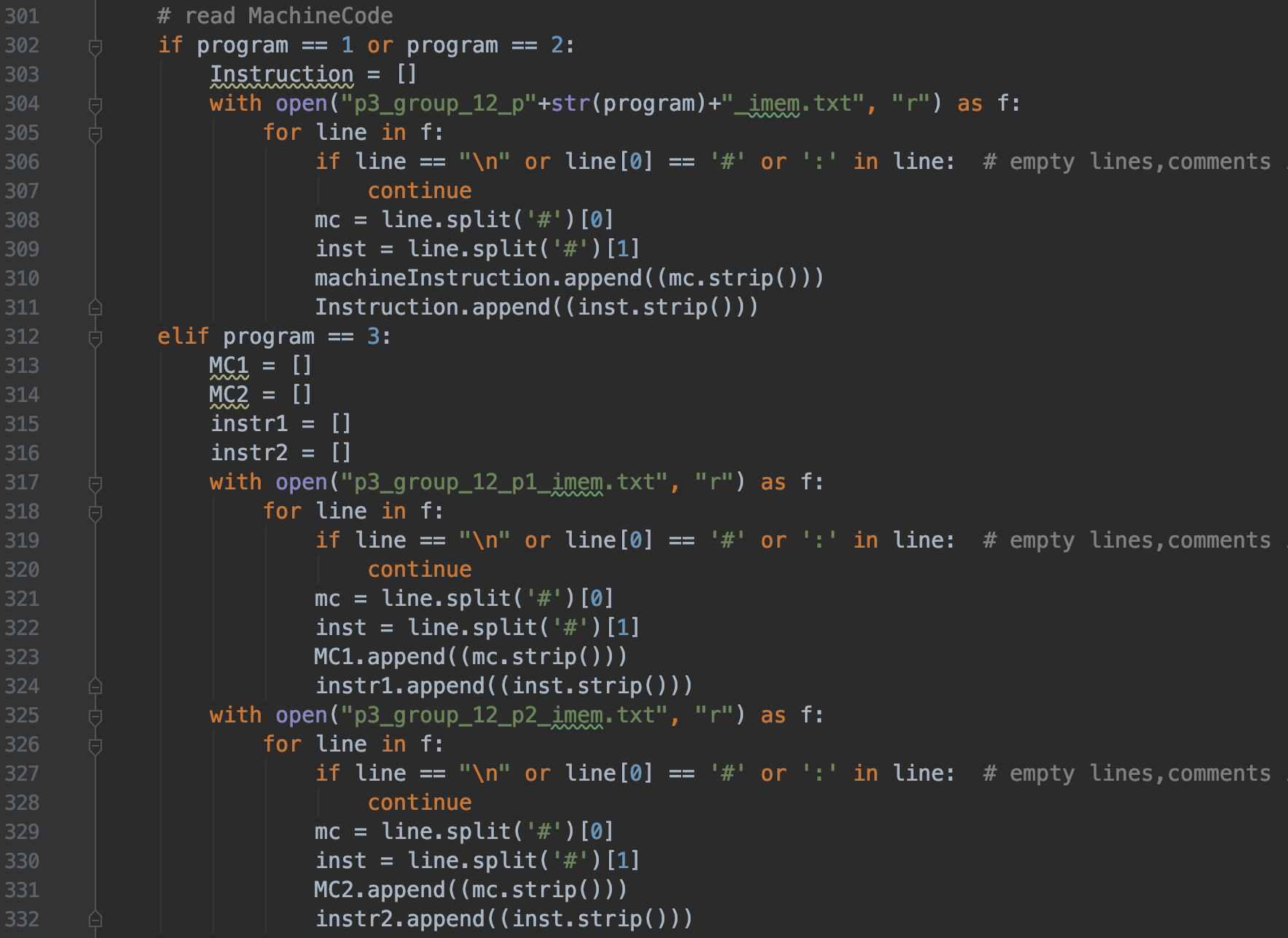
* All the numbers in Pattern D are random numbers produced by Binomial distribution with p = 0.667, to make sure that our both programs can work on arbitrary data.

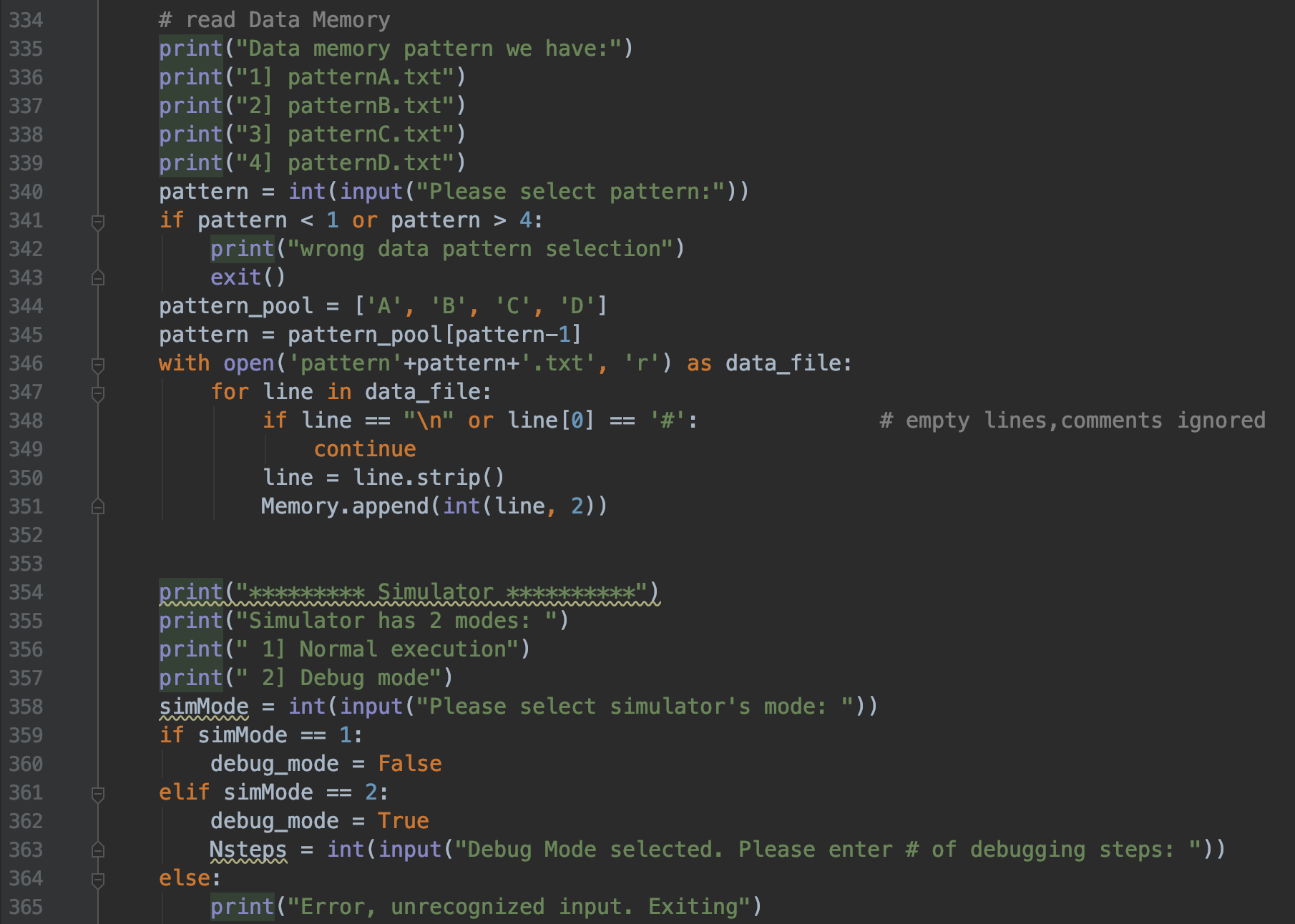
##### 3. Python Simulator Code

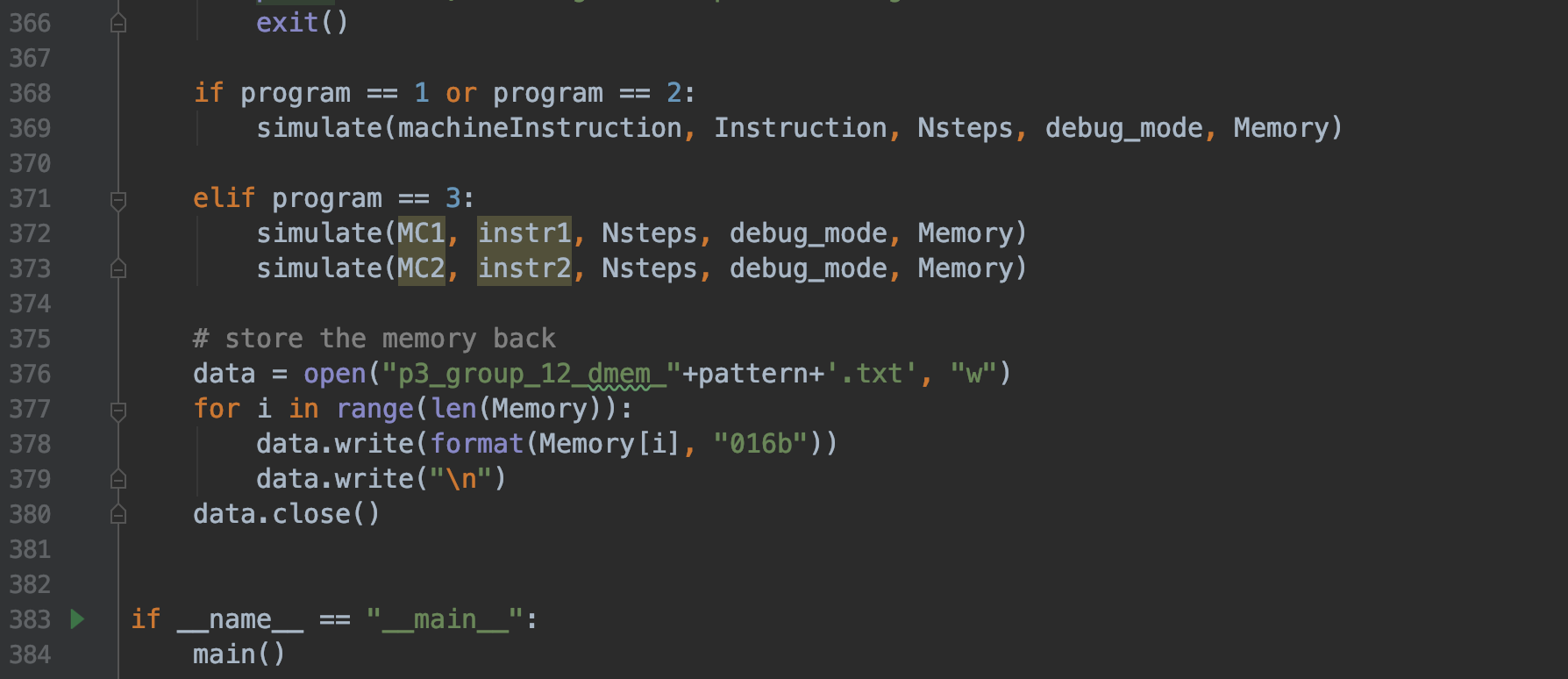






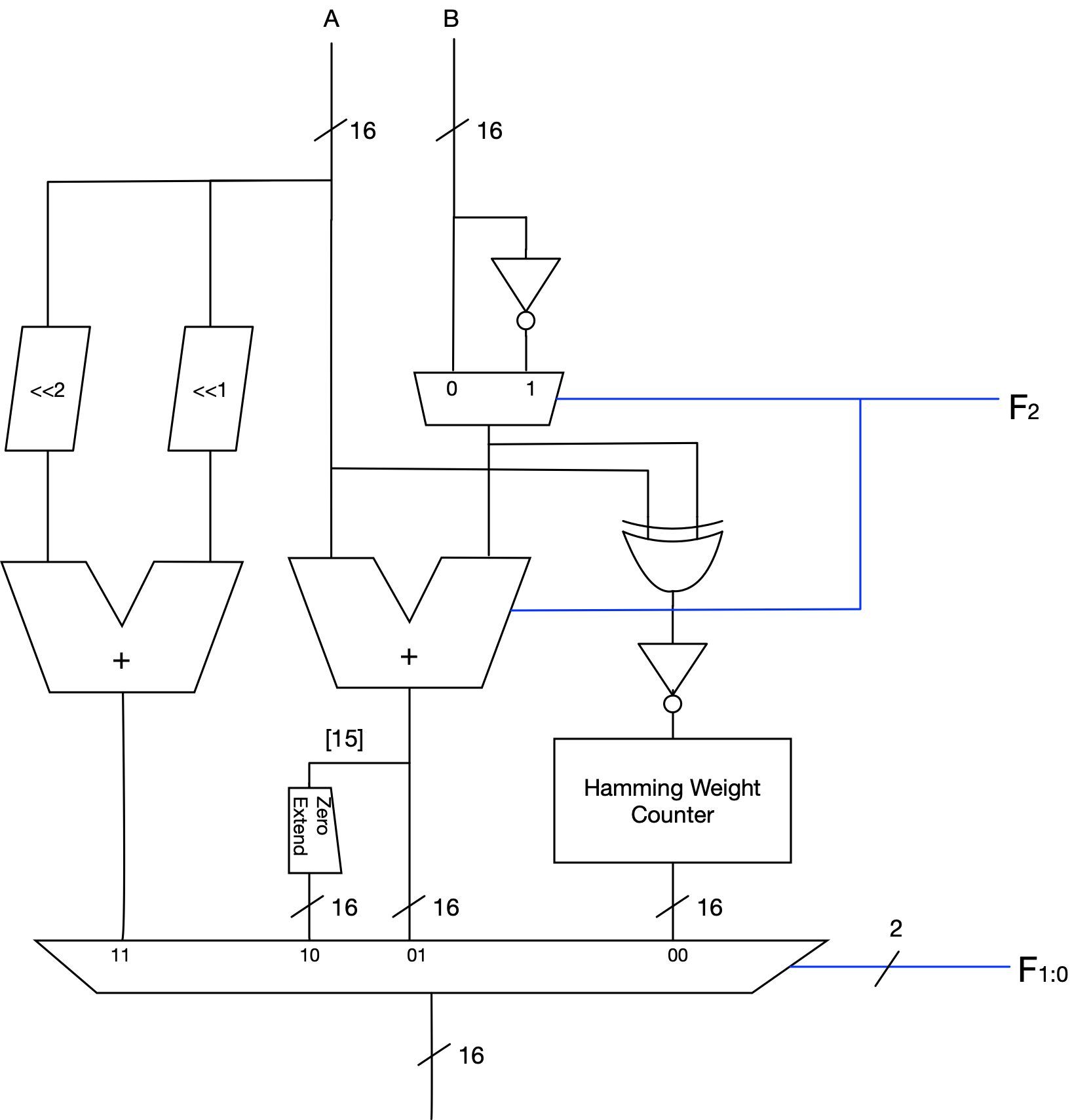




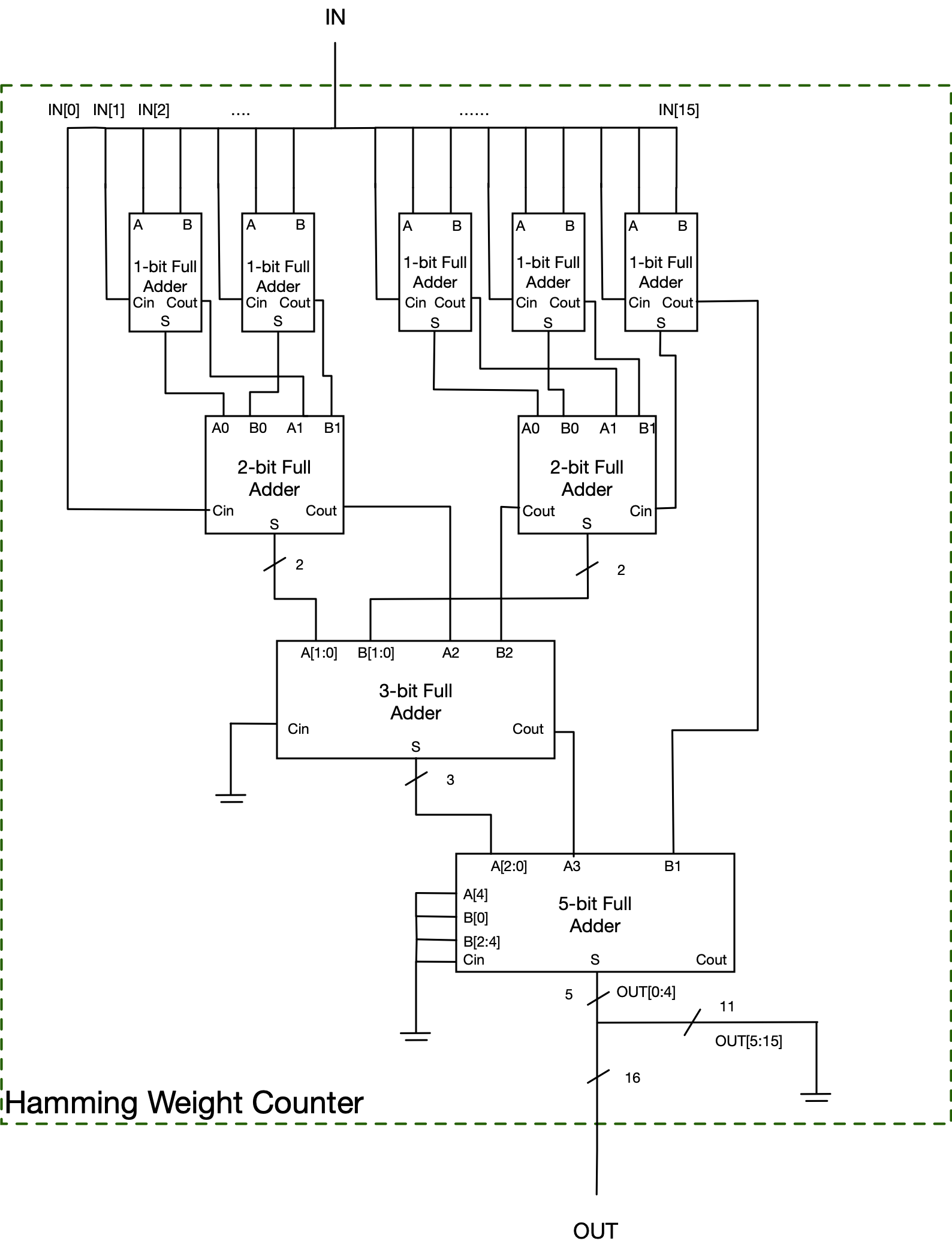


### 4. Hardware Schematics

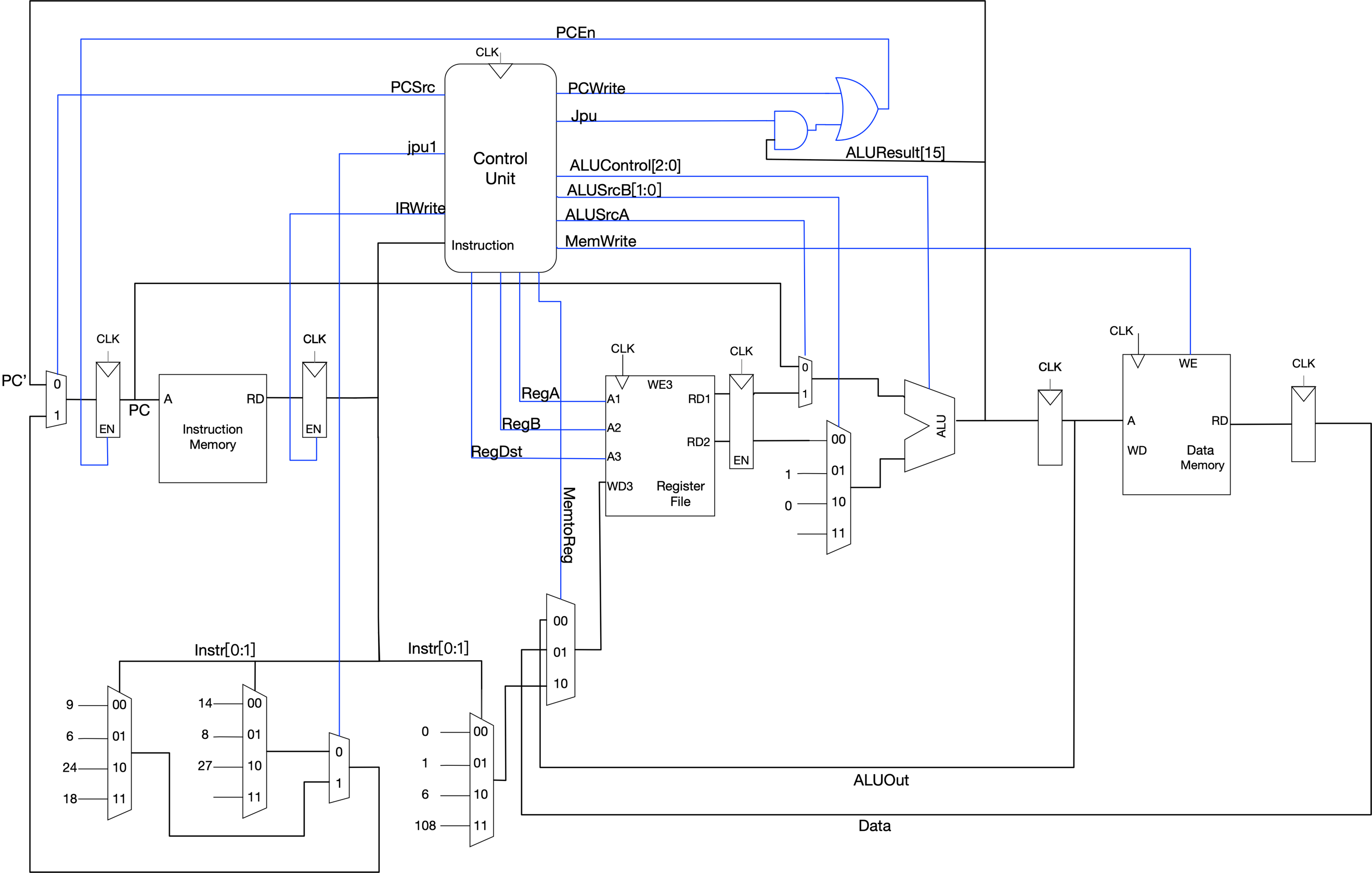
###### 4.a. ALU Schematic



**Hamming Weight Counter**



###### 4.b. CPU Datapath



###### 4.c. FSM for multi-cycle implementation

