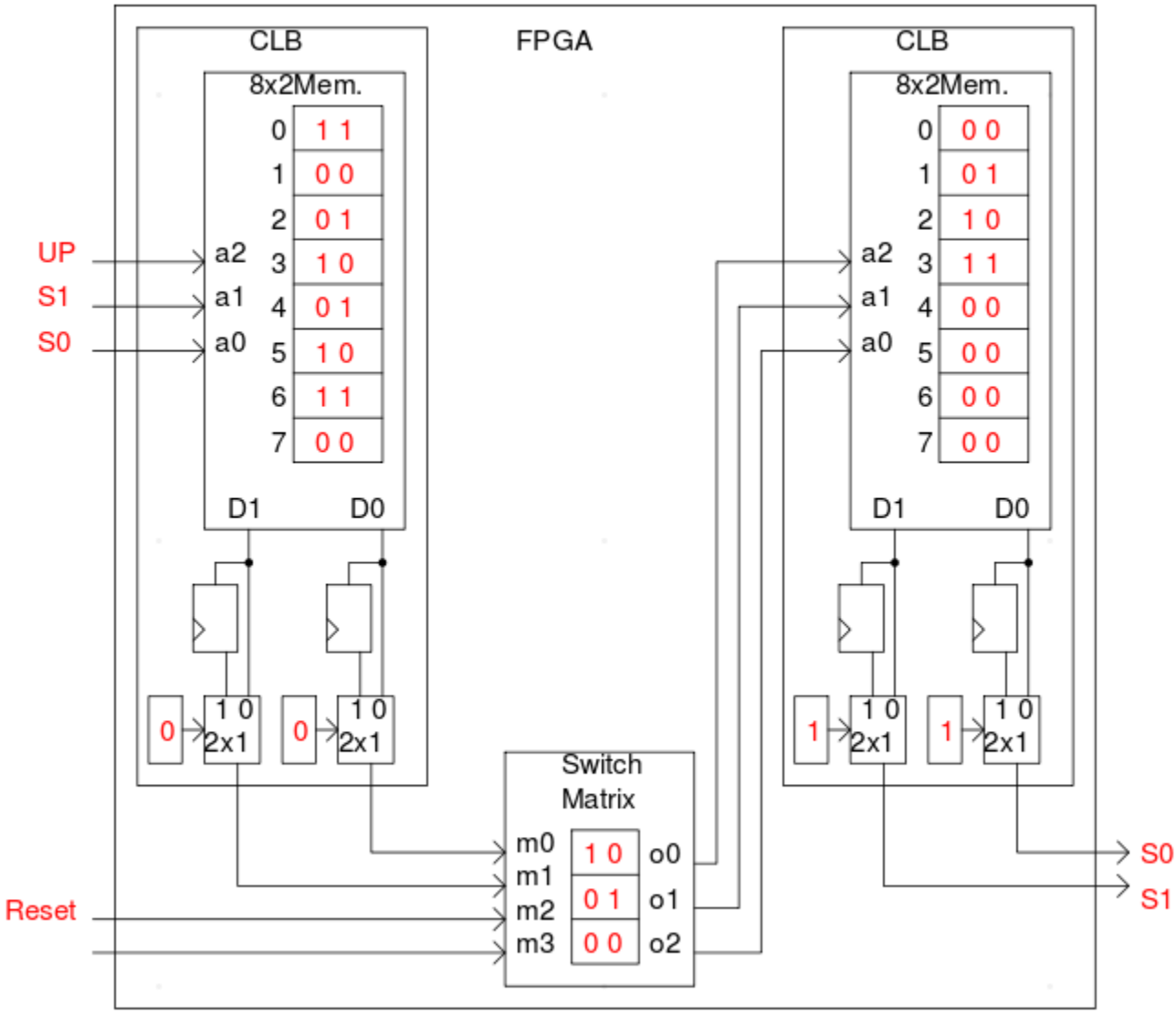
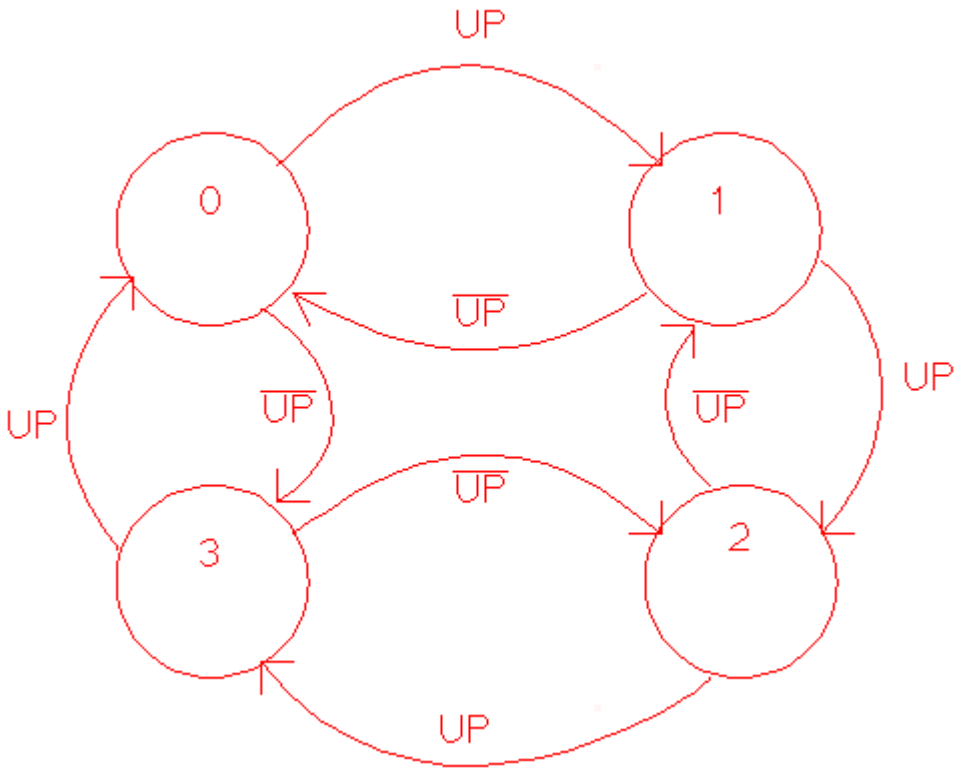
1. Design a 2 bit up/down counter with reset using the FPGA circuit below.



There are 4 inputs and 2 outputs for the circuit. Since the CLB LUT's have only 3 inputs, the largest truth table size for the circuit is 8 entries. However, there are two LUT's in the FPGA. What would normally be a 4 variable truth table must be split into two 3 variable truth tables. The first truth table implements an up/down counter without reset. The state diagram is below. The second LUT implements the reset.



UP	S1	S0	D1	D0
0	0	0	1	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

RESET	D1	D0	N1	N0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

The switch matrix is set so the Reset signal is routed to the a2 input of the second LUT. T D0 and D1 output of the first LUT is routed to the a0 and a1 input of the second LUT. The flip-flops in the first LUT are bypassed. The flip-flops in the second LUT are used.

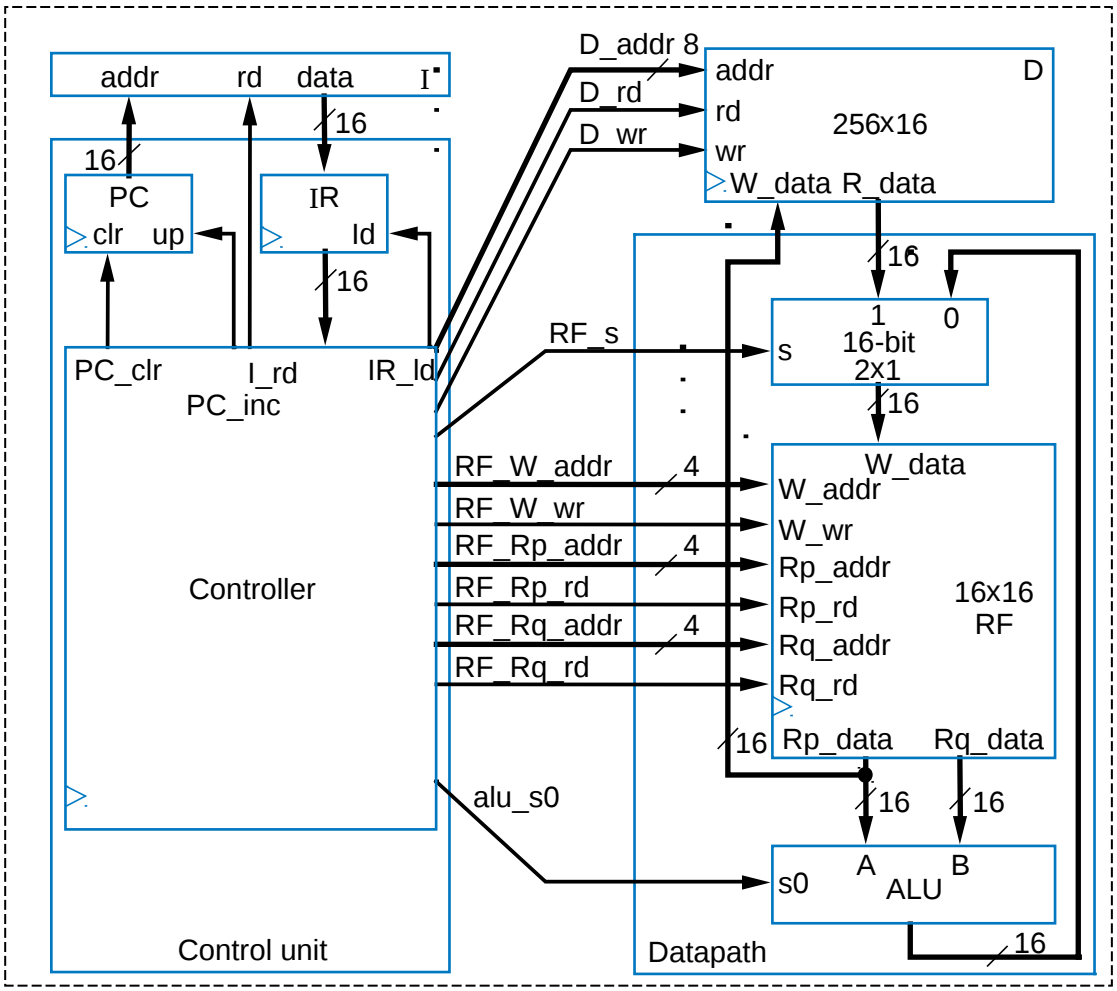
2. Using the 3 instruction processor, with 3 additional instructions for multiply, divide, and subtract, write the code segment that calculates a set of terms in the cosine and sine function in the Taylor series from problem 1. Your code must calculate variables for use the next time through the code. As an example, if you are calculating $X^2/2!$ and $X^3/3!$ the next time through your code it will calculate $X^4/4!$ and $X^5/5!$. You should NOT write instructions that change the flow of the program (no jmp instructions). Assume the ALU from the processor does floating point arithmetic. See the last page of the exam for instruction mnemonics and a processor diagram.

Part A. List name, meaning, and location of constants and variables you will need in memory and the register file. Write these as comments. A comment starts with // and goes until the end of the line. It is possible a location may have multiple meanings in your code. If you need a constant, you may assume the constant is in data memory or the register file as long as your code doesn't overwrite it.

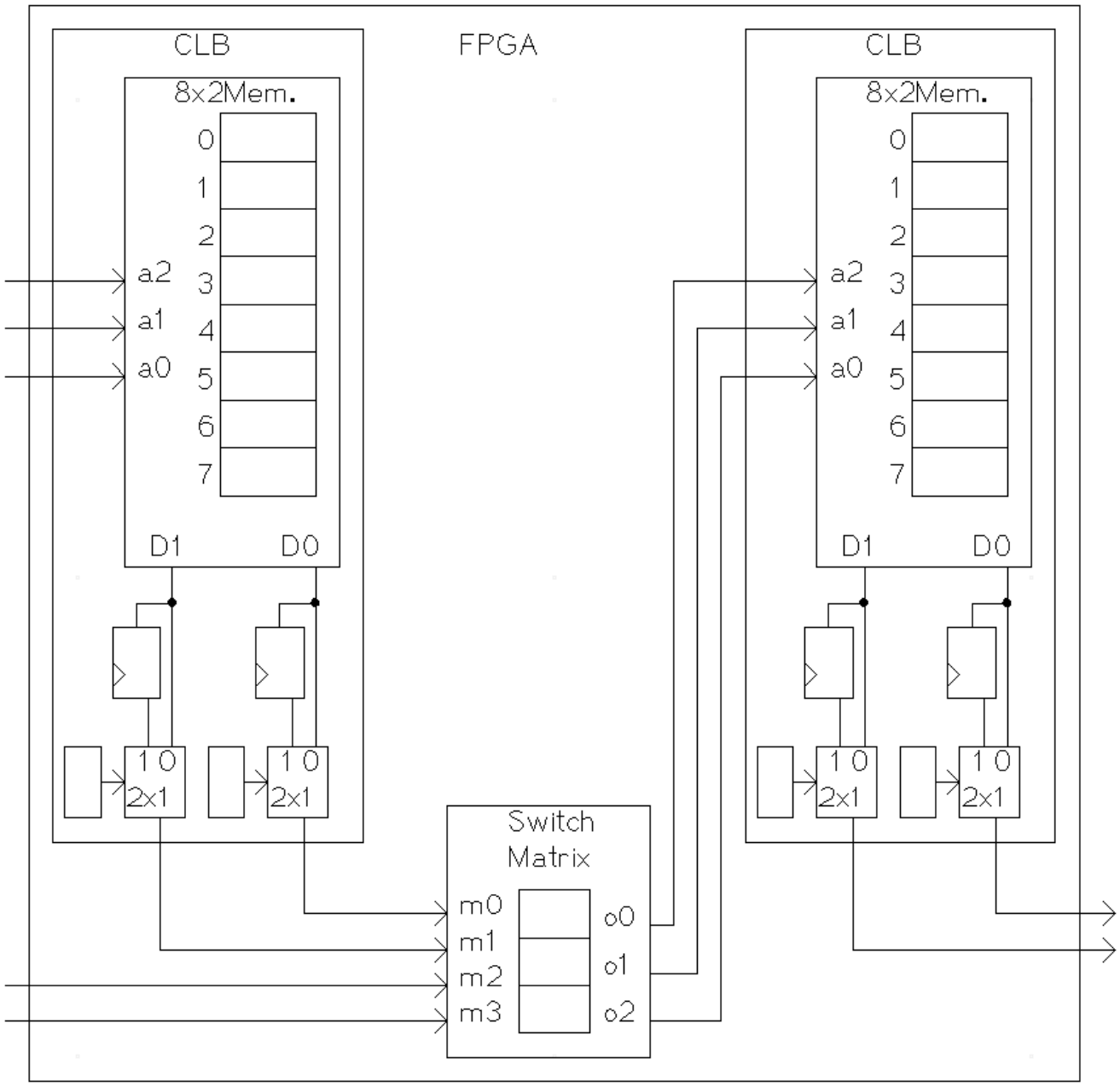
// examples
// R[0] is used for A.
// D[0] is used for B.
// R[15] is -1.

Part B. Write your code. (hint: Use comments in your code to help the grader understand what you are doing).

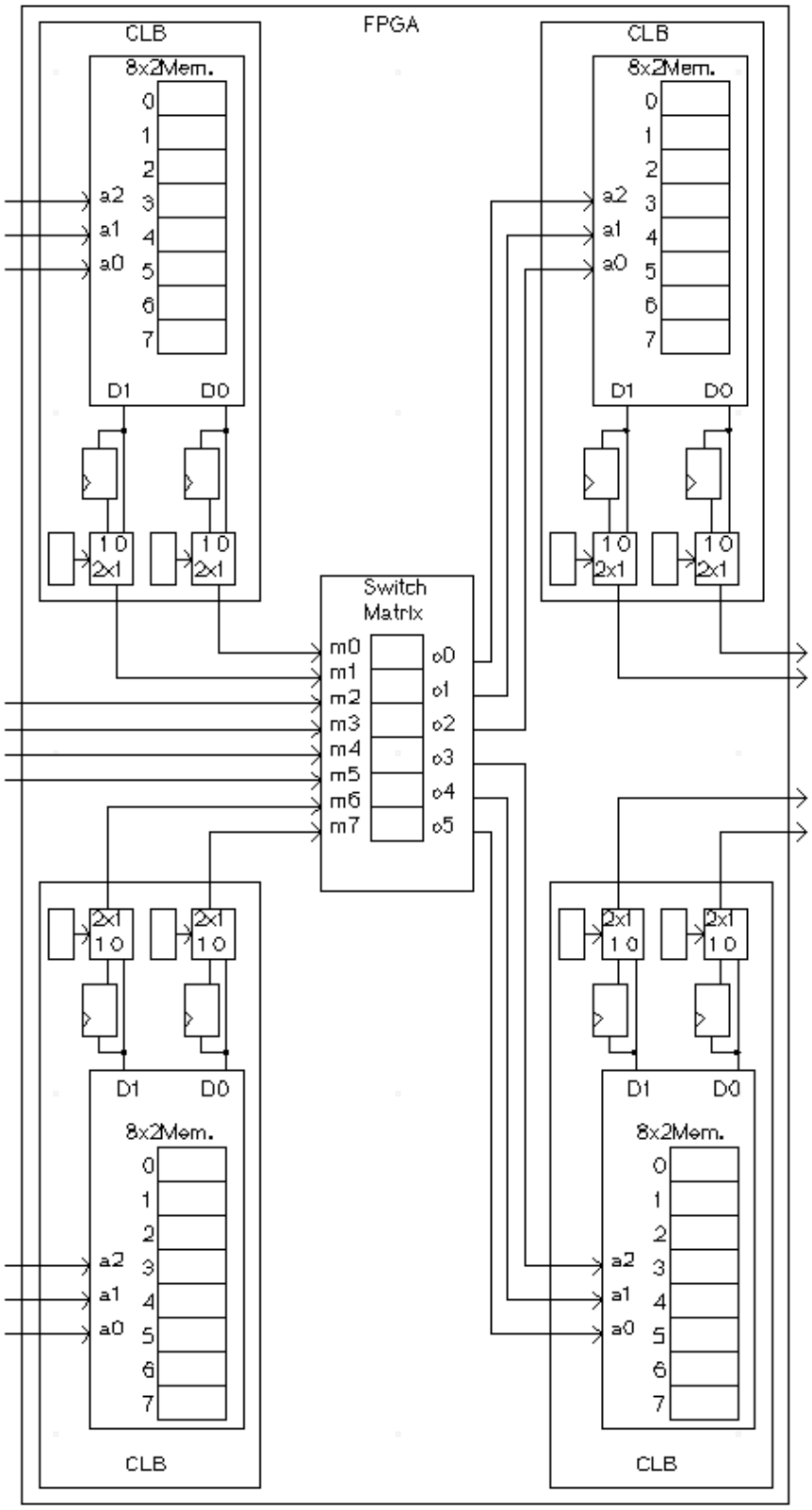
Instruction mnemonic	Description
Mov Ra, d	RF[a] = D[d]
Mov d, Ra	D[d] = RF[a]
Add Ra, Rb, Rc	RF[a] = RF[b] + RF[c]
Sub Ra, Rb, Rc	RF[a] = RF[b] - RF[c]
Mult Ra, Rb, Rc	RF[a] = RF[b] * RF[c]
Div Ra, Rb, Rc	RF[a] = RF[b] / RF[c]



3. Map the following into the provided CLB and Switch Matrix. Design a 2-bit gray code counter with count enable and reset. A gray code counter is a counter where only one bit may change when incrementing to the next count or rolling over.



4. Using the following FPGA with 4 CLB's, design a shift register with the maximum amount of delay possible from input signal Sin to output signal Sout (25 points).



5. Using the eight-instruction processor (a six-instruction processor with 2 additional instructions for multiply and divide) write assembly code to calculate the equation below. The register file is in an unknown state. Assume the ALU from the processor does floating point arithmetic. If you need a floating point constant, assume that it is in data memory. You must show the address location of constants, variables, and result in data memory. The instruction mnemonics and a processor diagram are at the end of the exam. (25 points).

$$X = 1.0 - N * (X * Y - X * Z) / (X - Y)^2$$

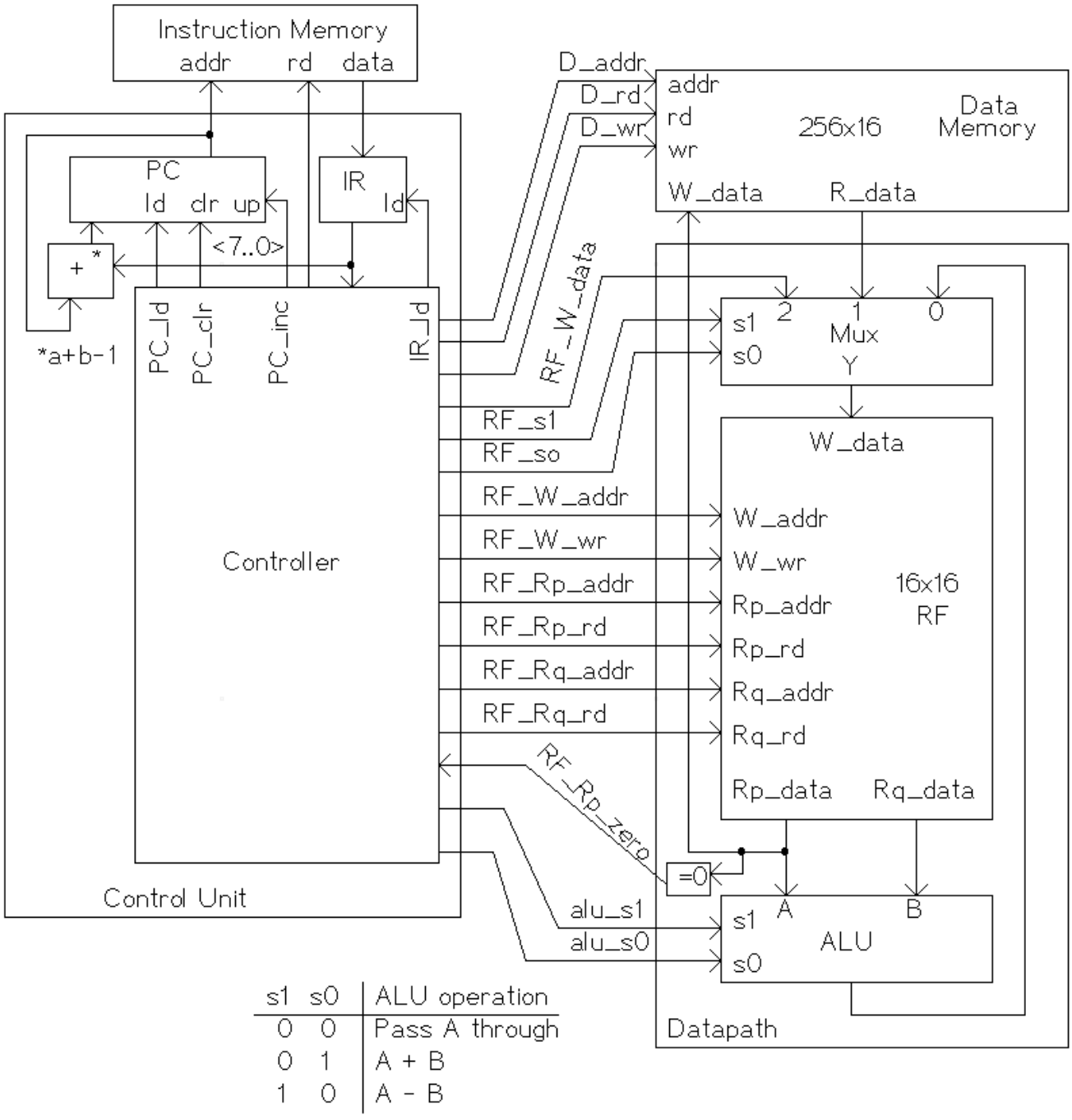
6. Design a new instruction for the six-instruction processor that performs a jump if RF[rb] is greater than RF[rc]. Draw a new FSM (you may draw the FSM changes on the last sheet), draw new diagrams or describe changes for the datapath and control unit (25 points).

Six-instruction processor

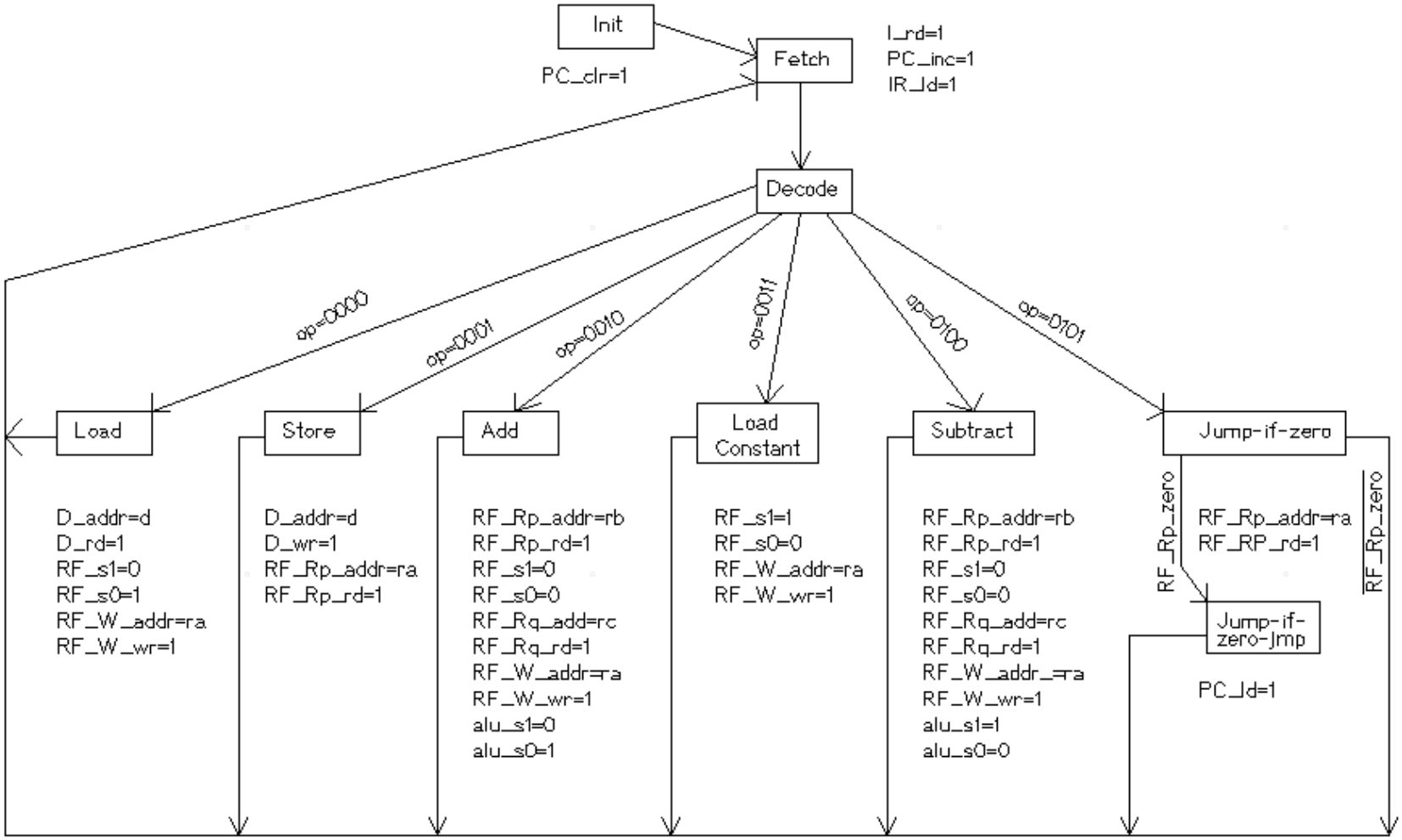
Instruction mnemonic	**Description**
Mov Ra, d	RF[a] = D[d]
Mov d, Ra	D[d] = RF[a]
Add Ra, Rb, Rc	RF[a] = RF[b] + RF[c]
Sub Ra, Rb, Rc	RF[a] = RF[b] - RF[c]
Mov Ra, #C	RF[a] = C
JMPZ Ra, offset	PC = PC + offset if RF[a] == 0

Eight-instruction processor

Instruction mnemonic	**Description**
Mov Ra, d	RF[a] = D[d]
Mov d, Ra	D[d] = RF[a]
Add Ra, Rb, Rc	RF[a] = RF[b] + RF[c]
Sub Ra, Rb, Rc	RF[a] = RF[b] - RF[c]
Mov Ra, #C	RF[a] = C
JMPZ Ra, offset	PC = PC + offset if RF[a] == 0
Mult Ra, Rb, Rc	RF[a] = RF[b] * RF[c]
Div Ra, Rb, Rc	RF[a] = RF[b] / RF[c]



s1	s0	ALU operation
0	0	Pass A through
0	1	A + B
1	0	A - B


Init
PC_clr=1
Fetch
I_rd=1
PC_inc=1
IR_ld=1
Decode
op=0000
op=0001
op=0010
op=0011
op=0100
op=0101
Load
Store
Add
Load Constant
Subtract
Jump-if-zero
D_addr=d
D_rd=1
RF_s1=0
RF_s0=1
RF_W_addr=ra
RF_W_wr=1
D_addr=d
D_wr=1
RF_Rp_addr=ra
RF_Rp_rd=1
RF_Rp_addr=rb
RF_Rp_rd=1
RF_s1=0
RF_s0=0
RF_Rq_add=rc
RF_Rq_rd=1
RF_W_addr=ra
RF_W_wr=1
alu_s1=0
alu_s0=1
RF_s1=1
RF_s0=0
RF_W_addr=ra
RF_W_wr=1
RF_Rp_addr=rb
RF_Rp_rd=1
RF_s1=0
RF_s0=0
RF_Rq_add=rc
RF_Rq_rd=1
RF_W_addr=ra
RF_W_wr=1
alu_s1=1
alu_s0=0
RF_Rp_zero
RF_Rp_addr=ra
RF_RP_rd=1
RF_Rp_zero
Jump-if-zero-jmp
PC_ld=1