计算机组成原理

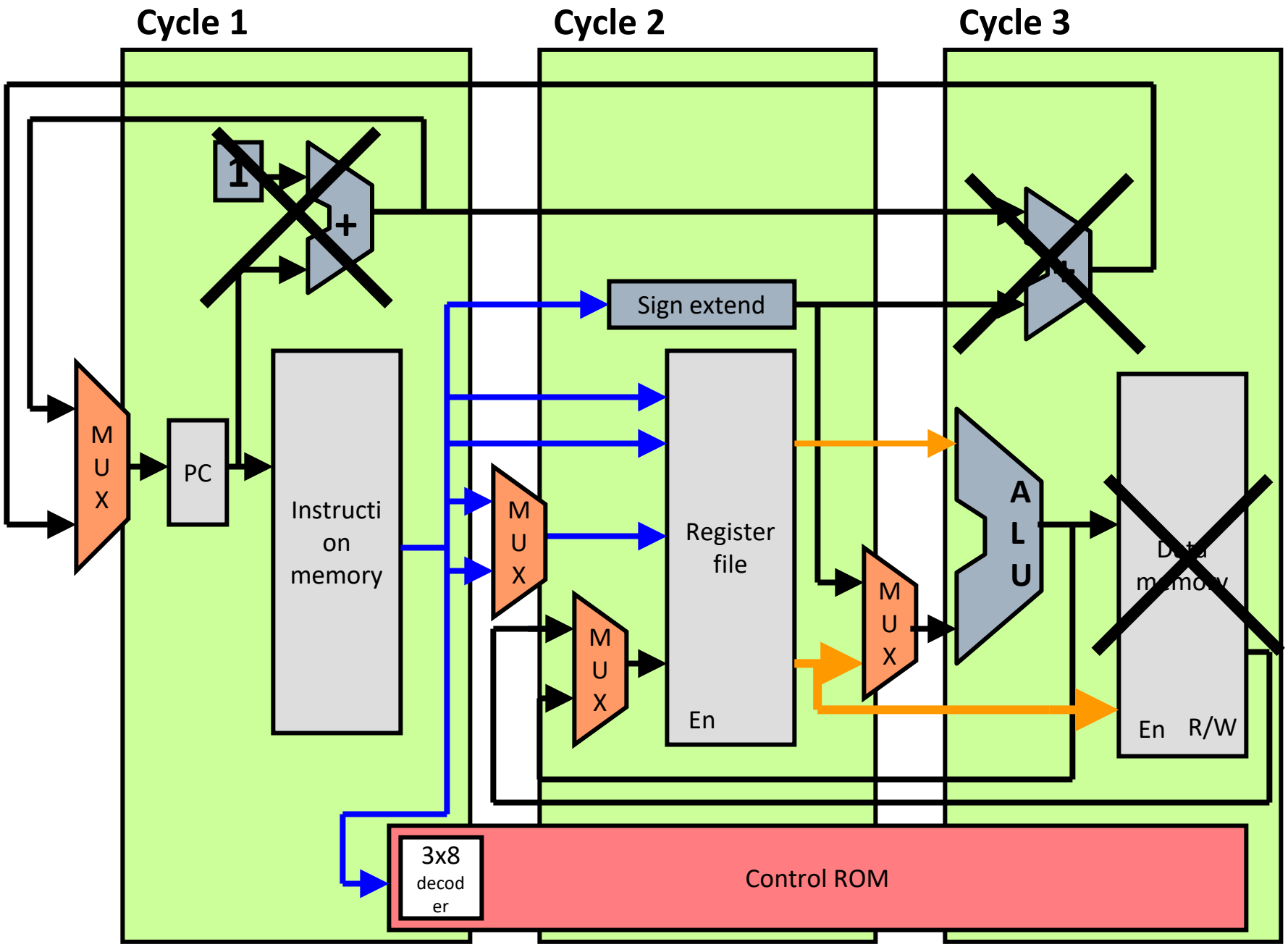
多周期处理器设计

熊波涛
xiongbotao@dlut.edu.cn

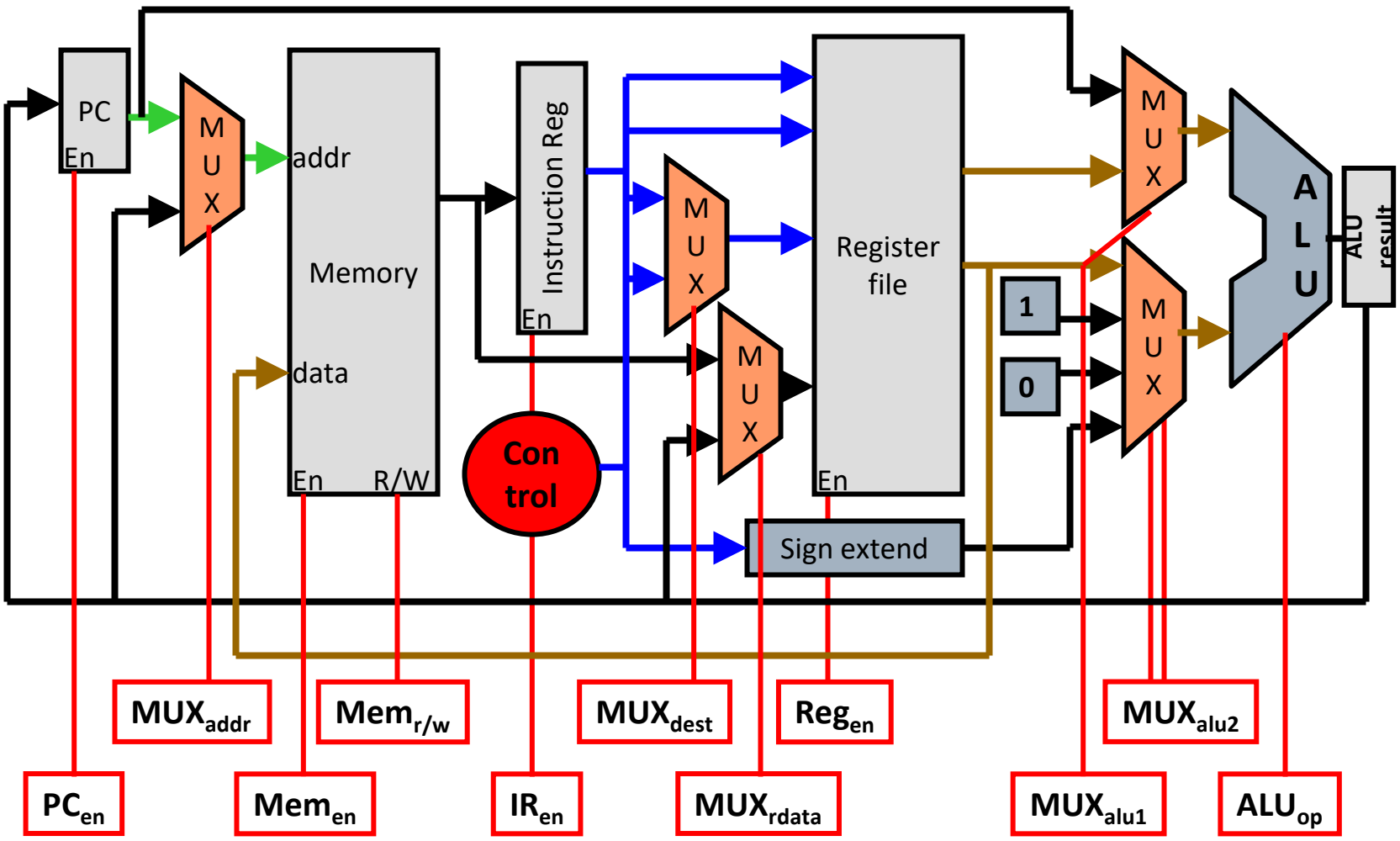
Multiple-Cycle Execution

- Each instruction takes multiple cycles to execute
  - Cycle time is reduced
  - Slower instructions take more cycles
  - Faster instruction take fewer cycles
    - We can start next instruction earlier, rather than just waiting
  - Can reuse datapath elements each cycle

- What is needed to make this work?
  - Since you are re-using elements for different purposes, you need more and/or wider MUXes.
  - You may need extra registers if you need to remember an output for 1 or more cycles.
  - Control is more complicated since you need to send new signals on each cycle.


Cycle 1
Cycle 2
Cycle 3
1
+
Sign extend
M
U
X
PC
Instructi
on
memory
Register
file
En
A
L
U
En
R/W
3x8
decod
er
Control ROM

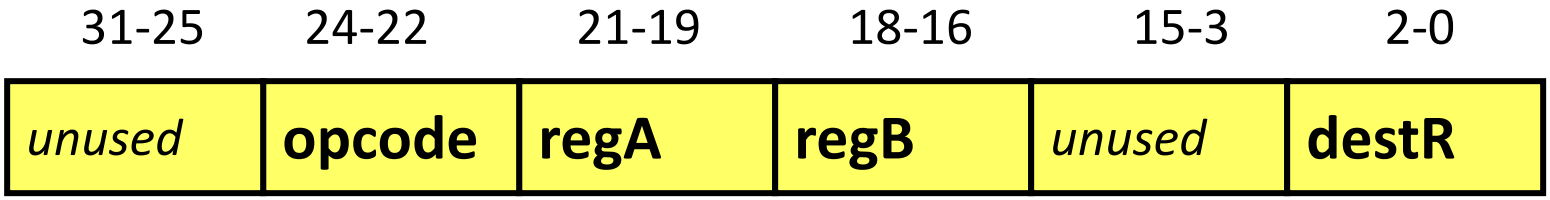
Multicycle LC2K Datapath



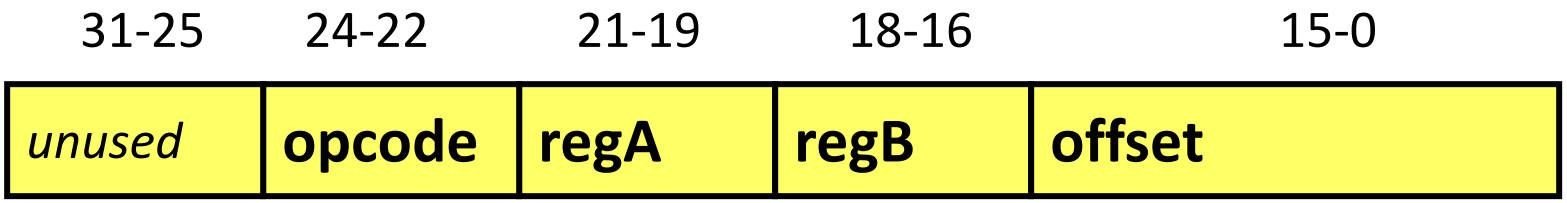
Recap: LC2K Instruction Formats

- Tells you which bit positions mean what

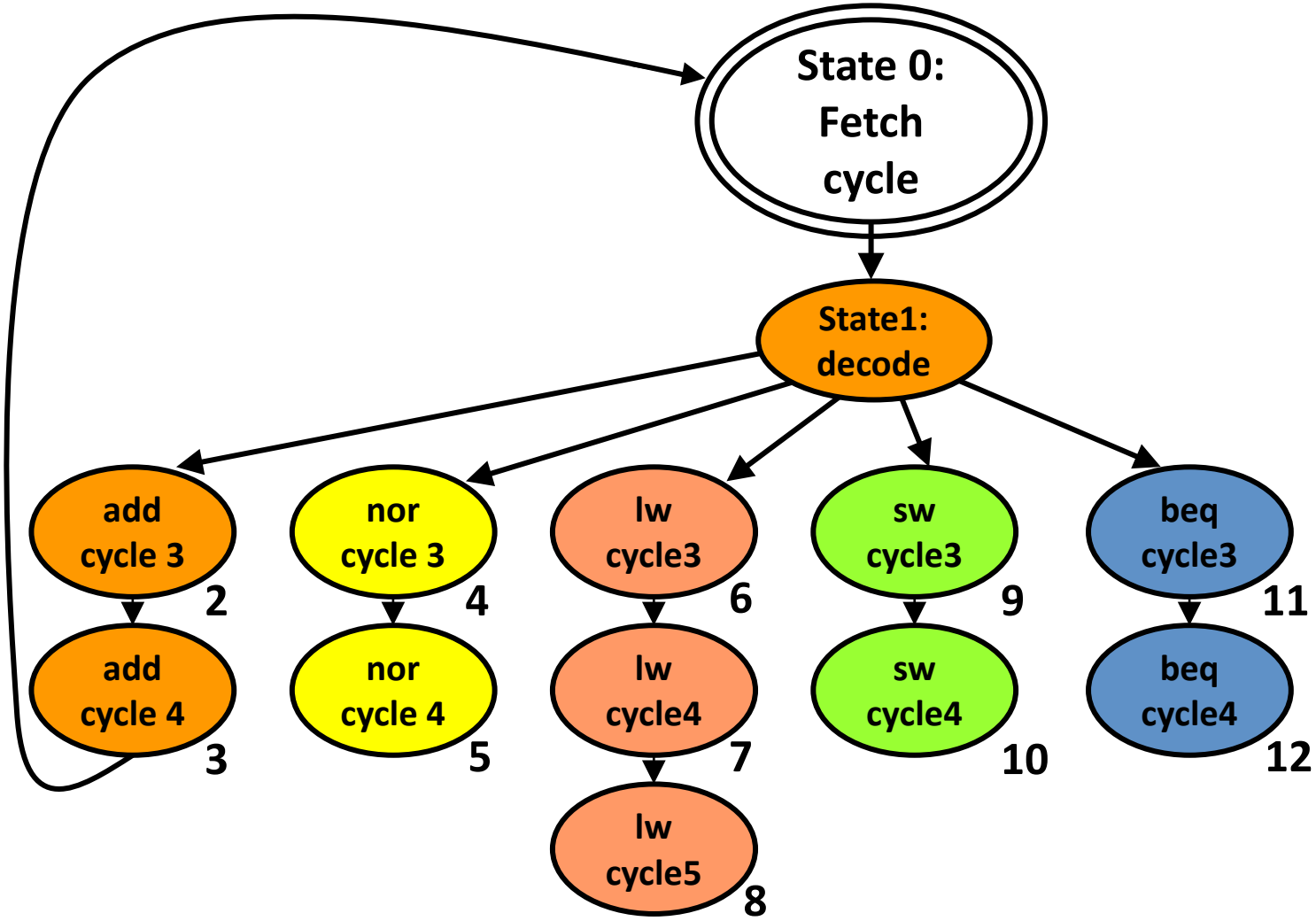
- R type instructions (add ‘000’, nor ‘001’)



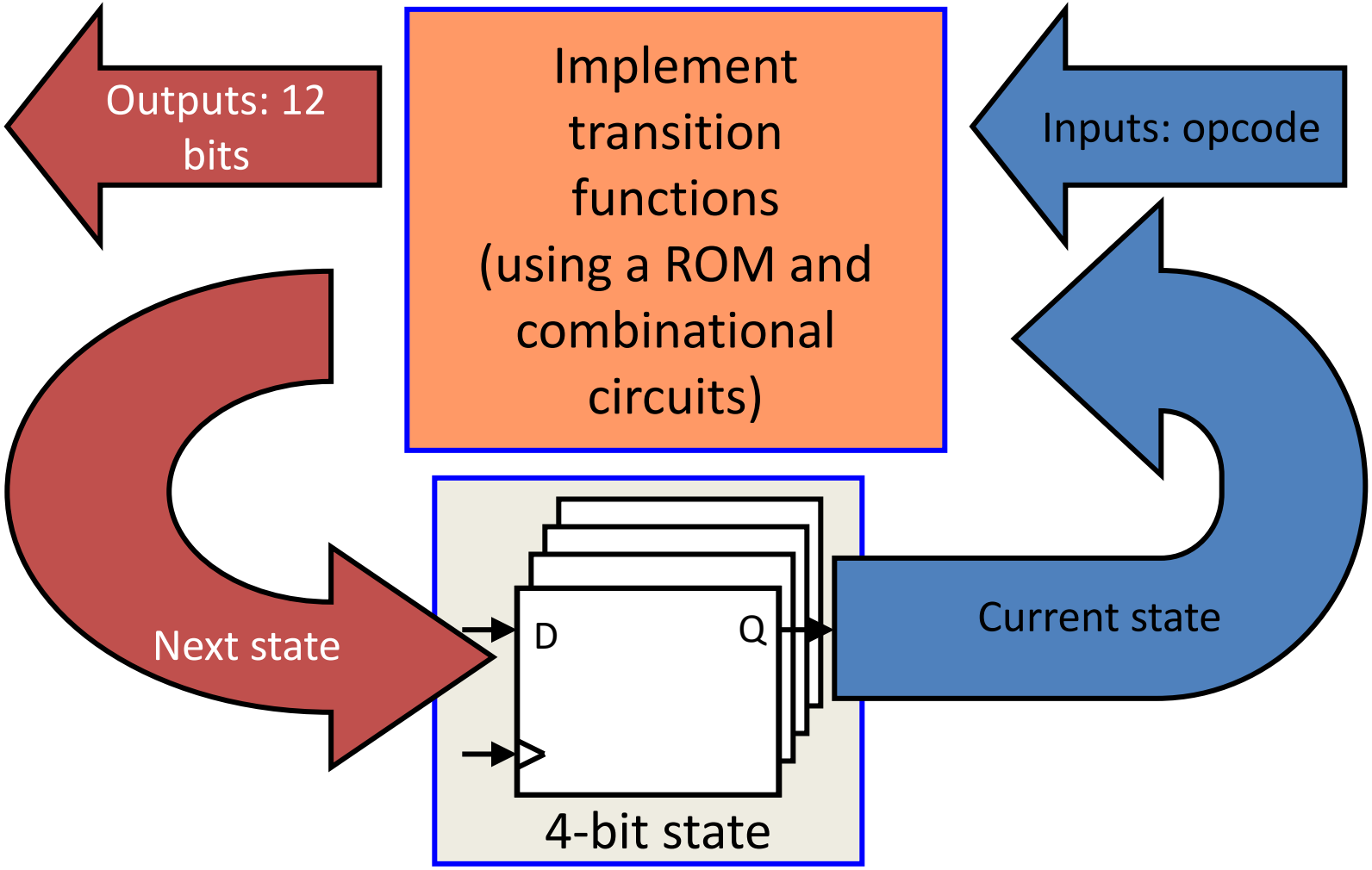
- I type instructions (lw ‘010’, sw ‘011’, beq ‘100’)



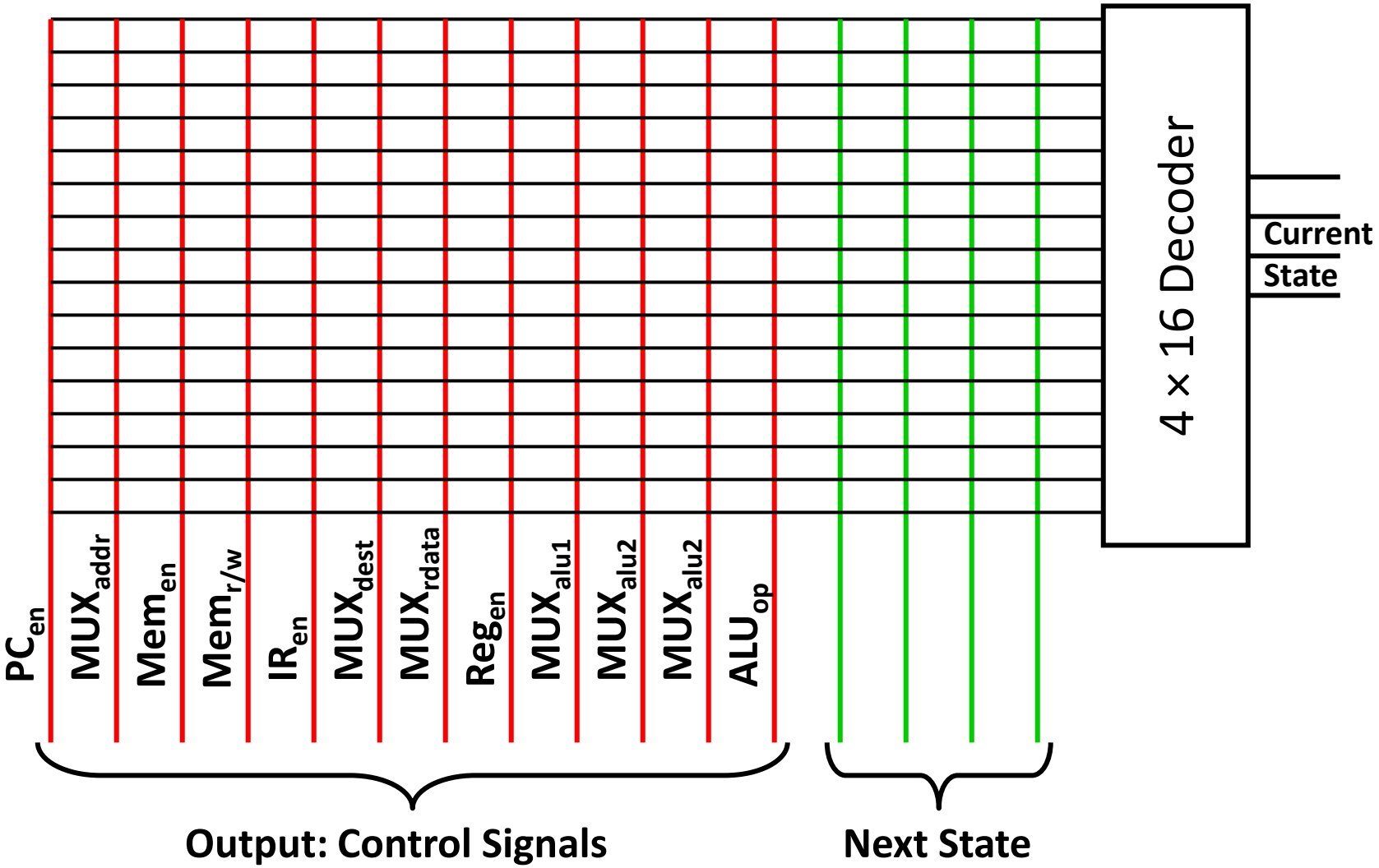
State machine for multi-cycle control signals (transition functions)



Implementing FSM

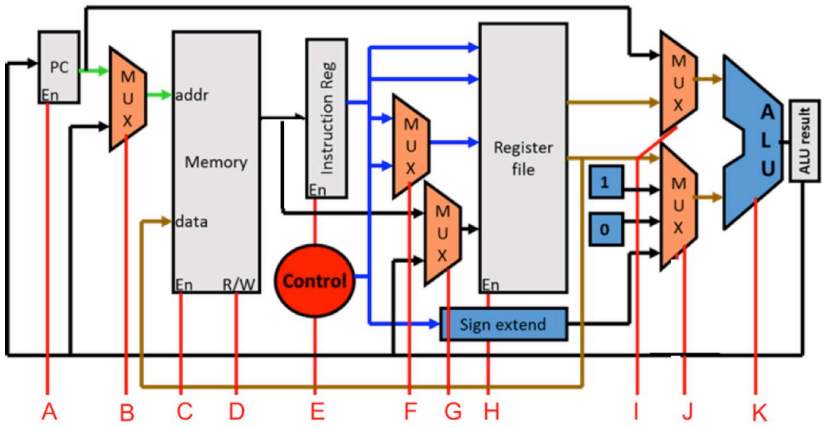


Building the Control Rom



First Cycle (State 0) Fetch Instr

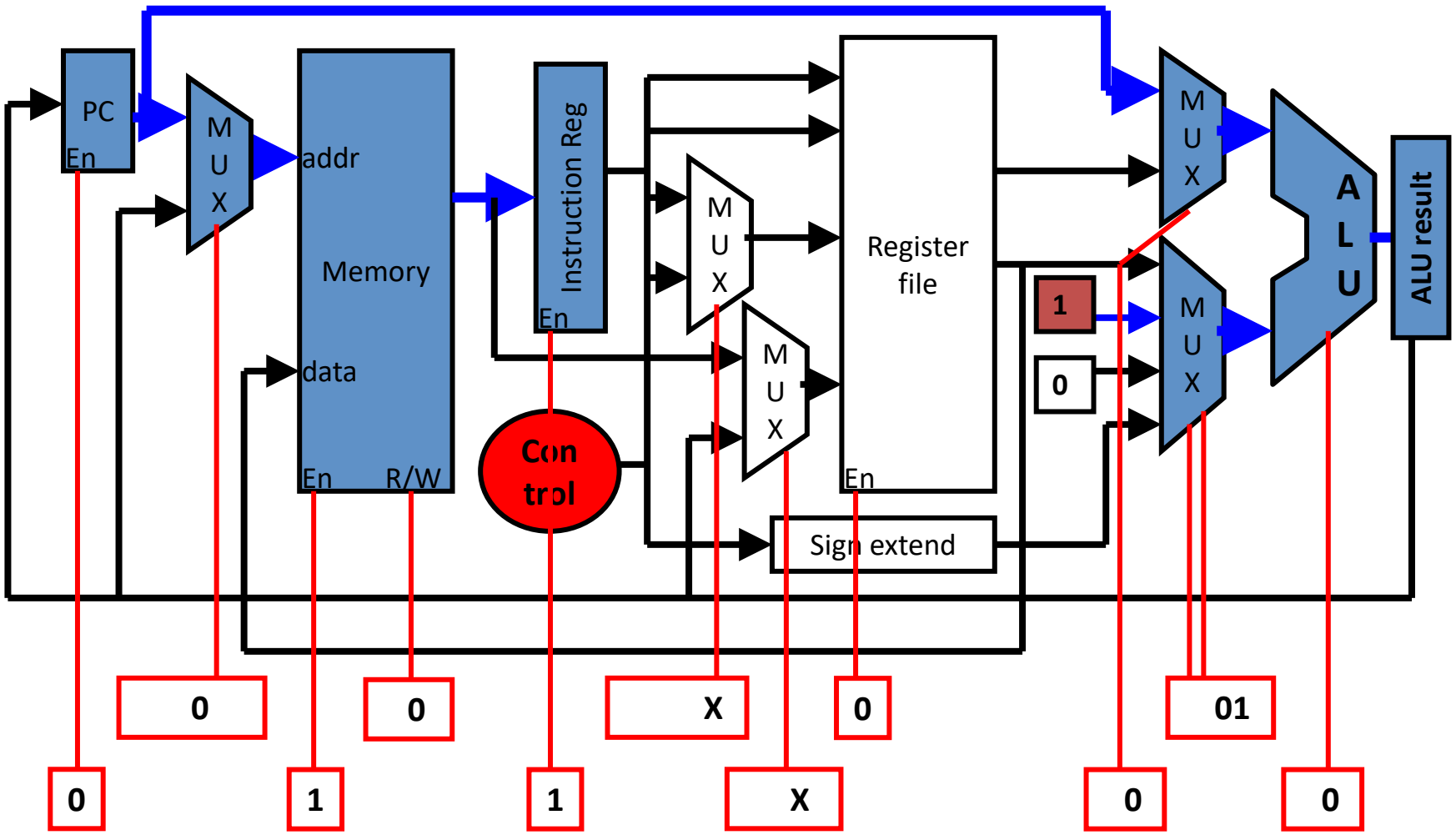
- What operations need to be done in the first cycle of executing any instruction?
  - Read memory[PC] and store into instruction register.
    - Must select PC in memory address MUX (MUX_{addr}= 0)
    - Enable memory operation (Mem_{en}= 1)
    - R/W should be (read) ($Mem_{r/w}$= 0)
    - Enable Instruction Register write (IR_{en}= 1)
  - Calculate PC + 1
    - Send PC to ALU (MUX_{alu1} = 0)
    - Send 1 to ALU (MUX_{alu2} = 01)
    - Select ALU add operation (ALU_{op} = 0)
  - PC_{en} = 0; Reg_{en} = 0; MUX_{dest} and MUX_{rdata}= X



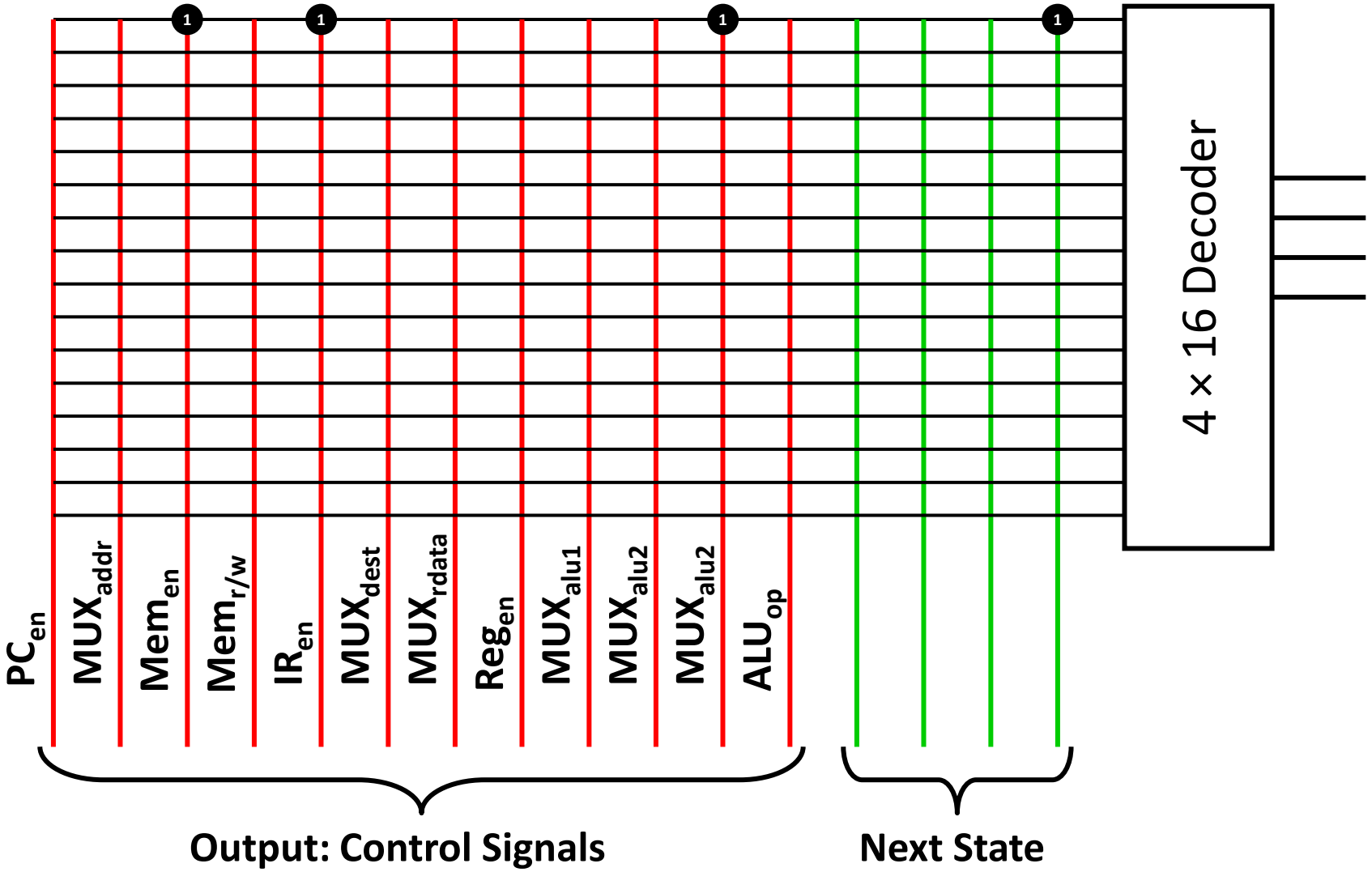
- Next State: Decode Instruction

First Cycle (State 0) Operation

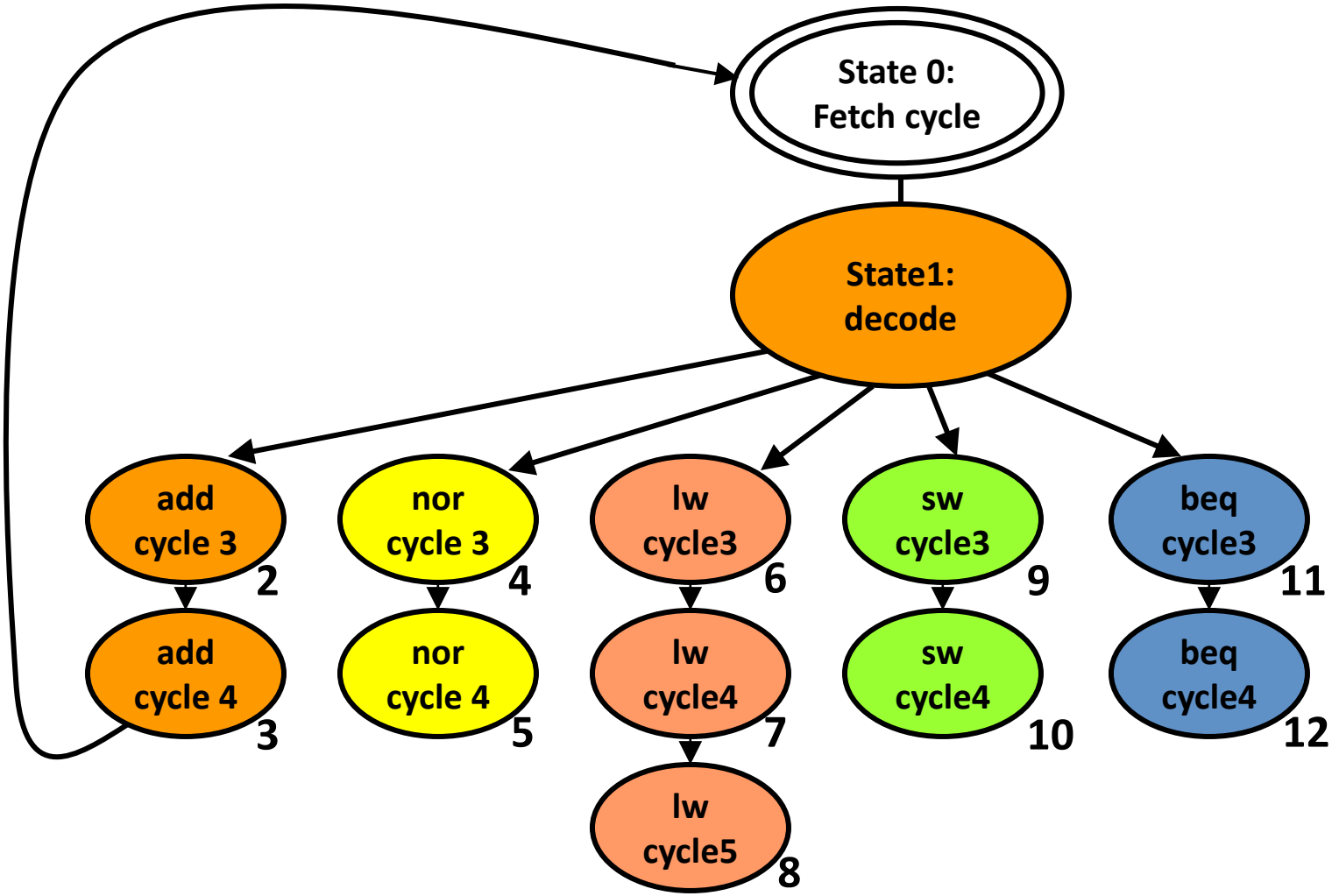
This is the same for all instructions
(since we don't know the instruction yet!)



Building the Control Rom

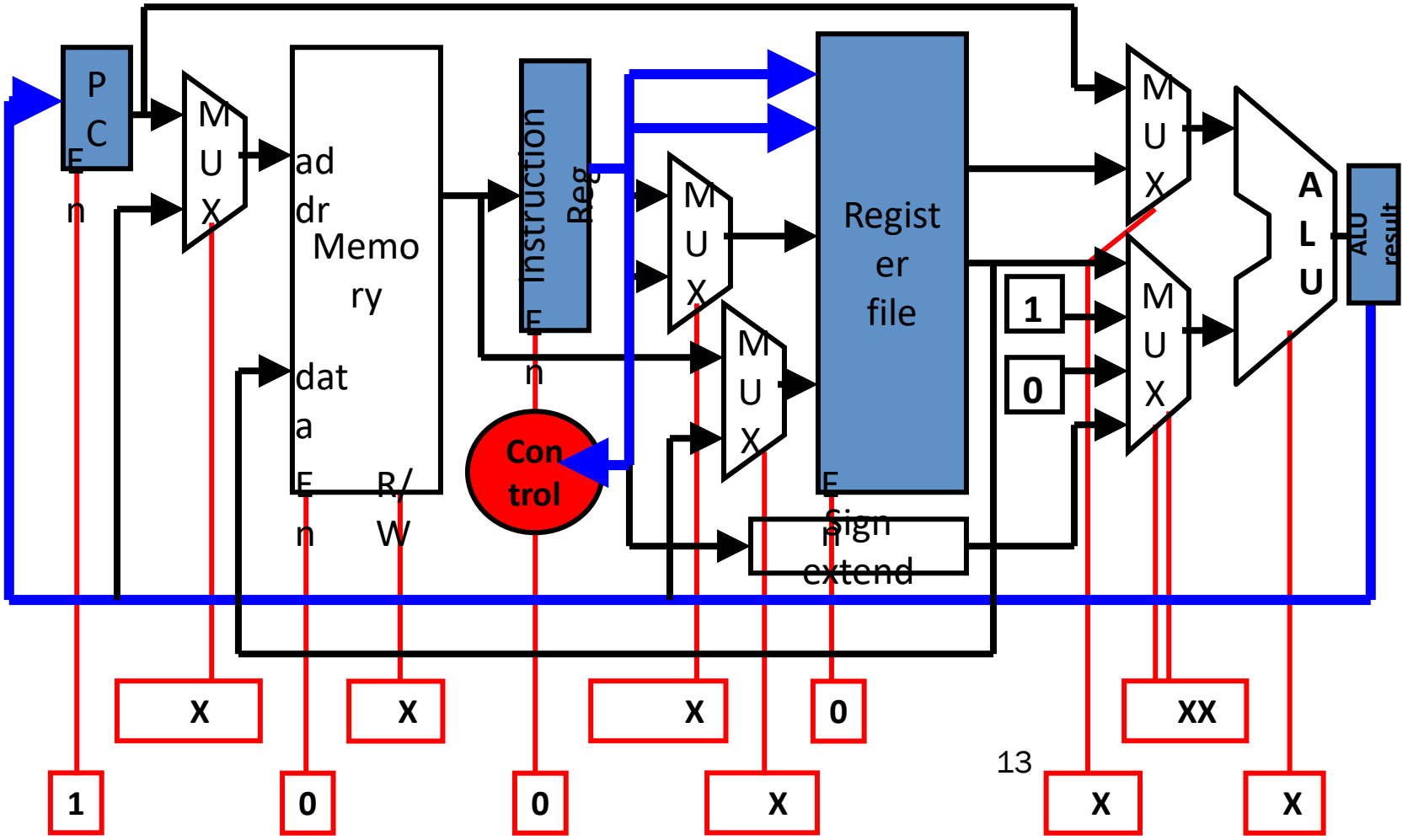


State 1: instruction decode

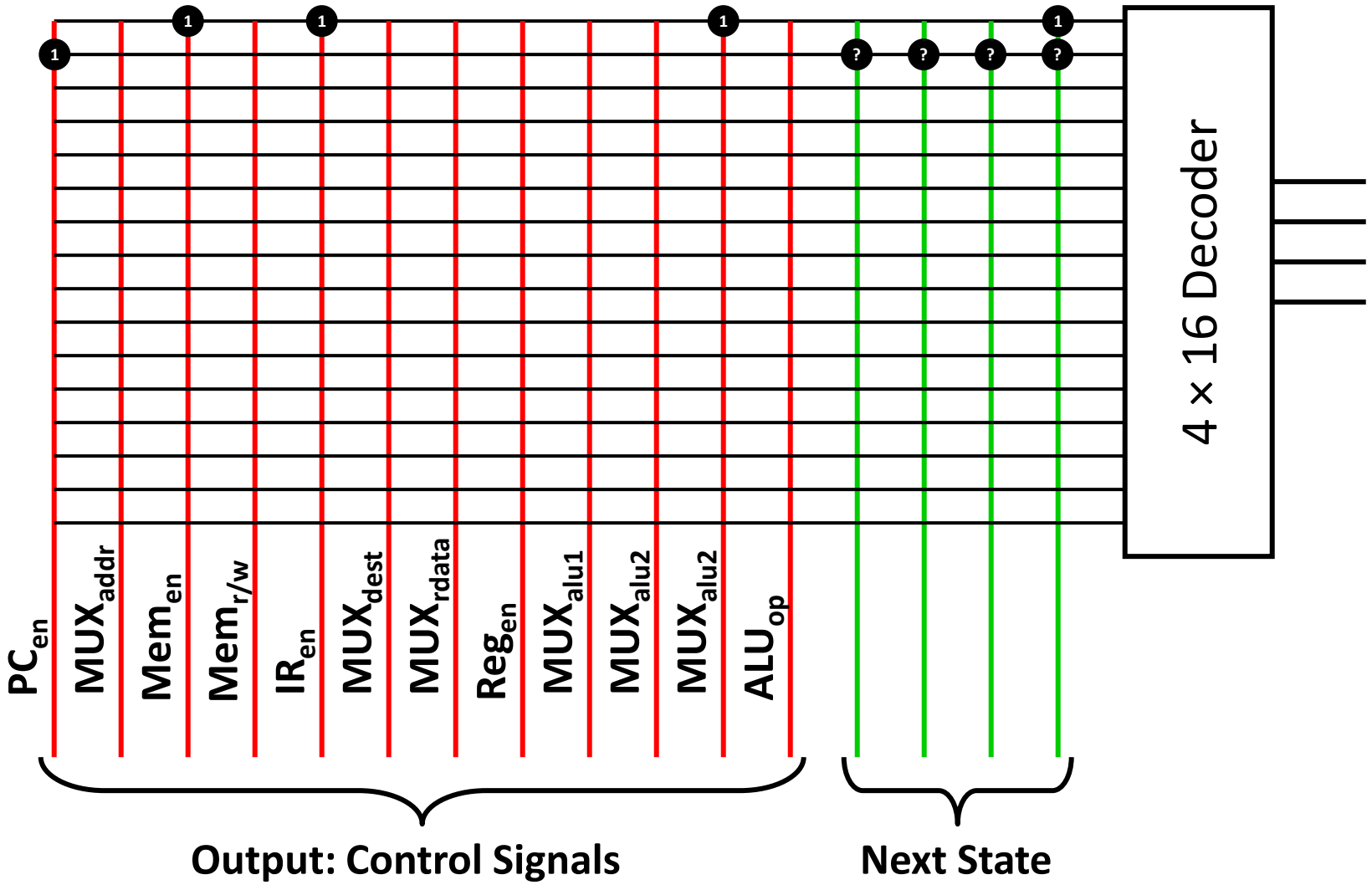


State 1: output function

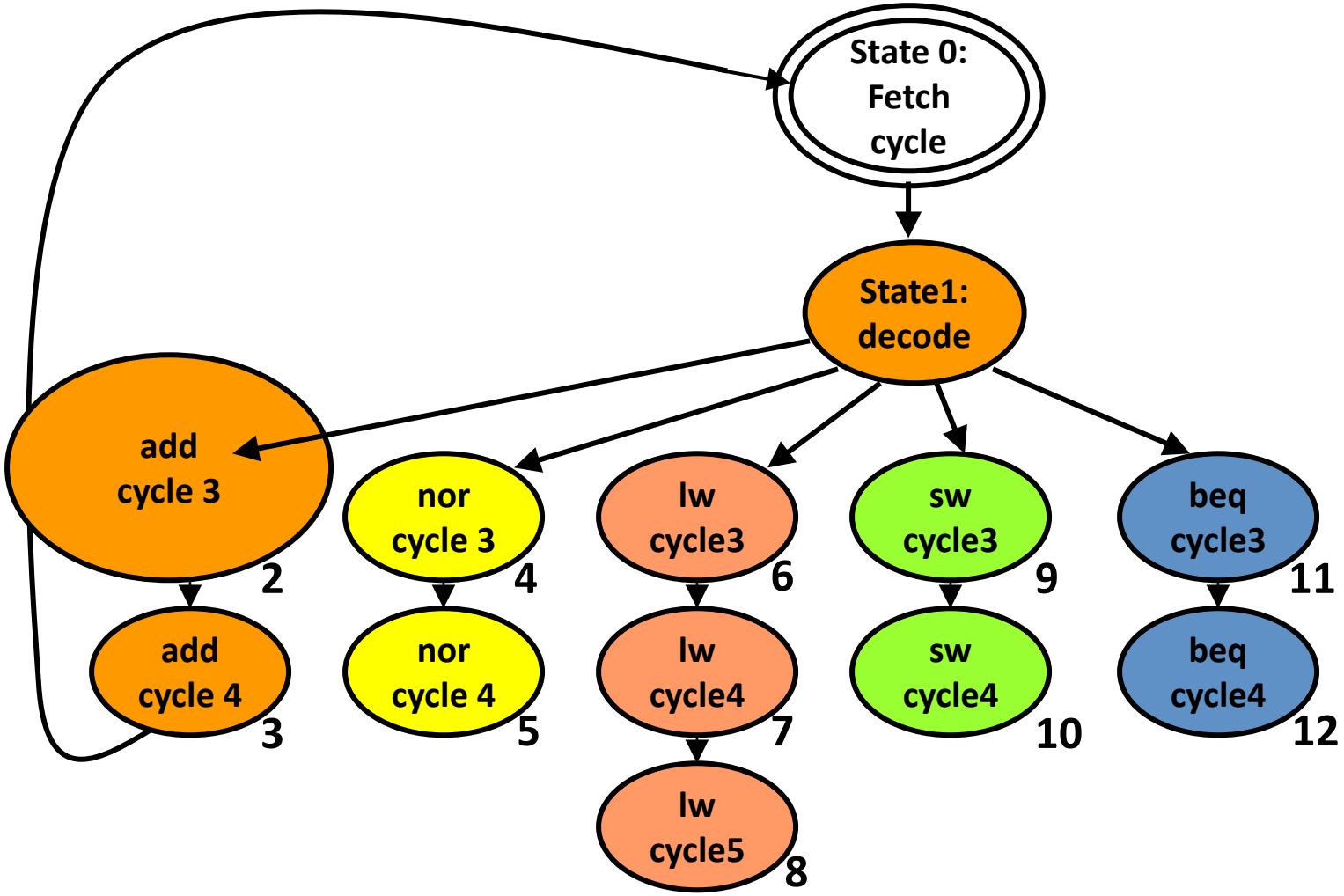
Update PC; read registers (regA and regB);
use opcode to determine next state



Building the Control Rom

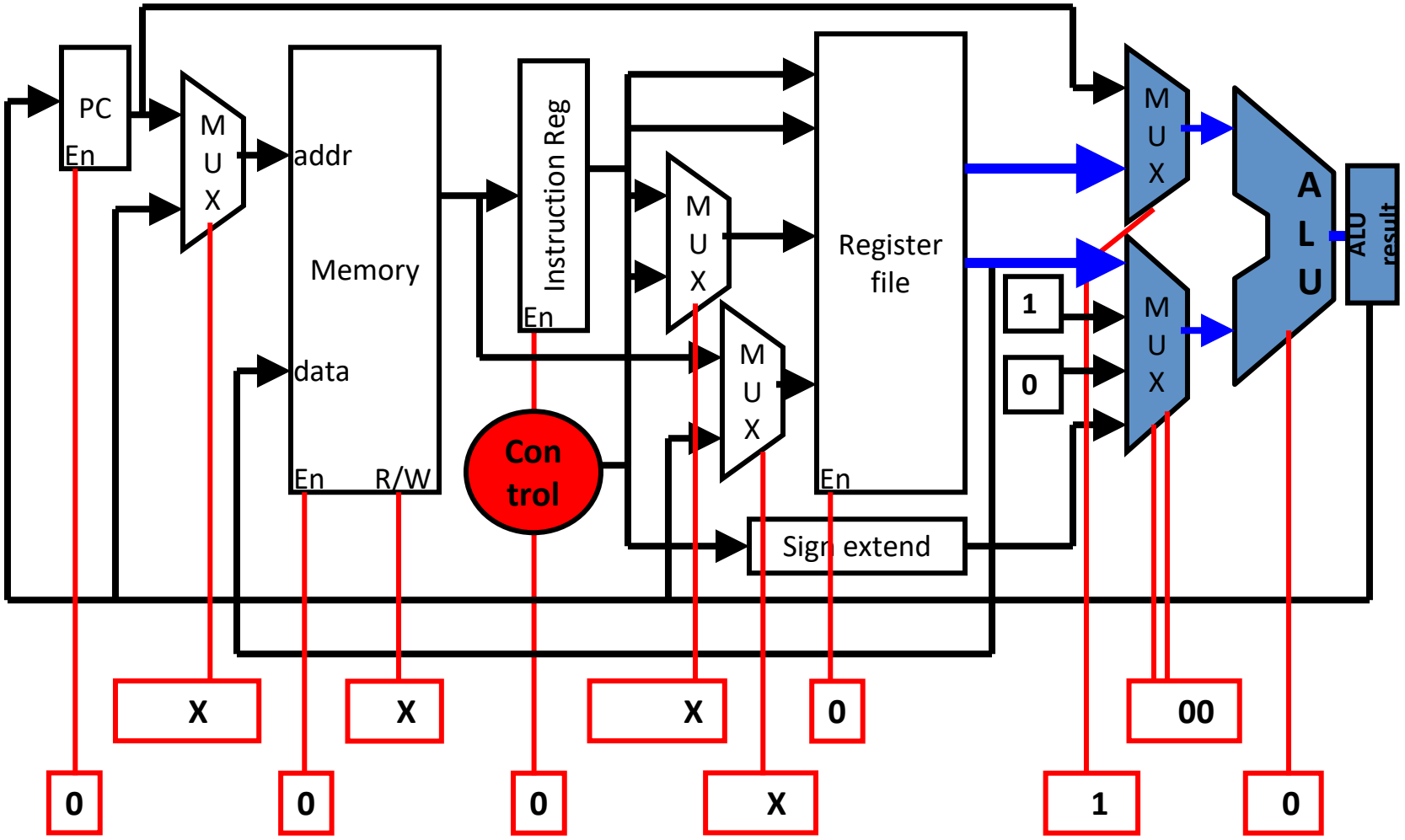


State 2: Add cycle 3

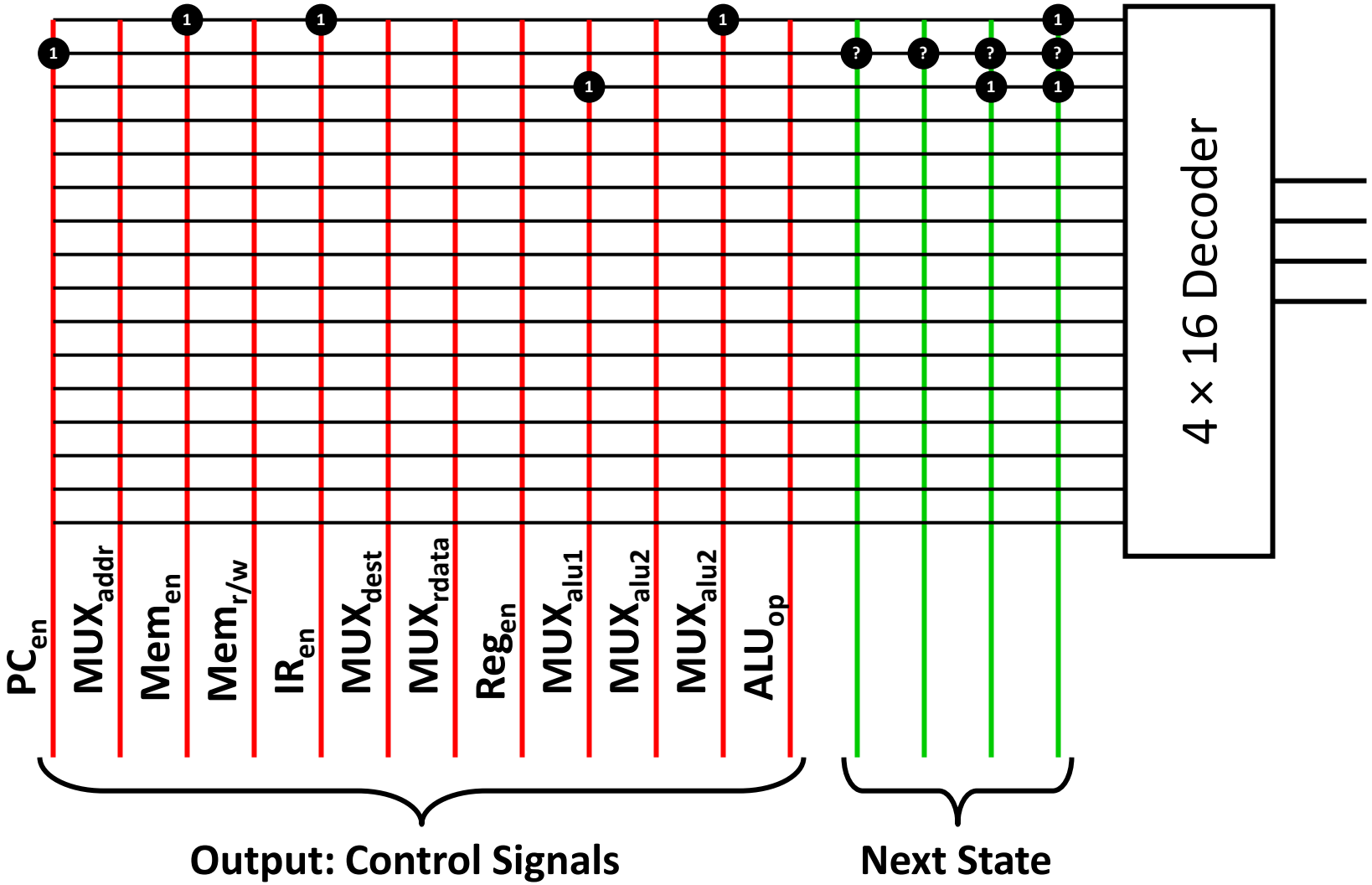


State 2: Add Cycle 3 Operation

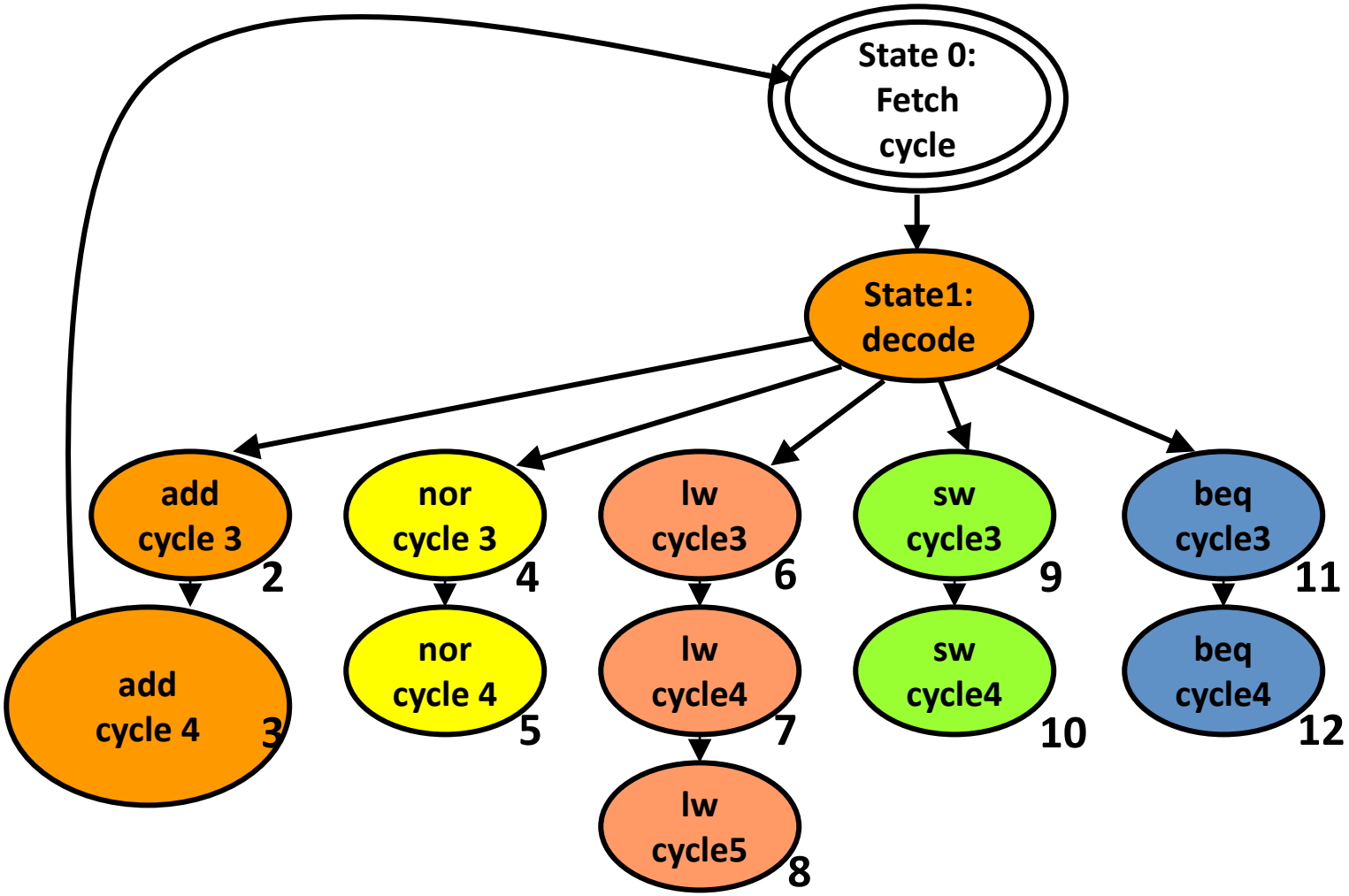
Send control signals to MUX to select values of regA and regB and control signal to ALU to add



Building the Control Rom

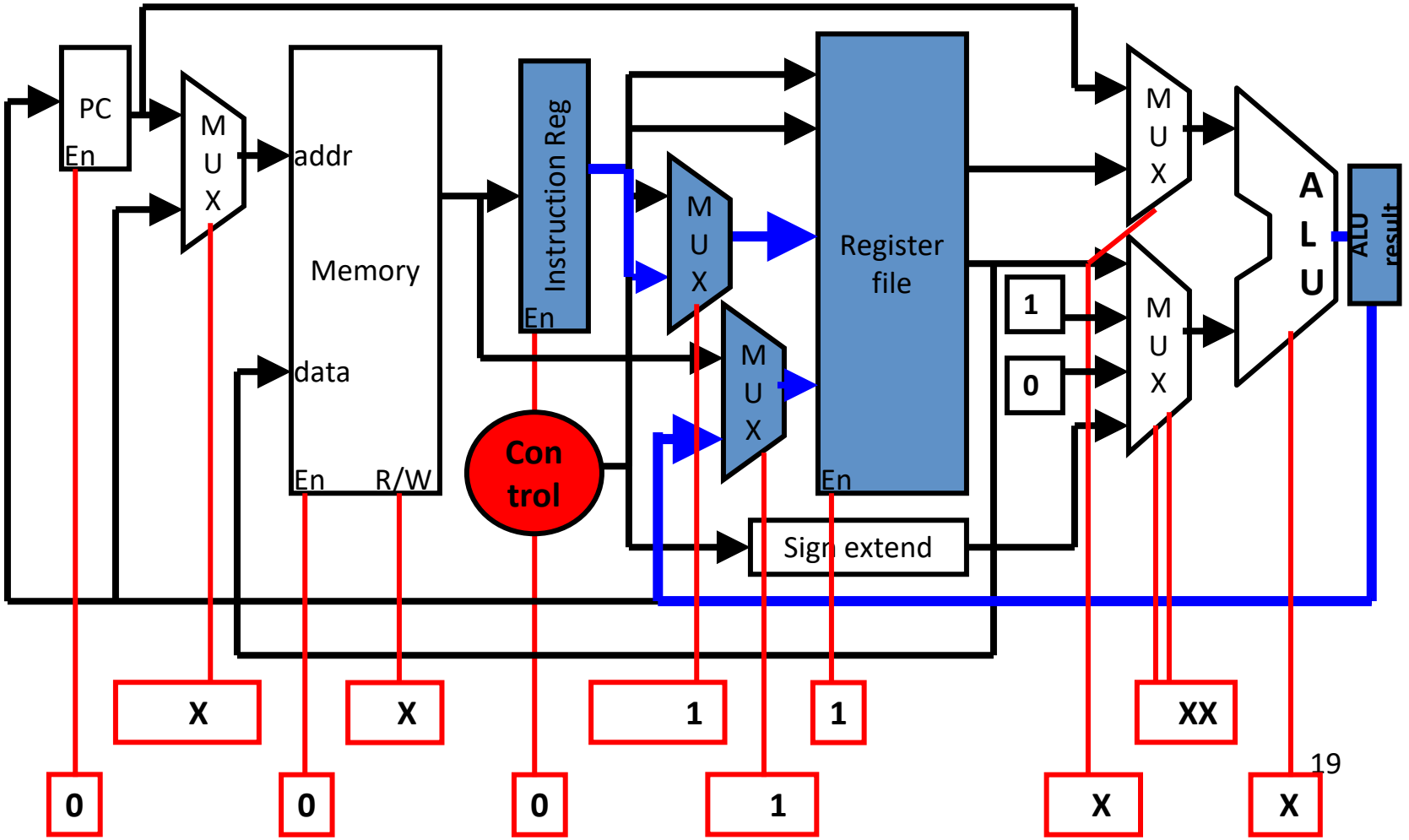


State 3: Add cycle 4

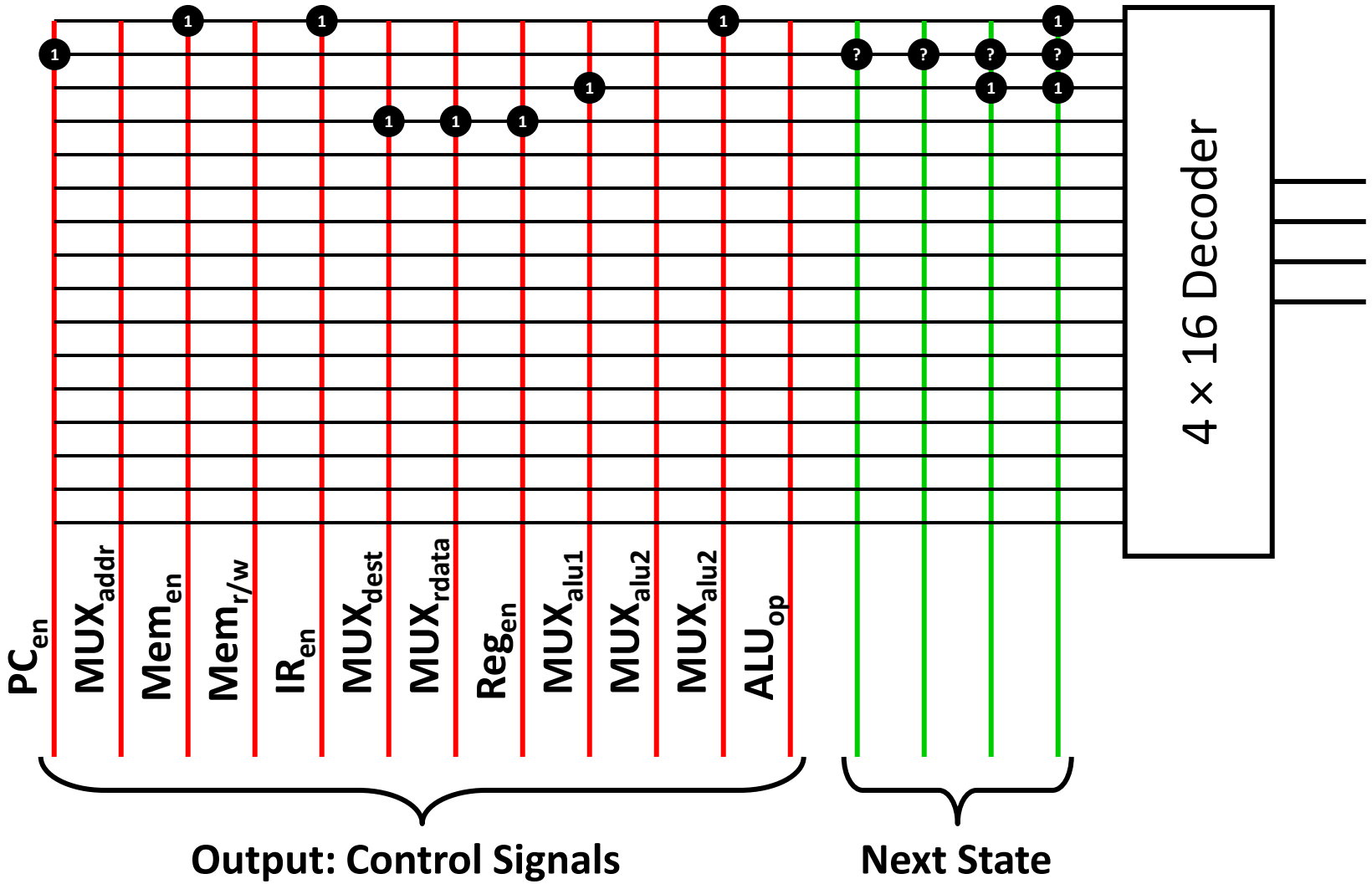


Add Cycle 4 (State 3) Operation

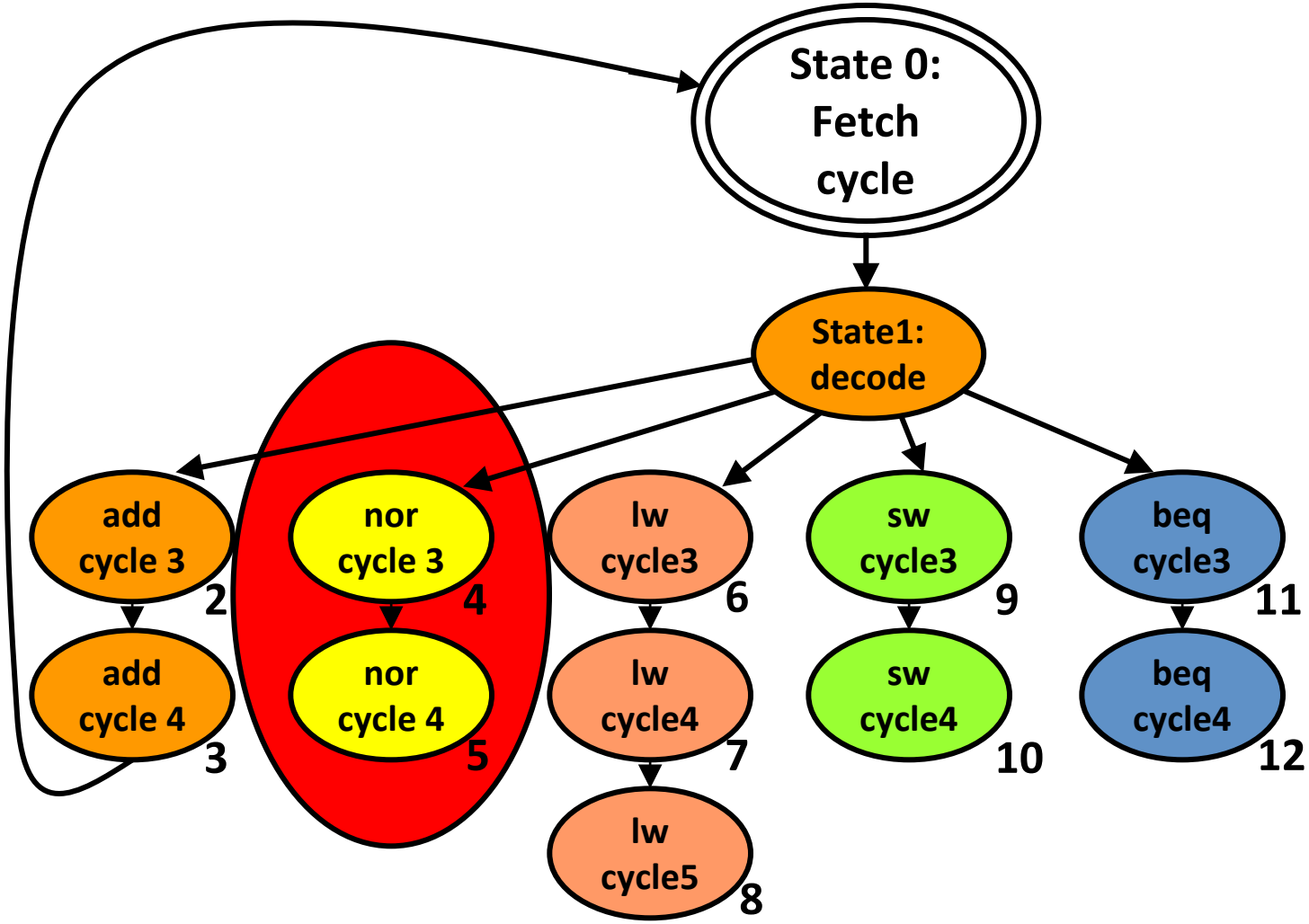
Send control signal to address MUX to select dest and to data MUX to select ALU output, then send write enable to register file.



Building the Control Rom

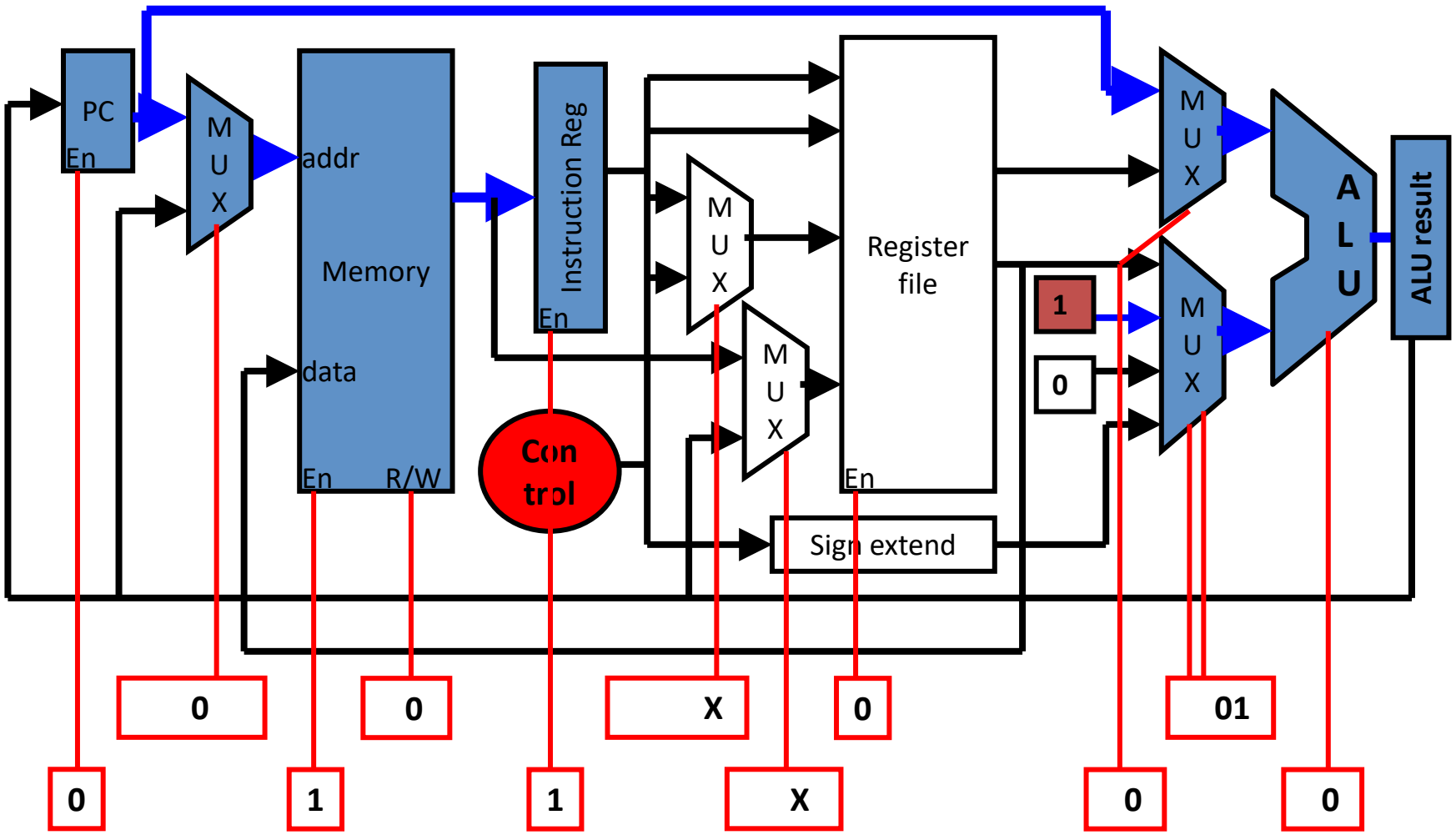


Return to State 0: Fetch cycle to execute the next instruction



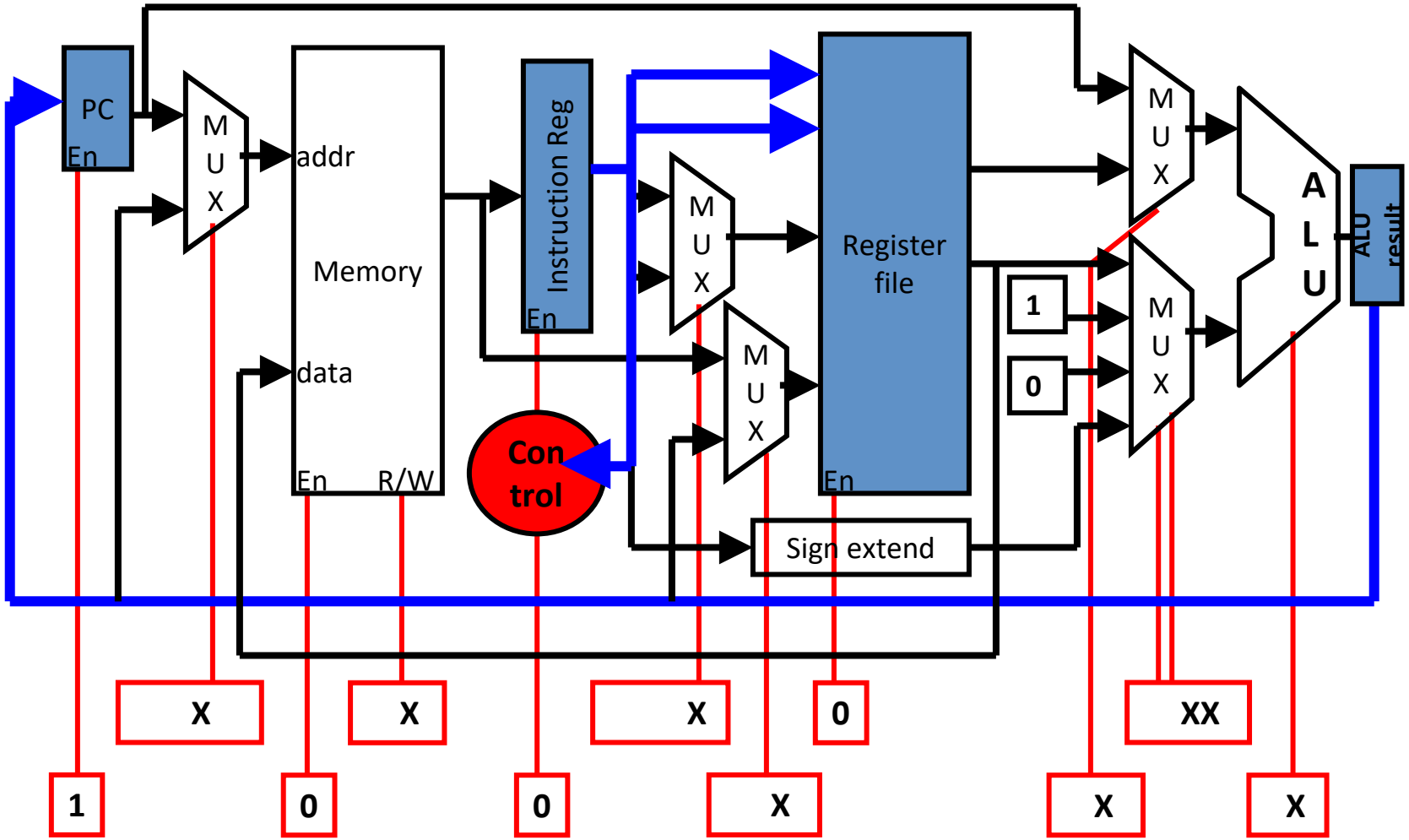
First Cycle (State 0) Operation

This is the same for all instructions
(since we don't know the instruction yet!)



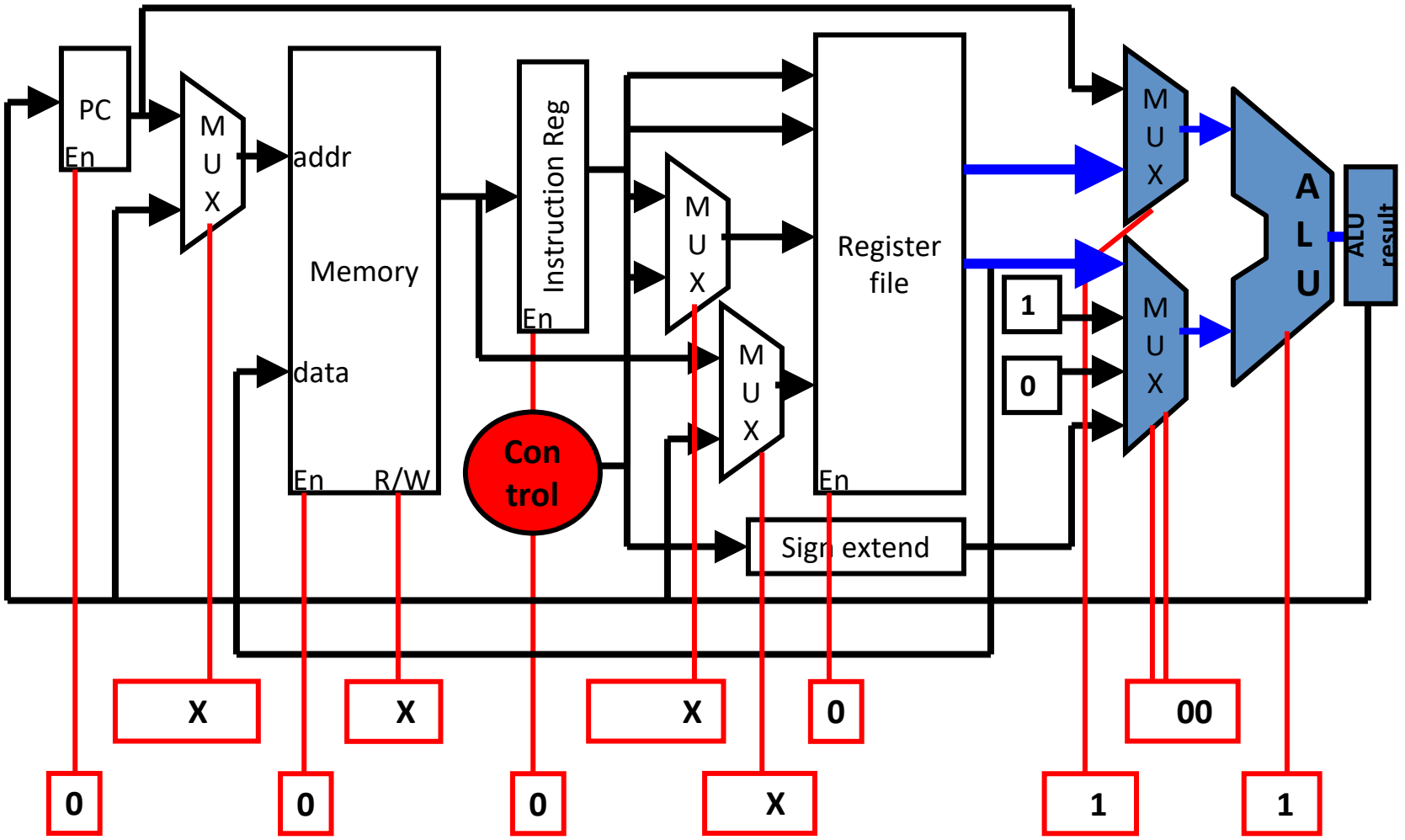
State 1: output function

Update PC; read registers (regA and regB);
use opcode to determine next state



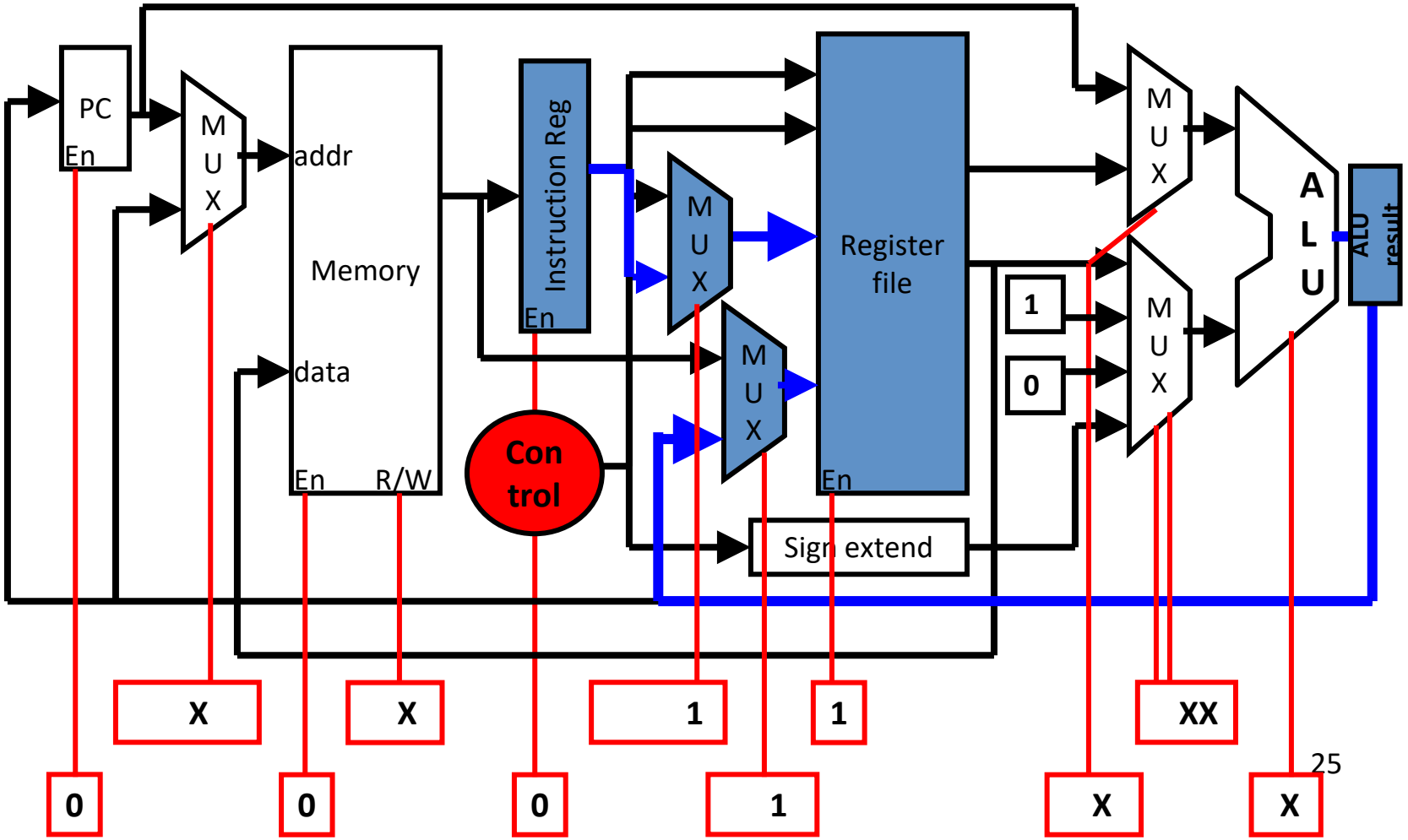
State 4: Nor Cycle 3 Operation

Send control signals to MUX to select values of regA and regB and control signal to ALU to add



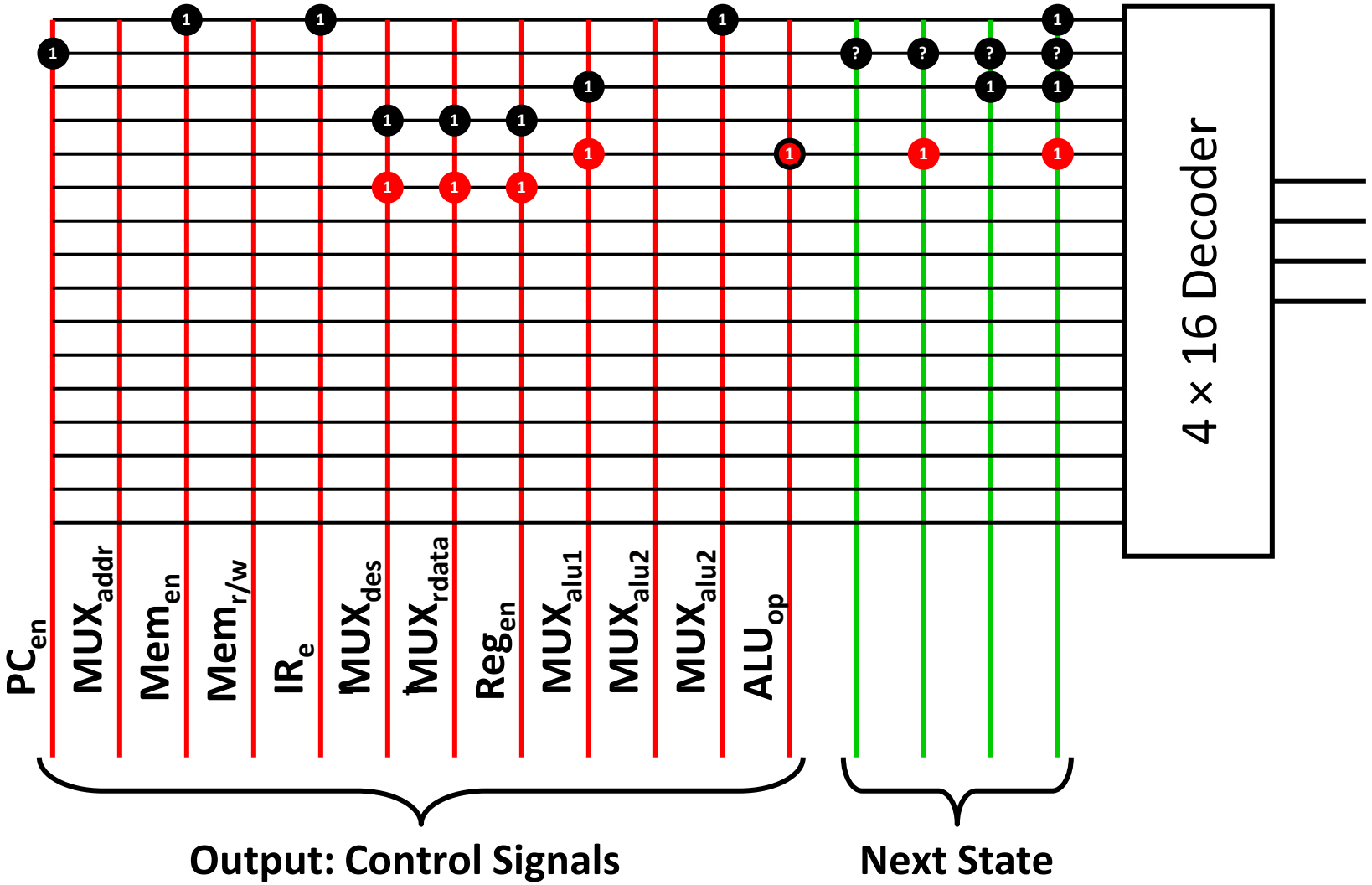
Nor Cycle 4 (State 5) Operation

Send control signal to address MUX to select dest and to data MUX to select ALU output, then send write enable to register file.

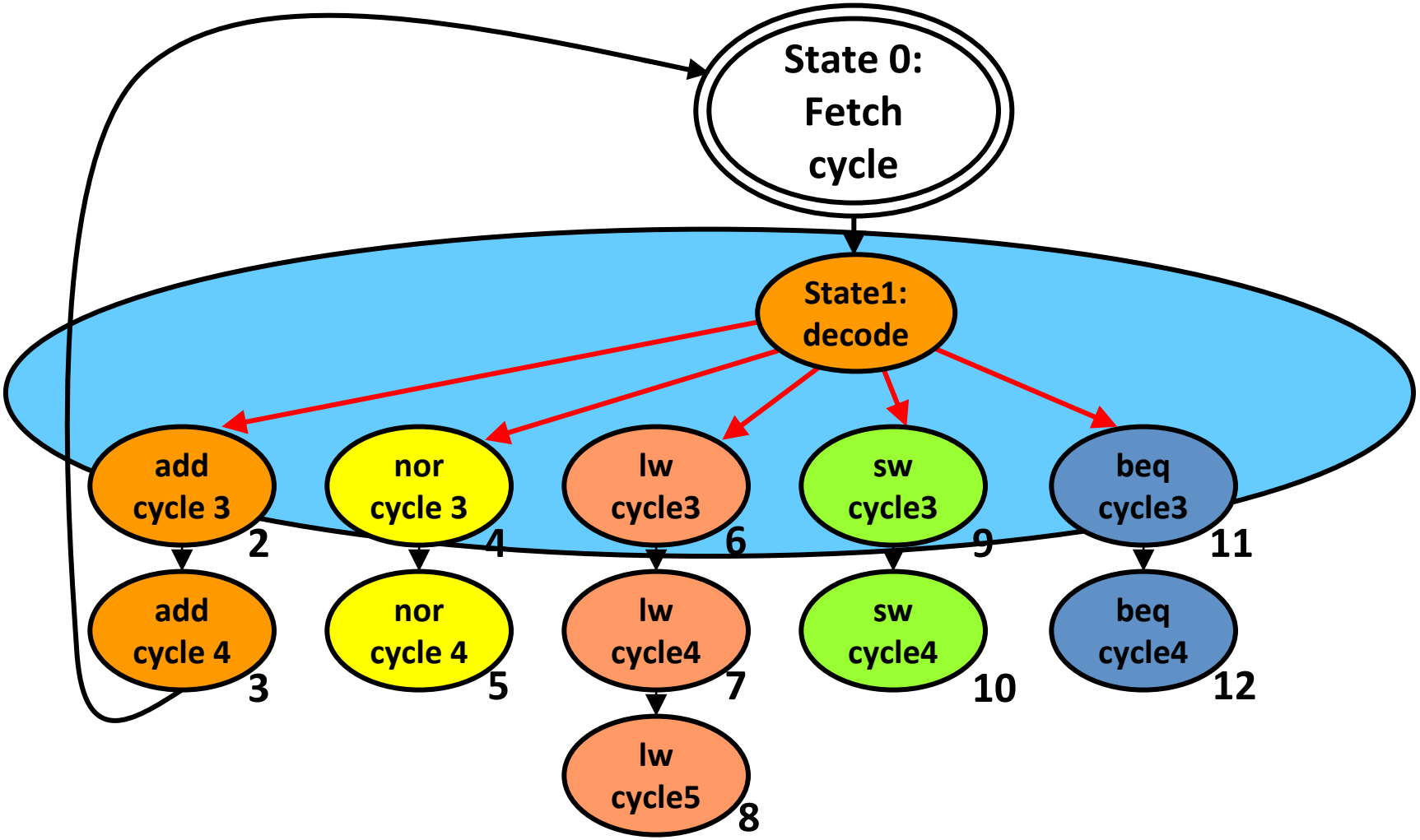


Control Rom for nor (4 and 5)

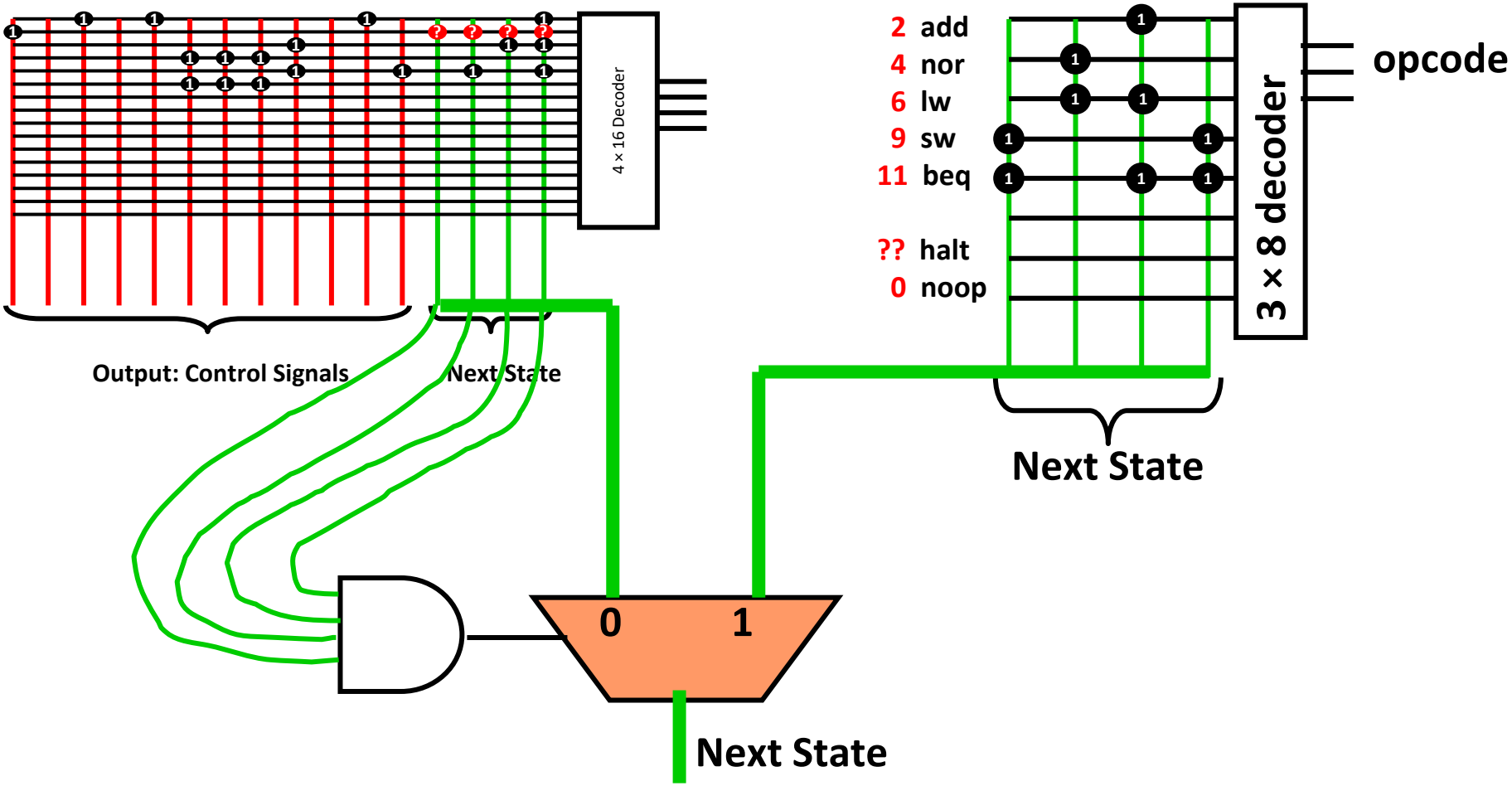
Poll: How many control bits are different between *nor (state 4)* and *add (state 2)*?



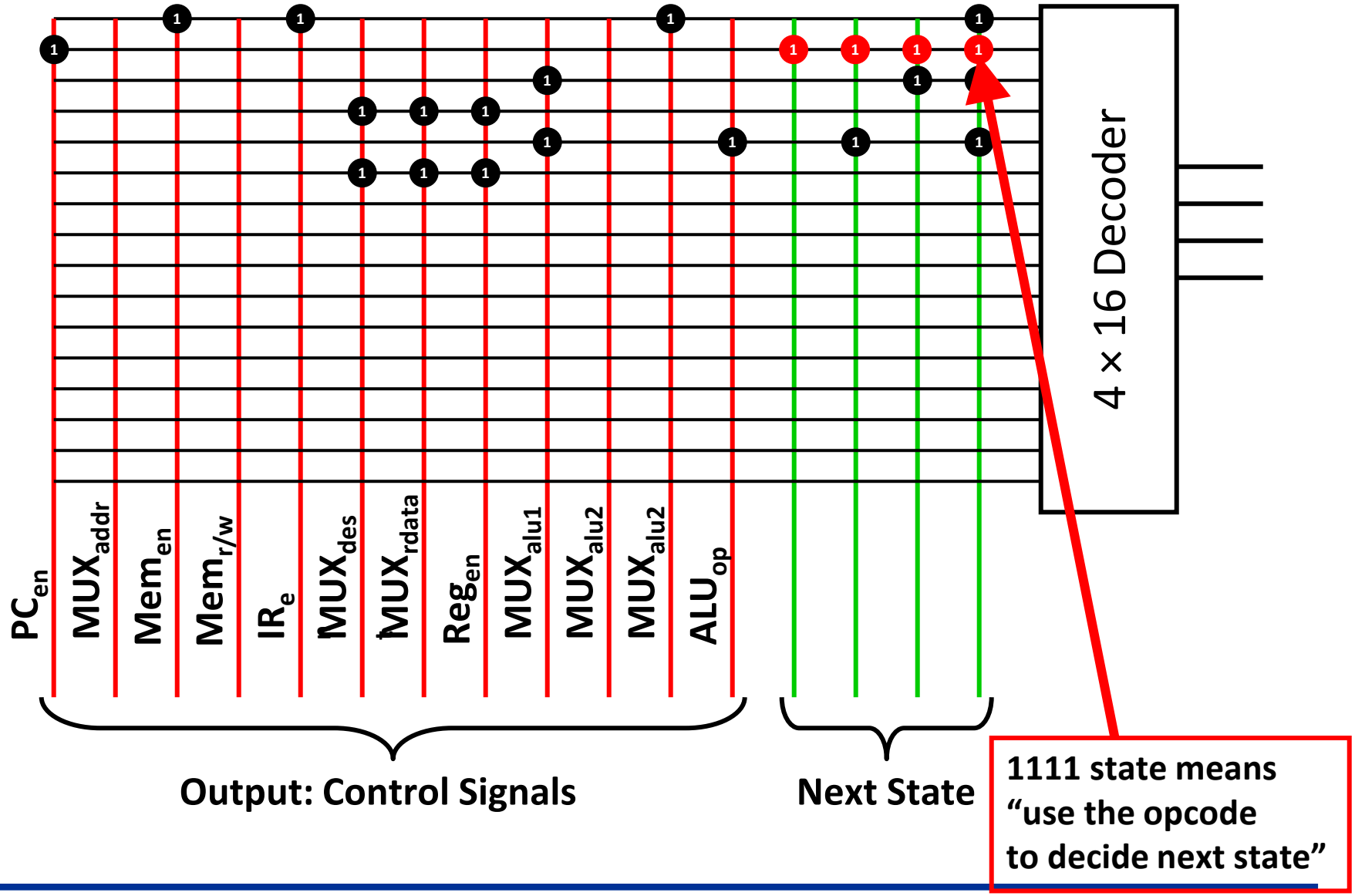
What about the transition from state 1?



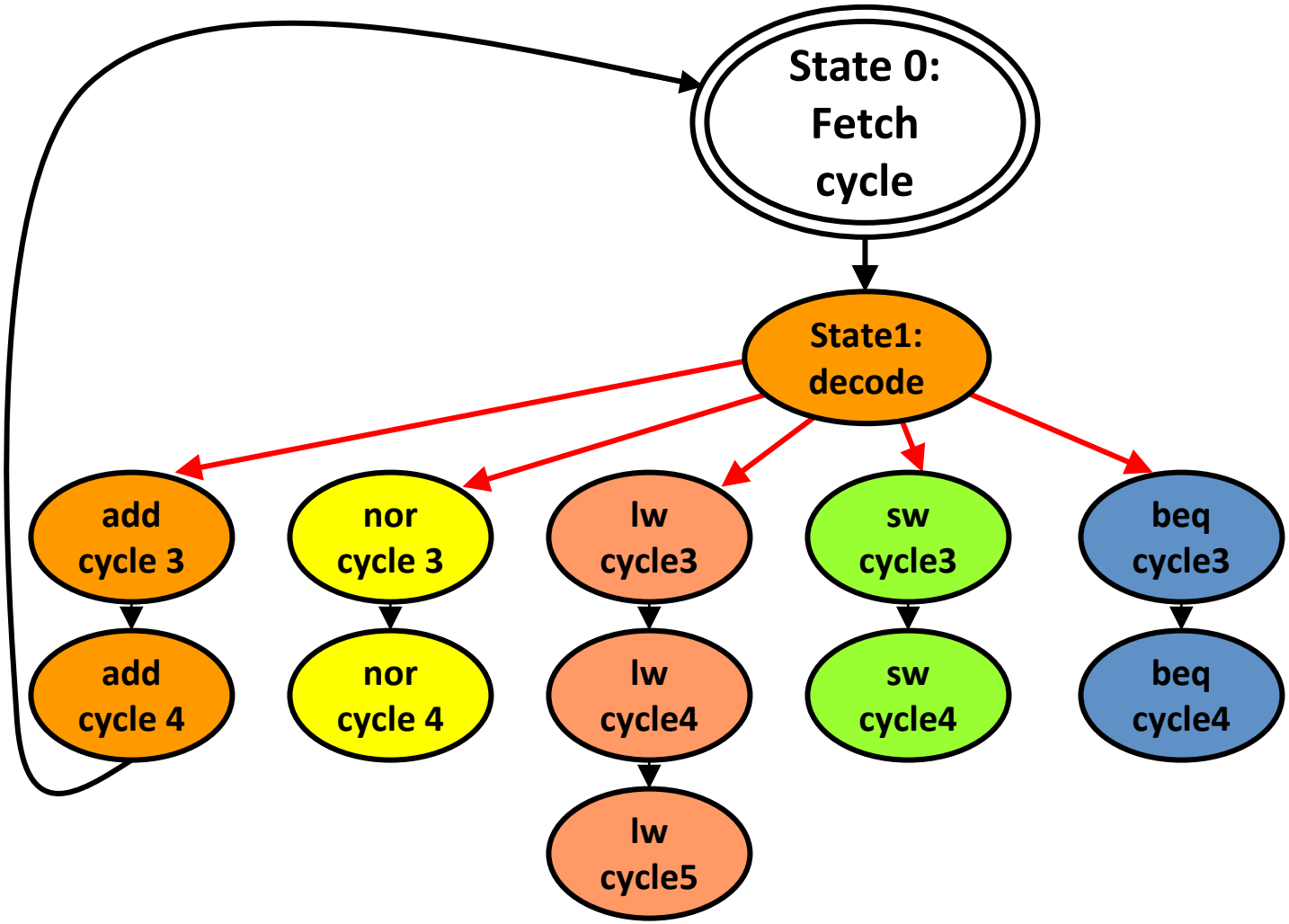
Complete transition function circuit



Control Rom (use of 1111 state)

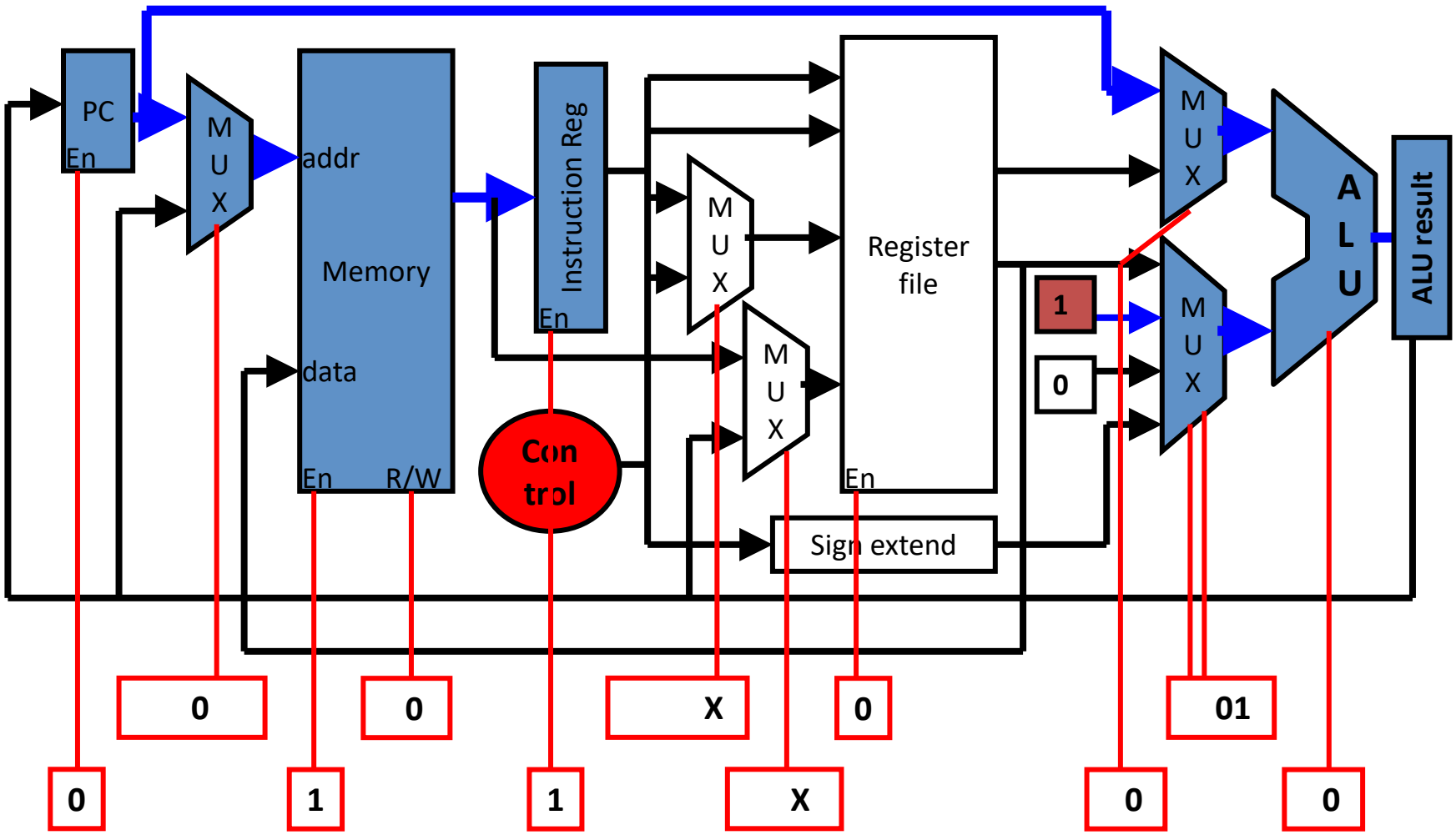


Return to State 0: Fetch cycle to execute the next instruction



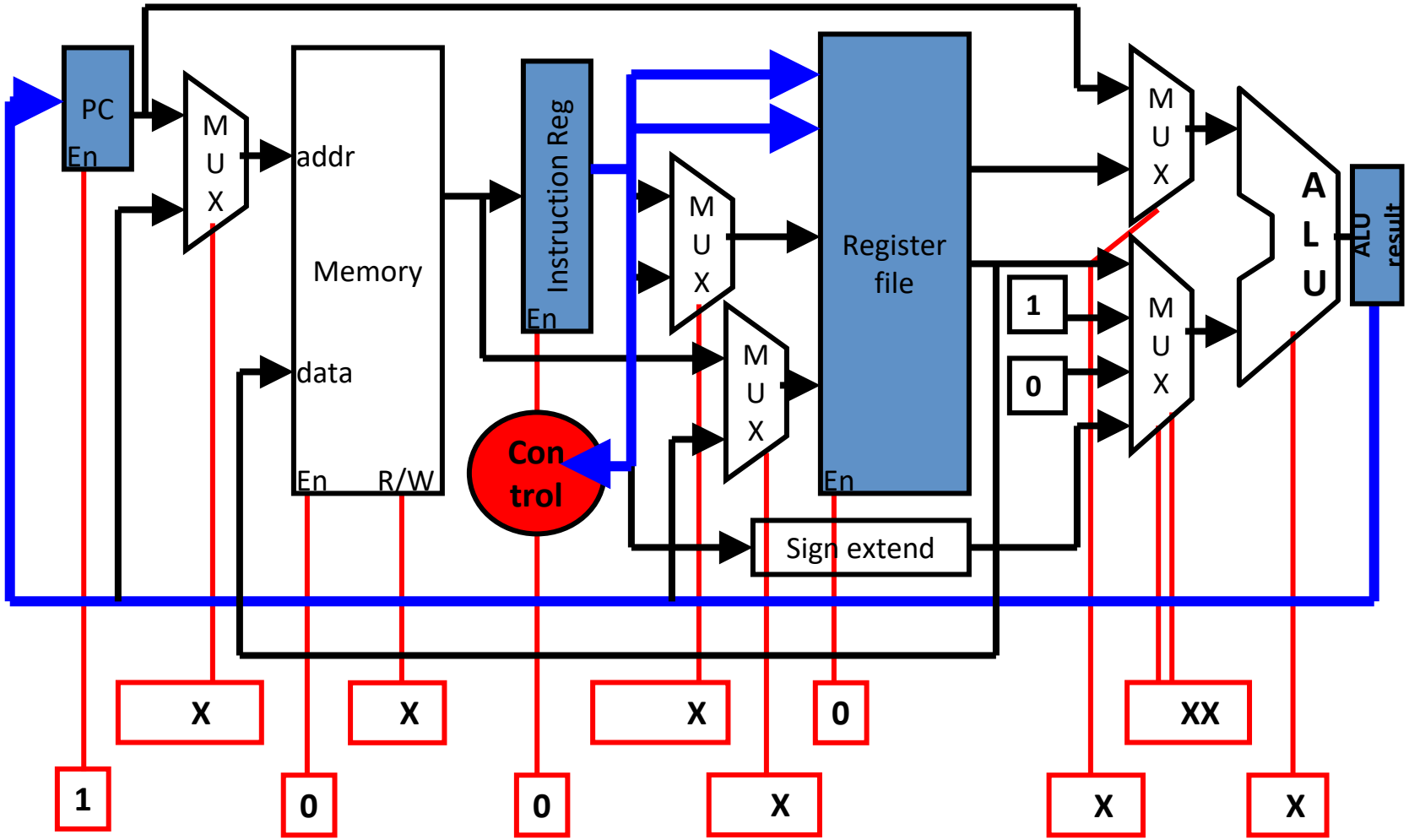
First Cycle (State 0) Operation

This is the same for all instructions
(since we don't know the instruction yet!)



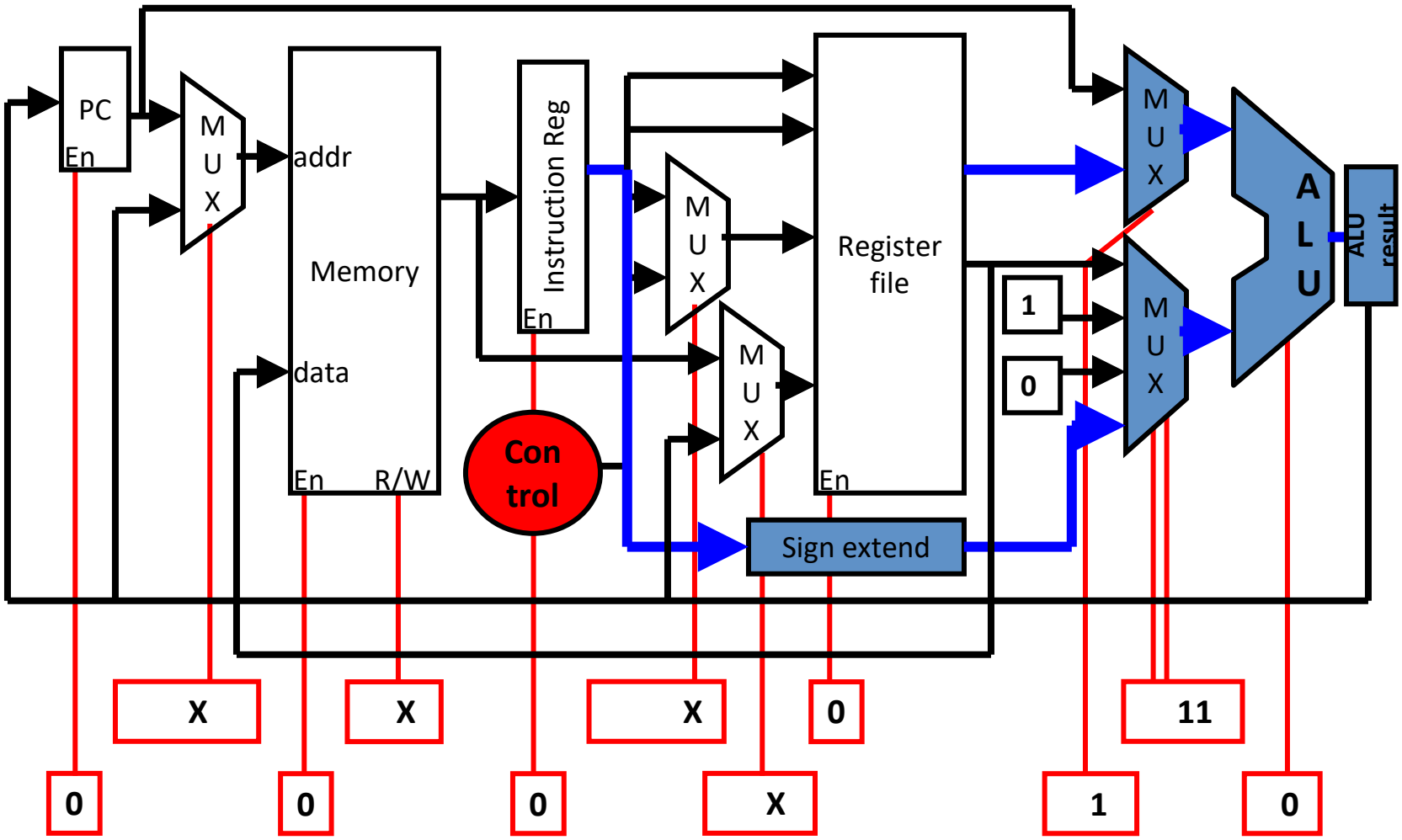
State 1: output function

Update PC; read registers (regA and regB);
use opcode to determine next state

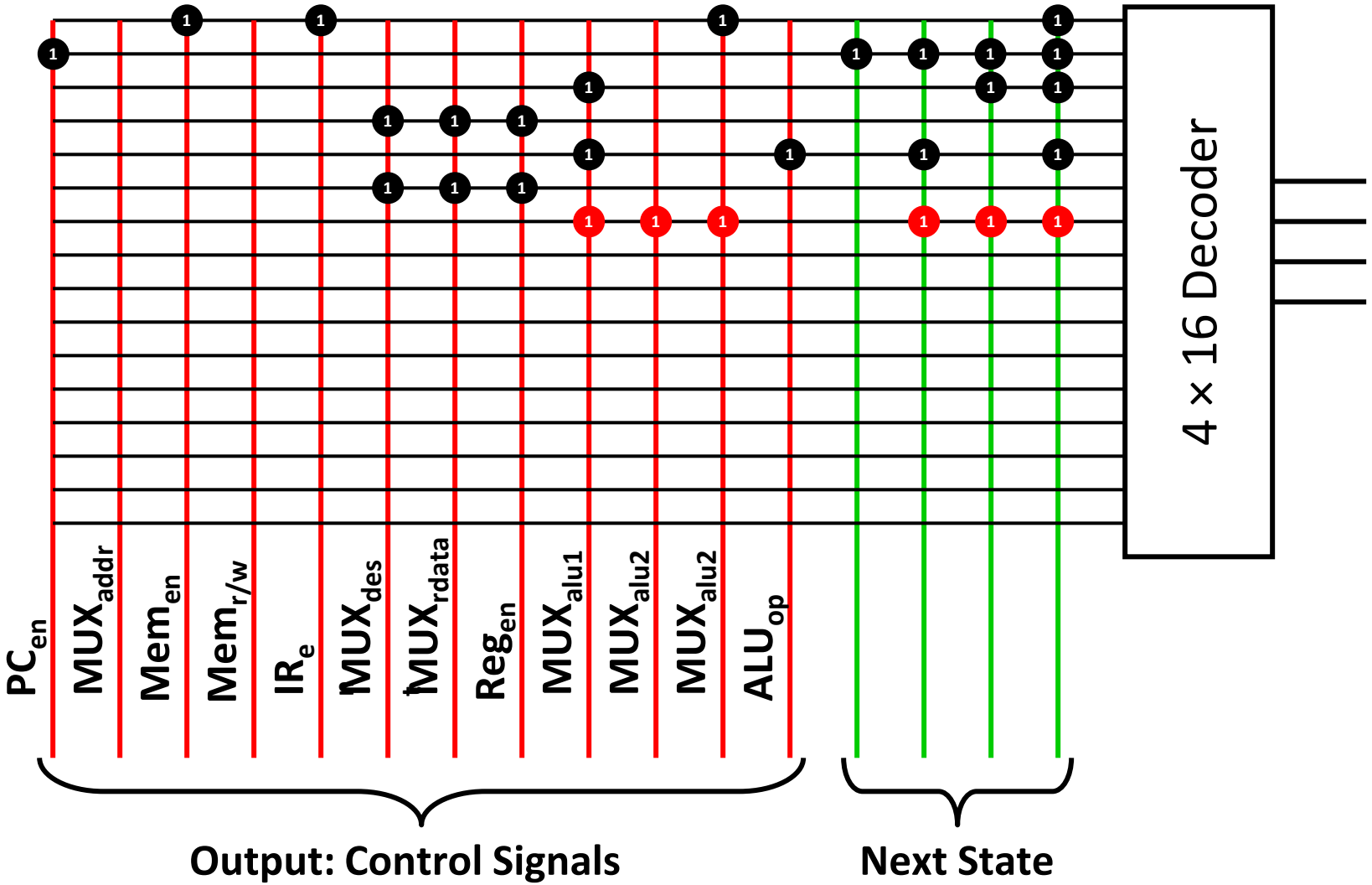


State 6: LW cycle 3

Calculate address for memory reference

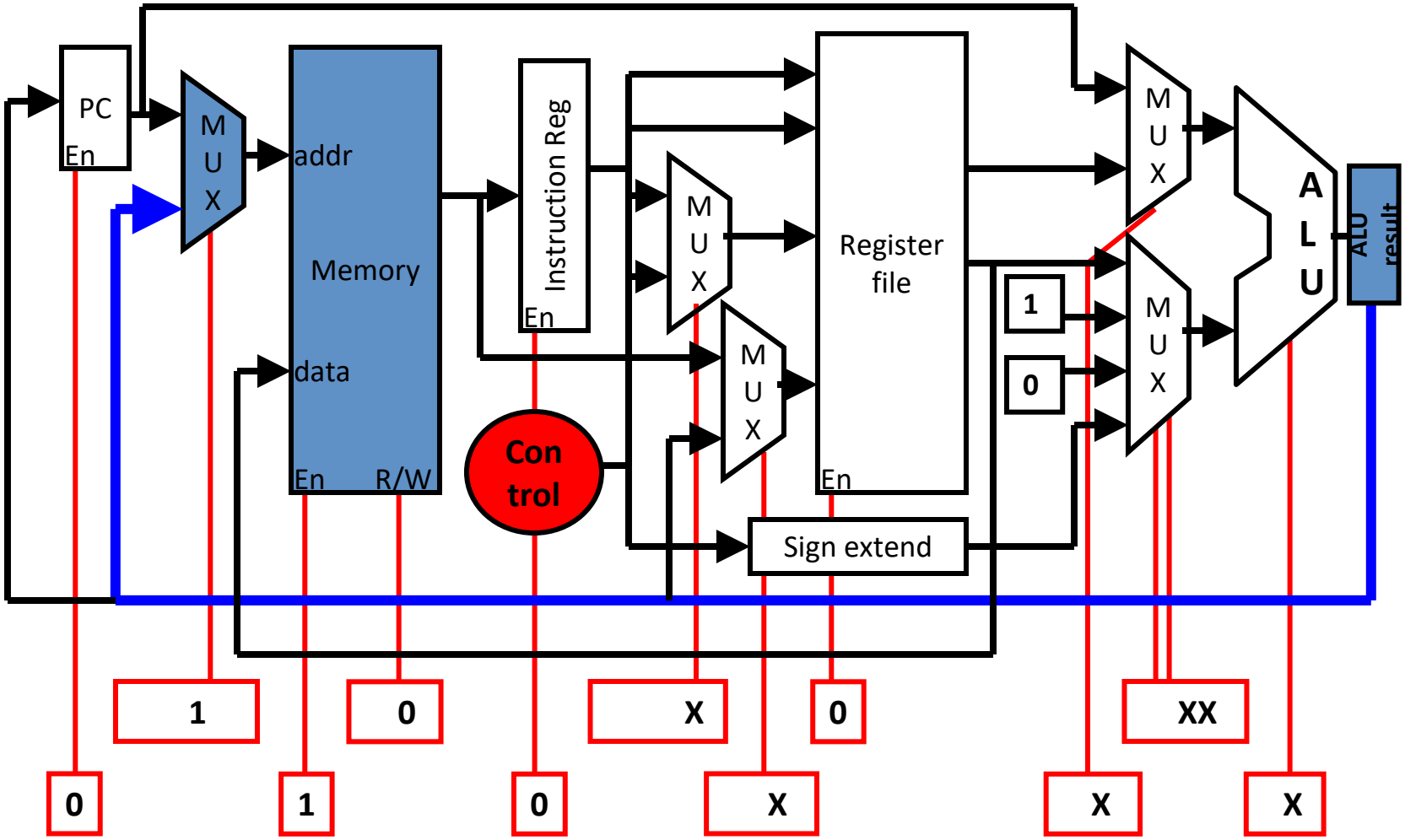


Control Rom (lw cycle 3)

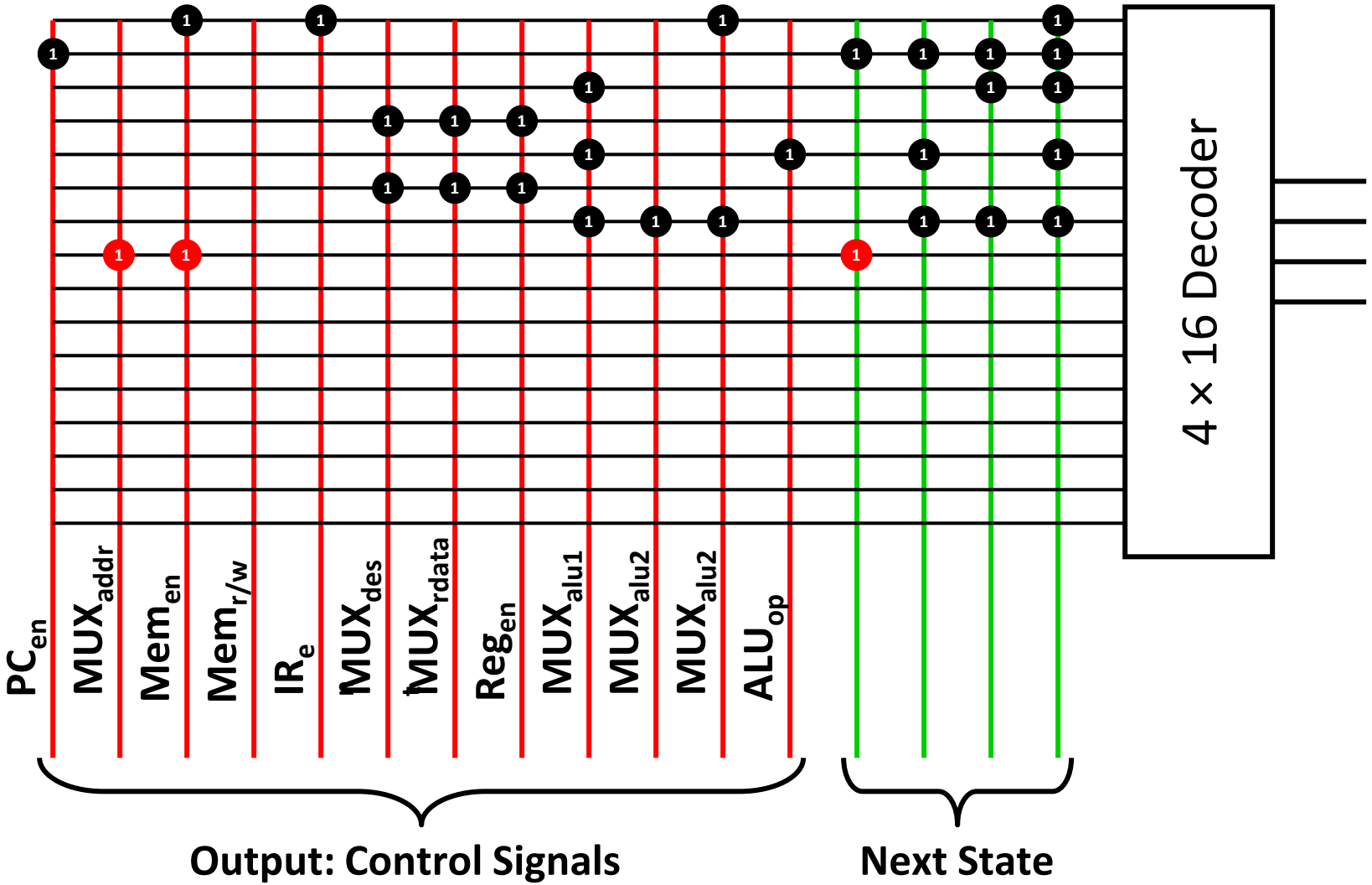


State 7: LW cycle 4

Read memory location

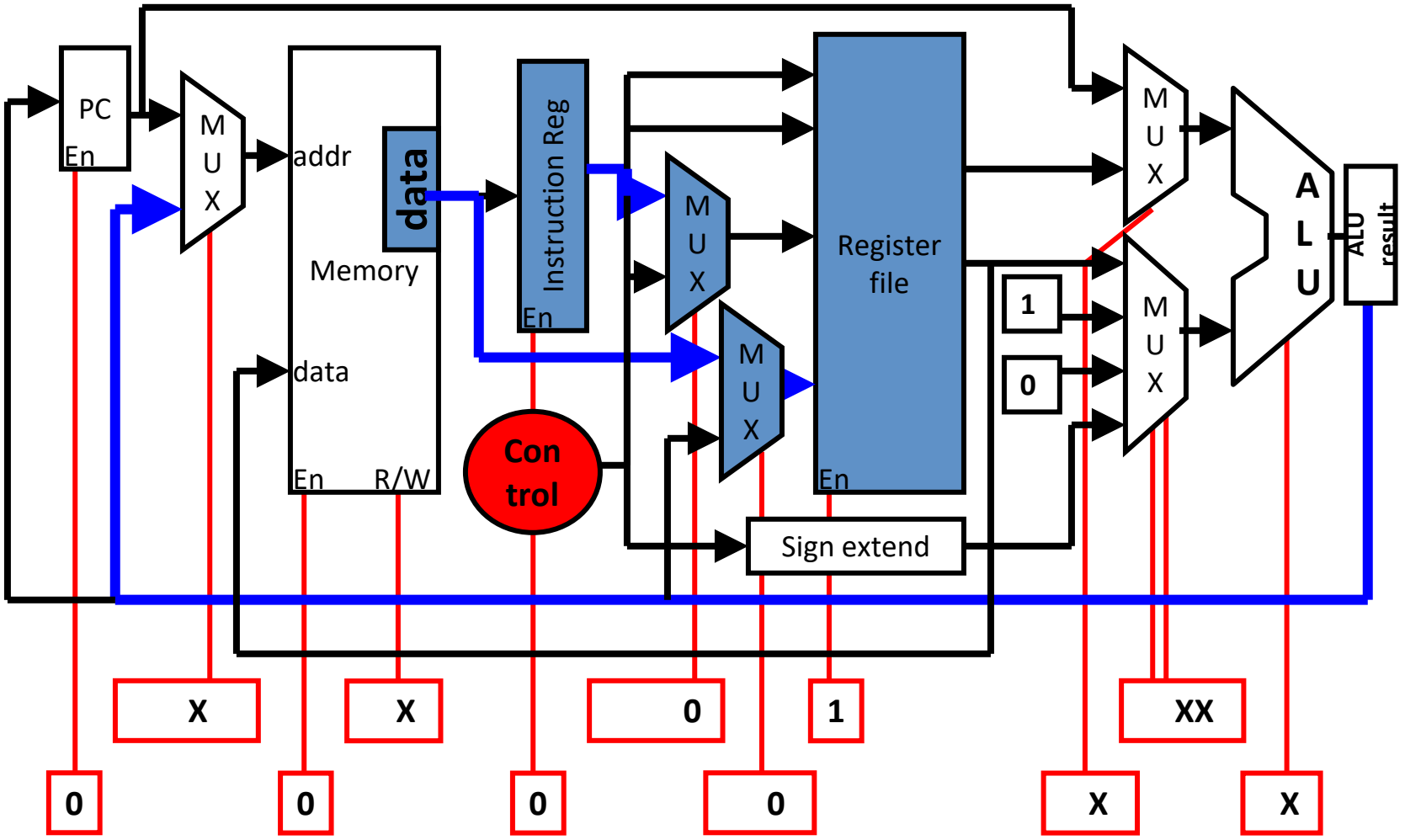


Control Rom (lw cycle 4)

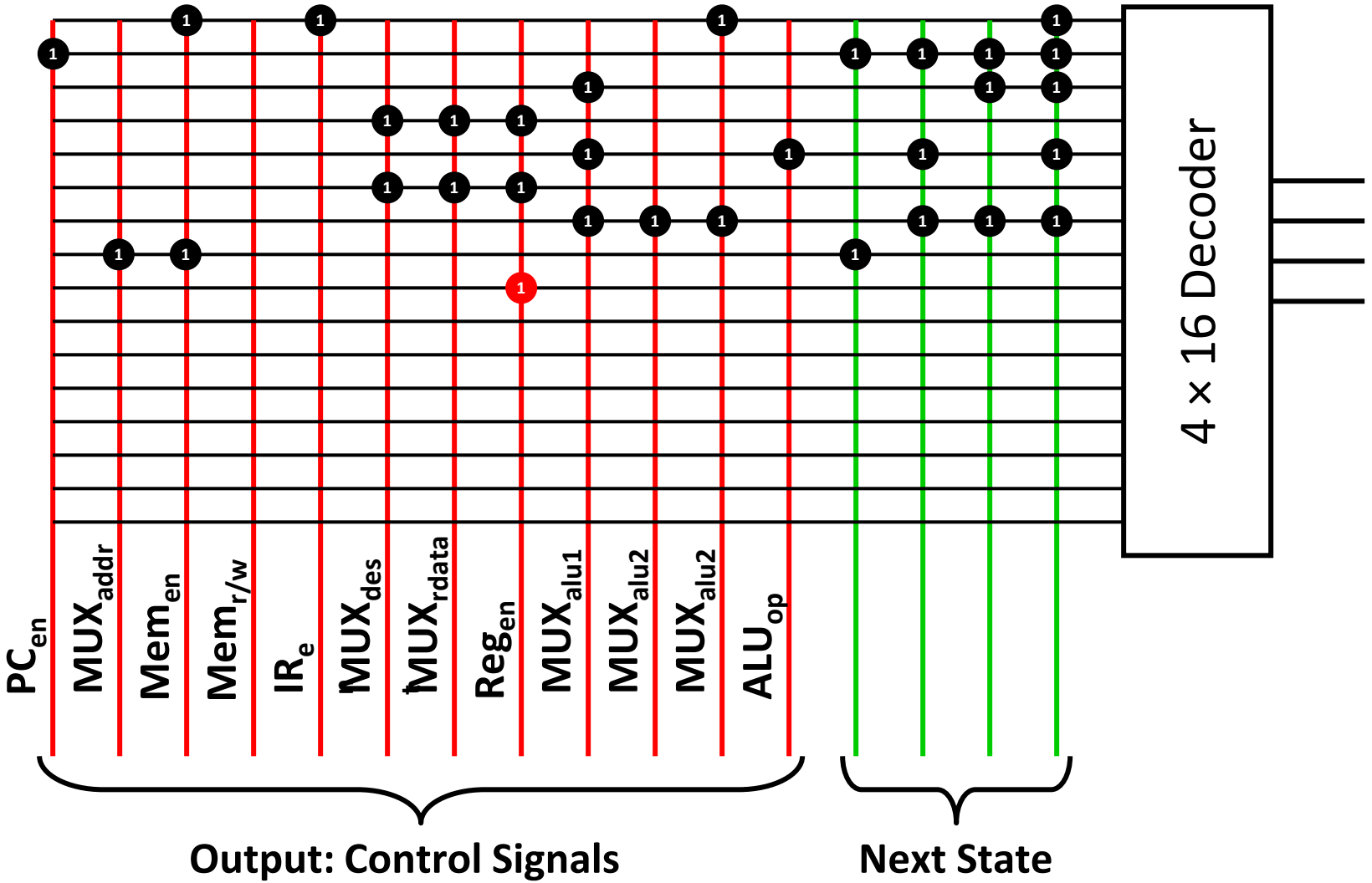


State 8: LW cycle 5

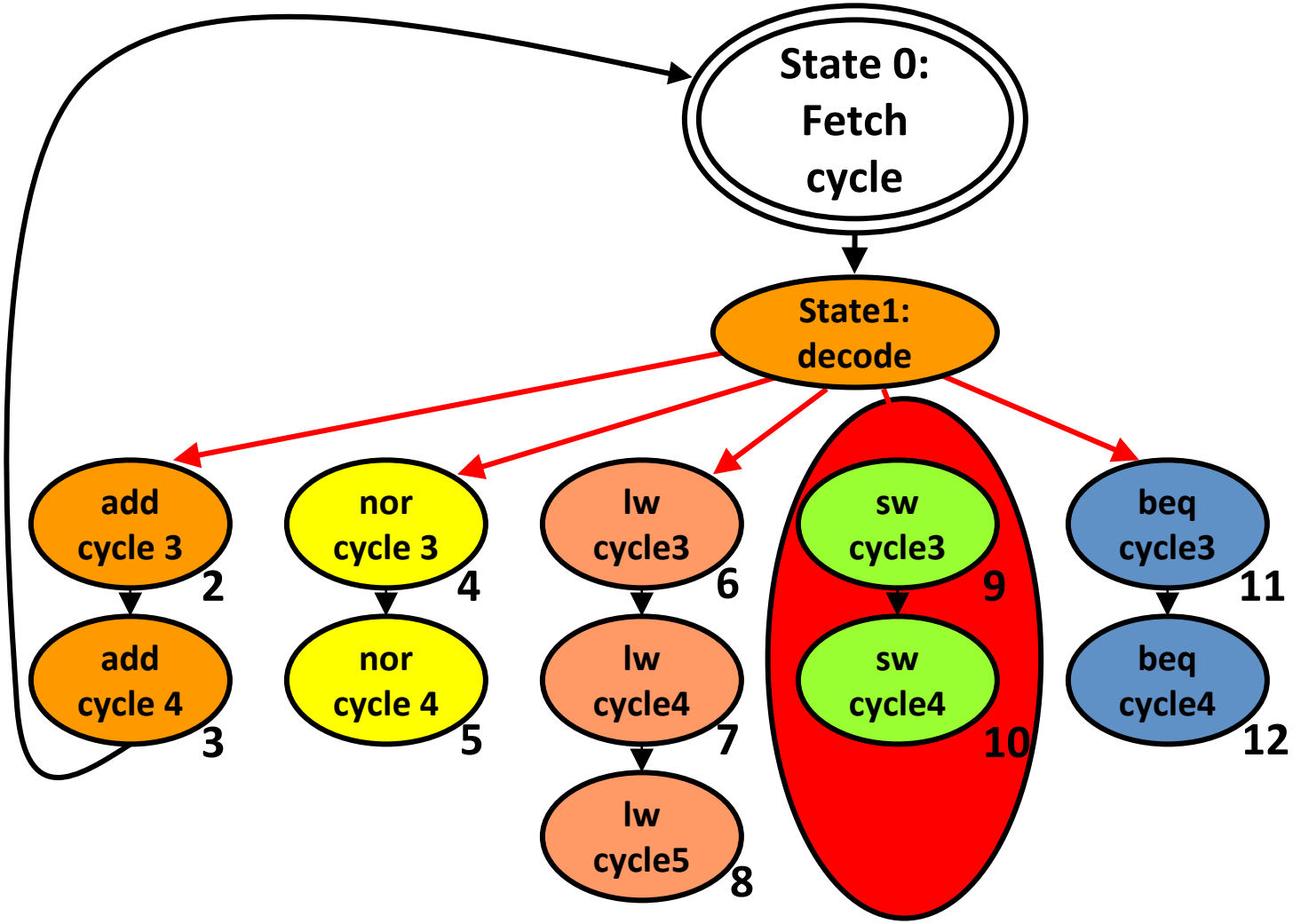
Write memory value to register file



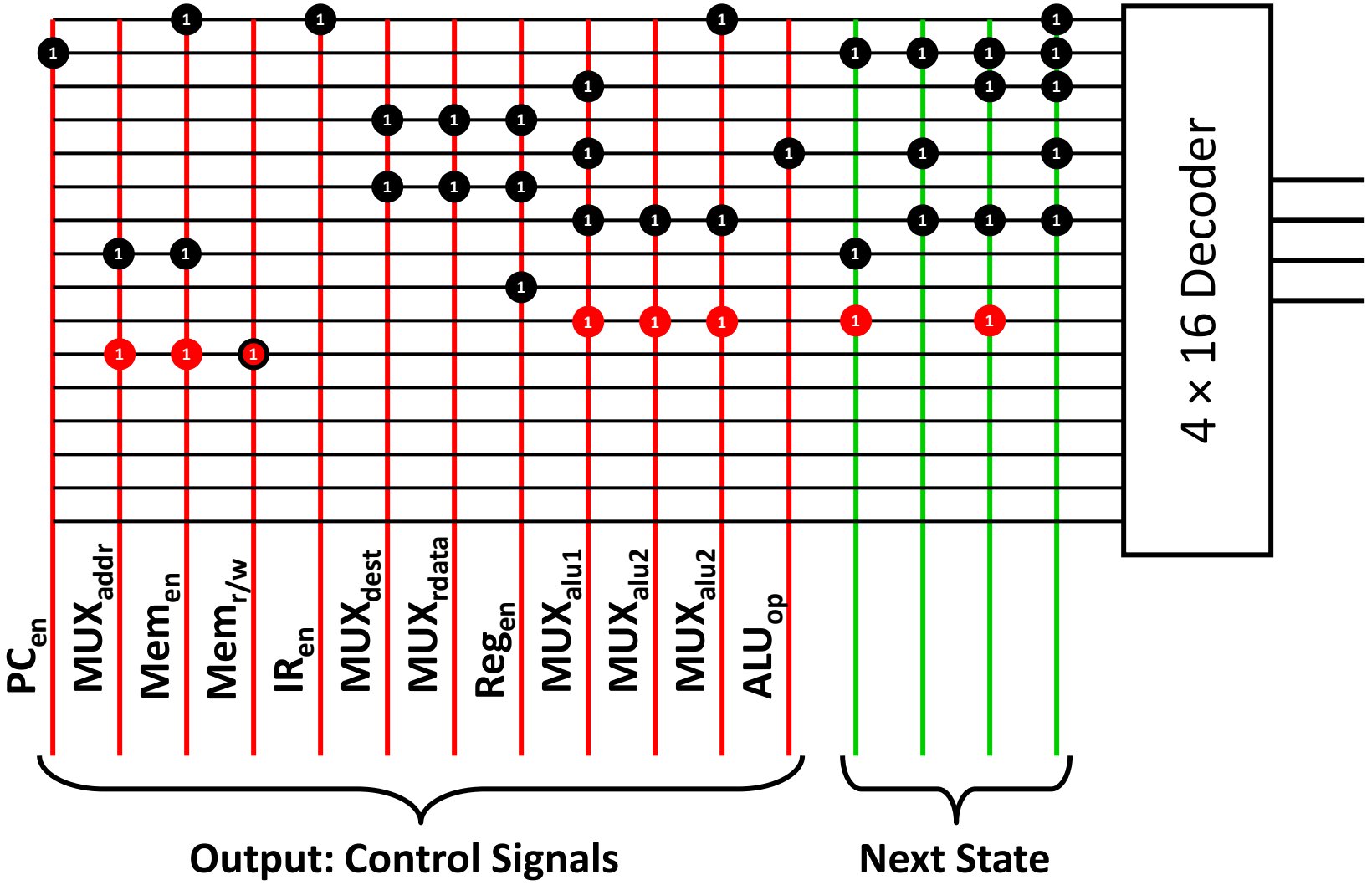
Control Rom (lw cycle 5)



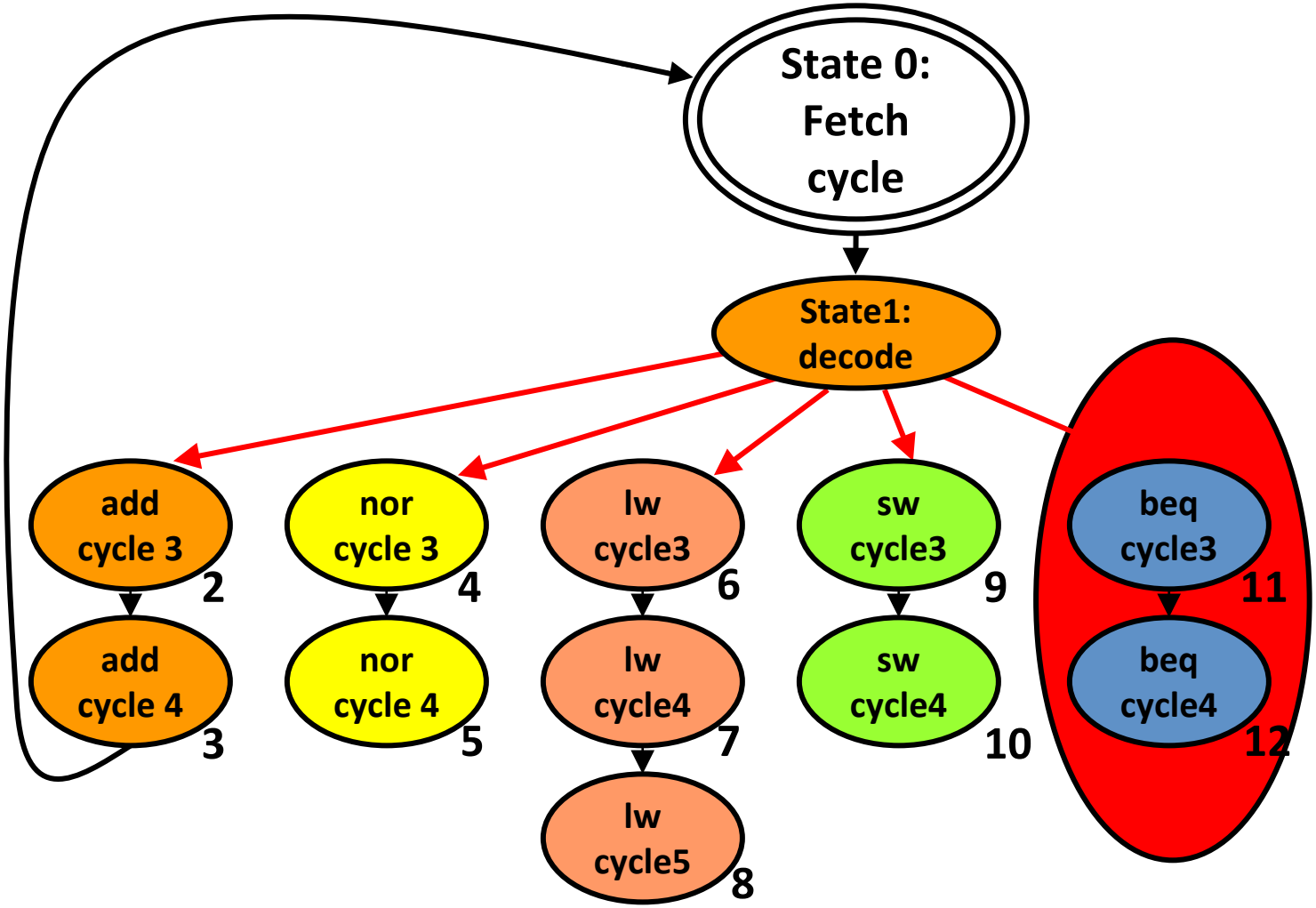
Return to State 0: Fetch cycle to execute the next instruction



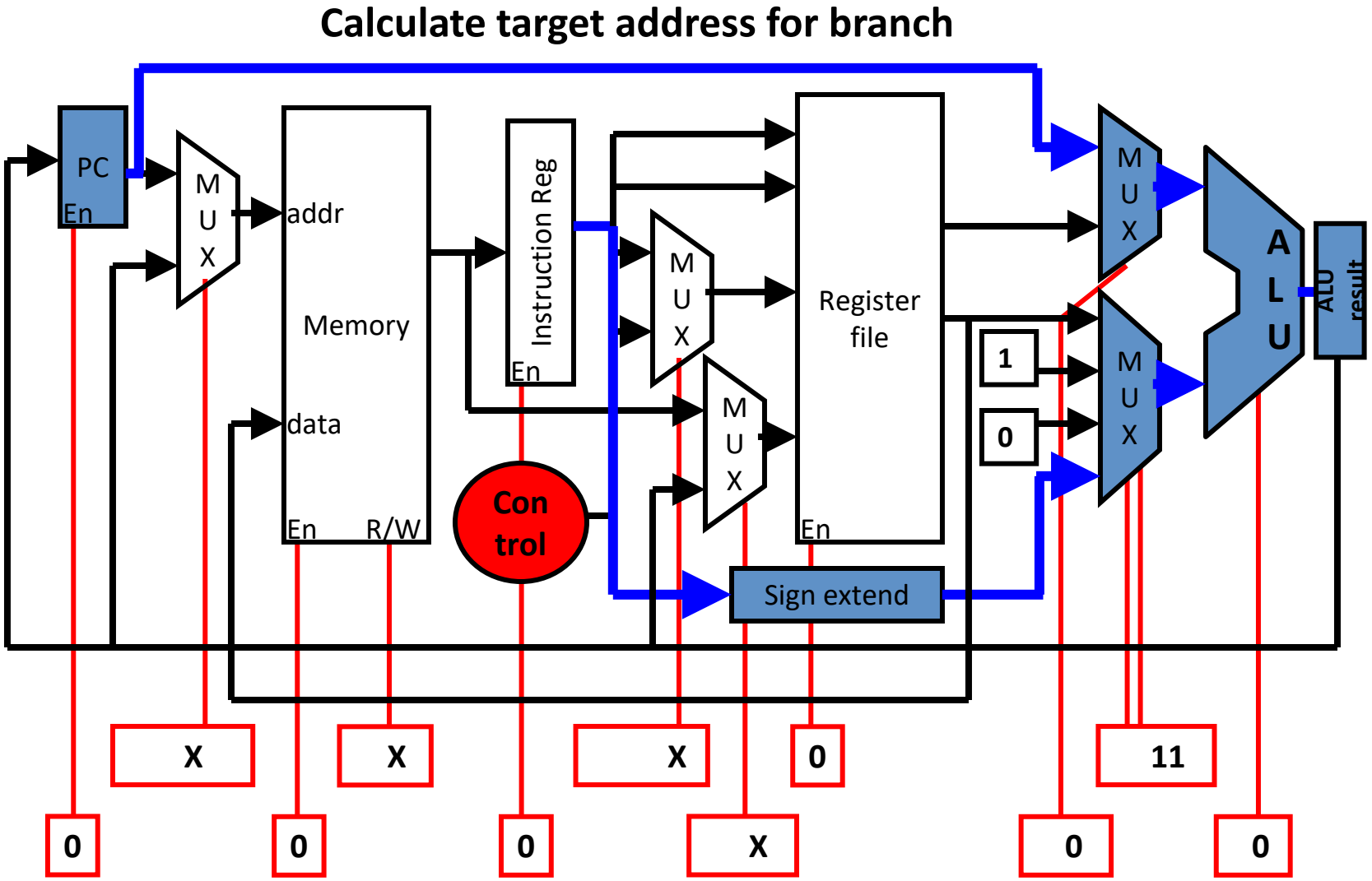
Control Rom (sw cycles 3 and 4)



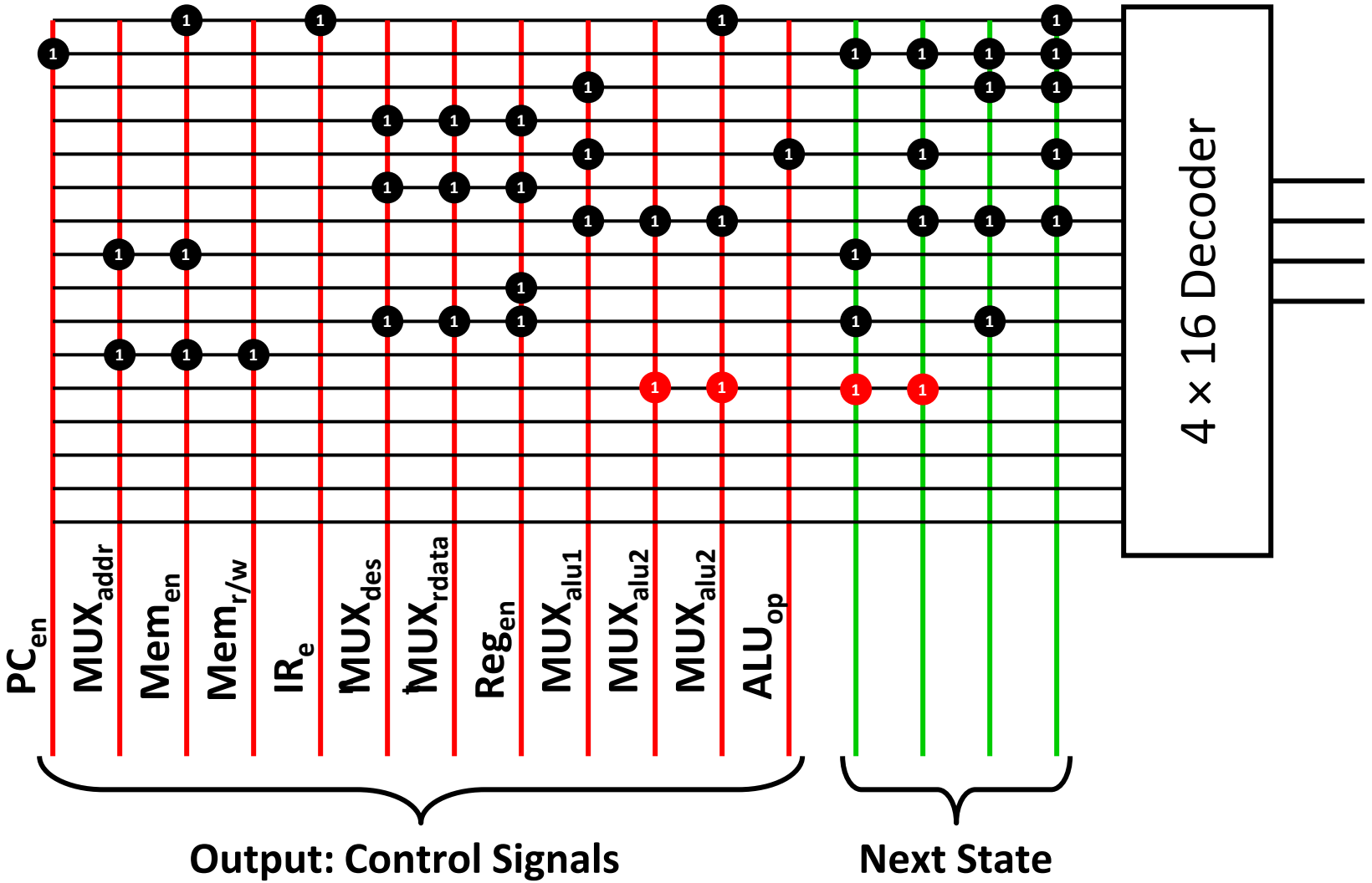
Return to State 0: Fetch cycle to execute the next instruction



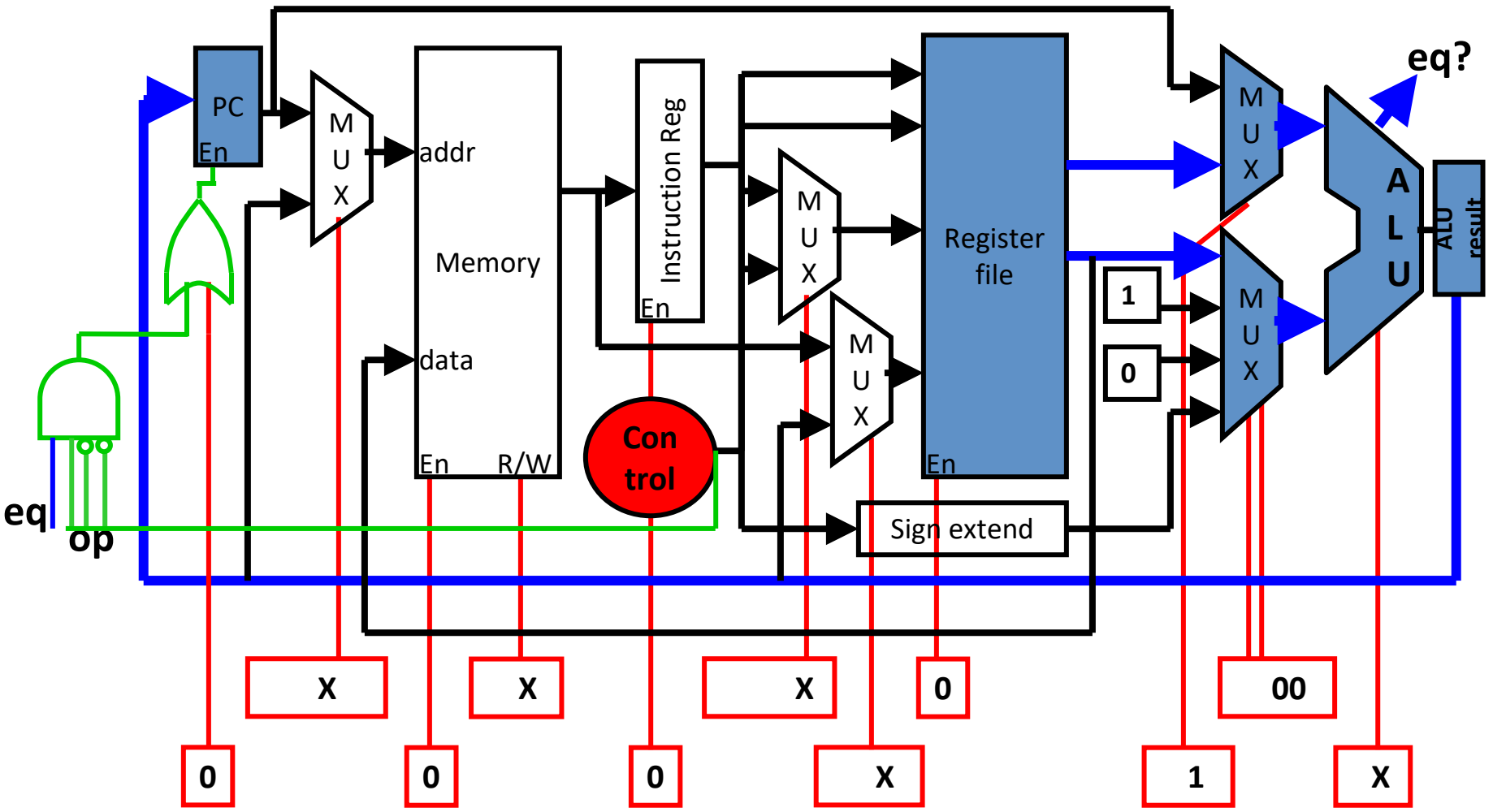
State 11:beq cycle 3（错误？？haha）



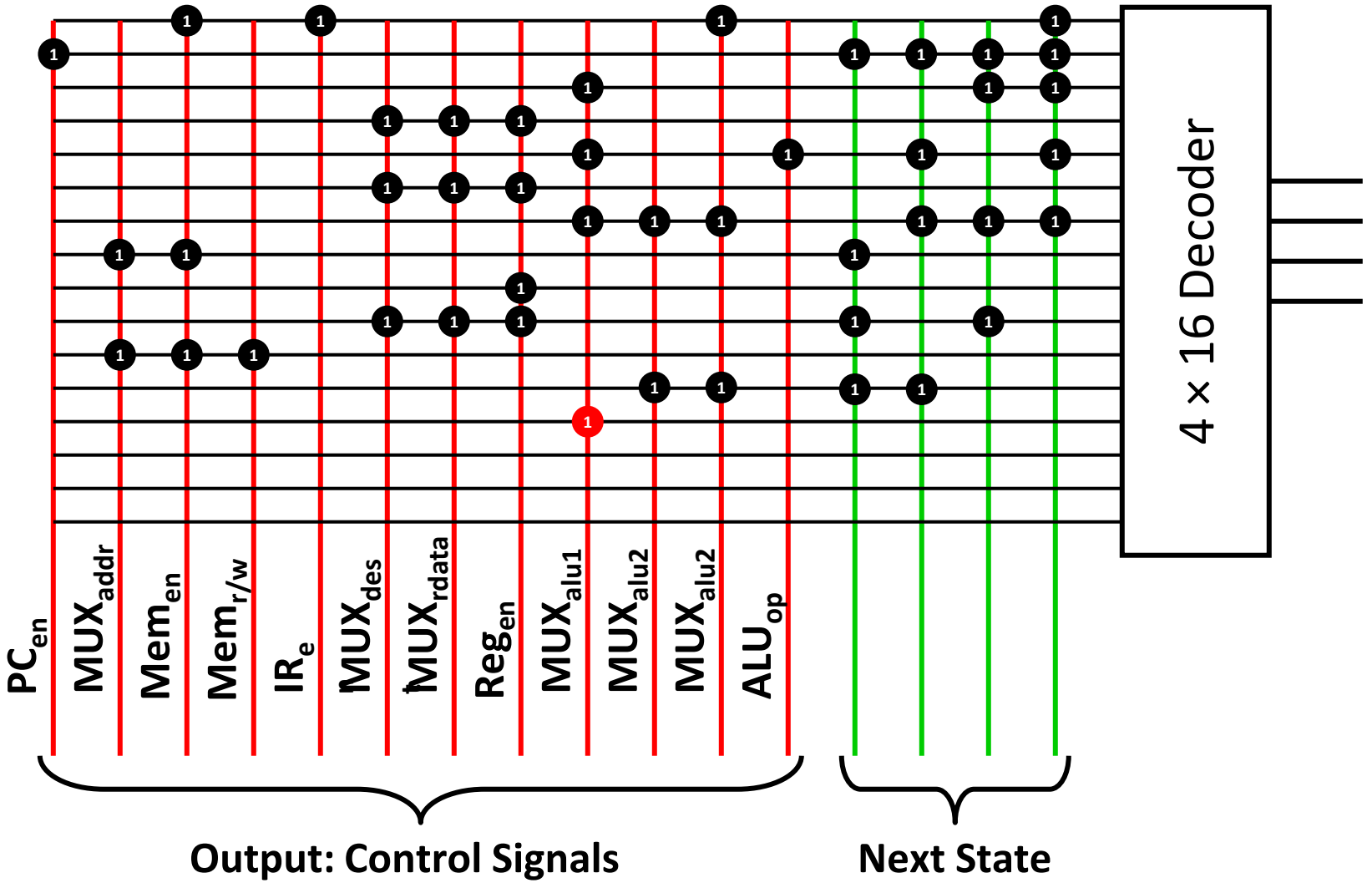
Control Rom (beq cycle 3)



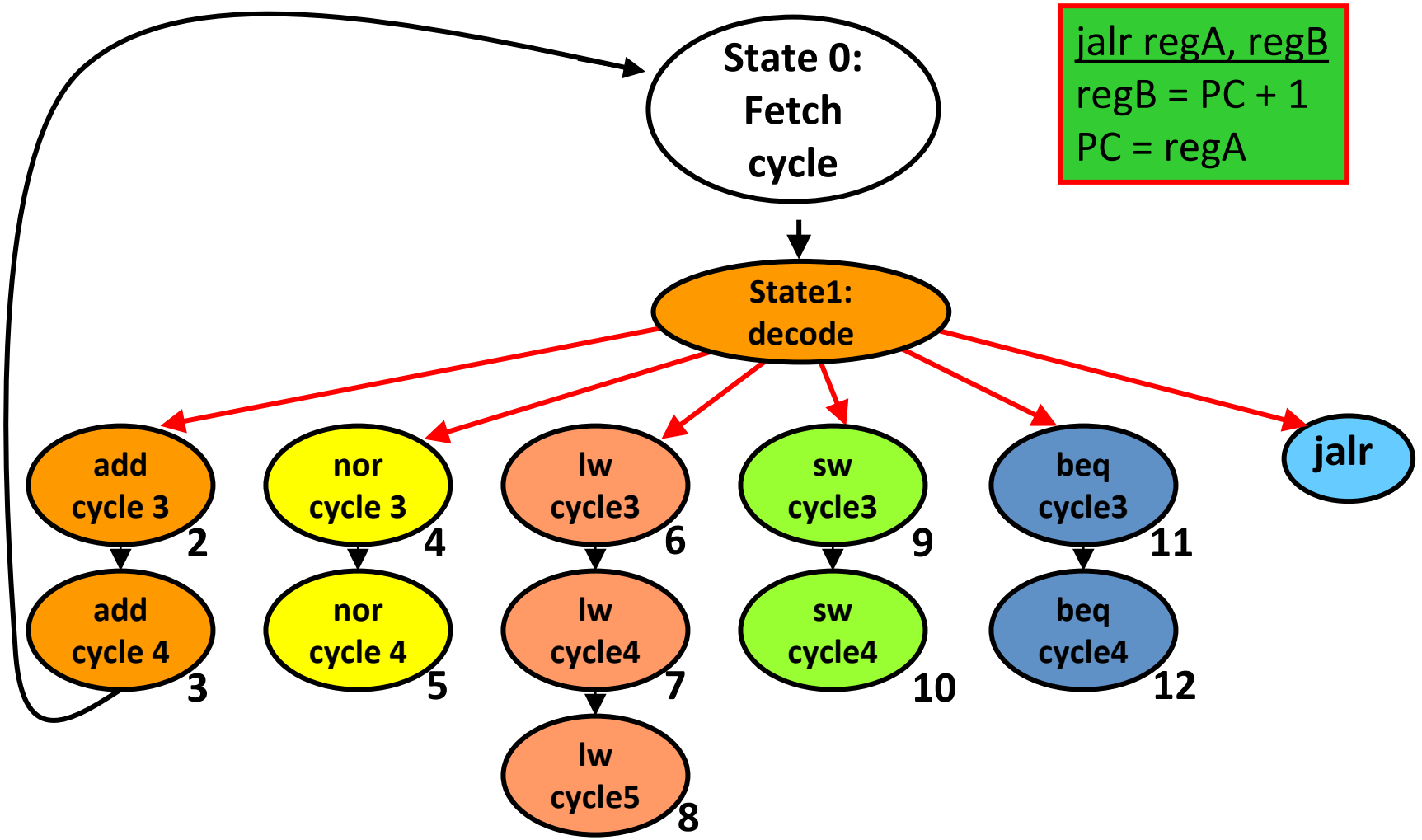
State 12:beq cycle 4（错误？？HaHa）



Control Rom (beq cycle 4)



OK, what about the JALR instruction?



Single and Multicycle Performance

1 ns – Register read/write time

2 ns – ALU/adder

2 ns – memory access

0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

SC: 2 + 1 + 2 + 2 + 1 = 8 ns

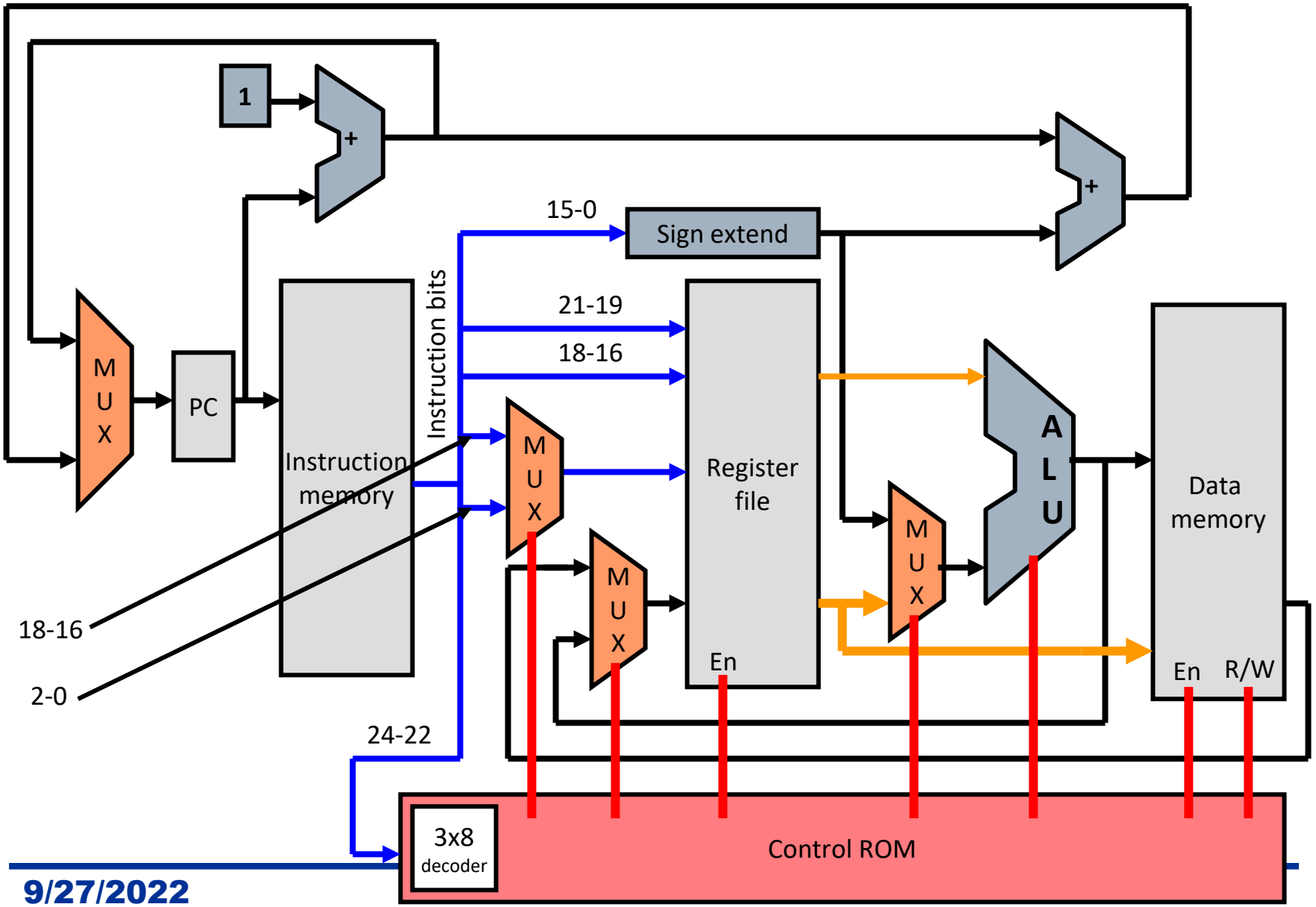
MC: MAX(2, 1, 2, 2, 1) = 2ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

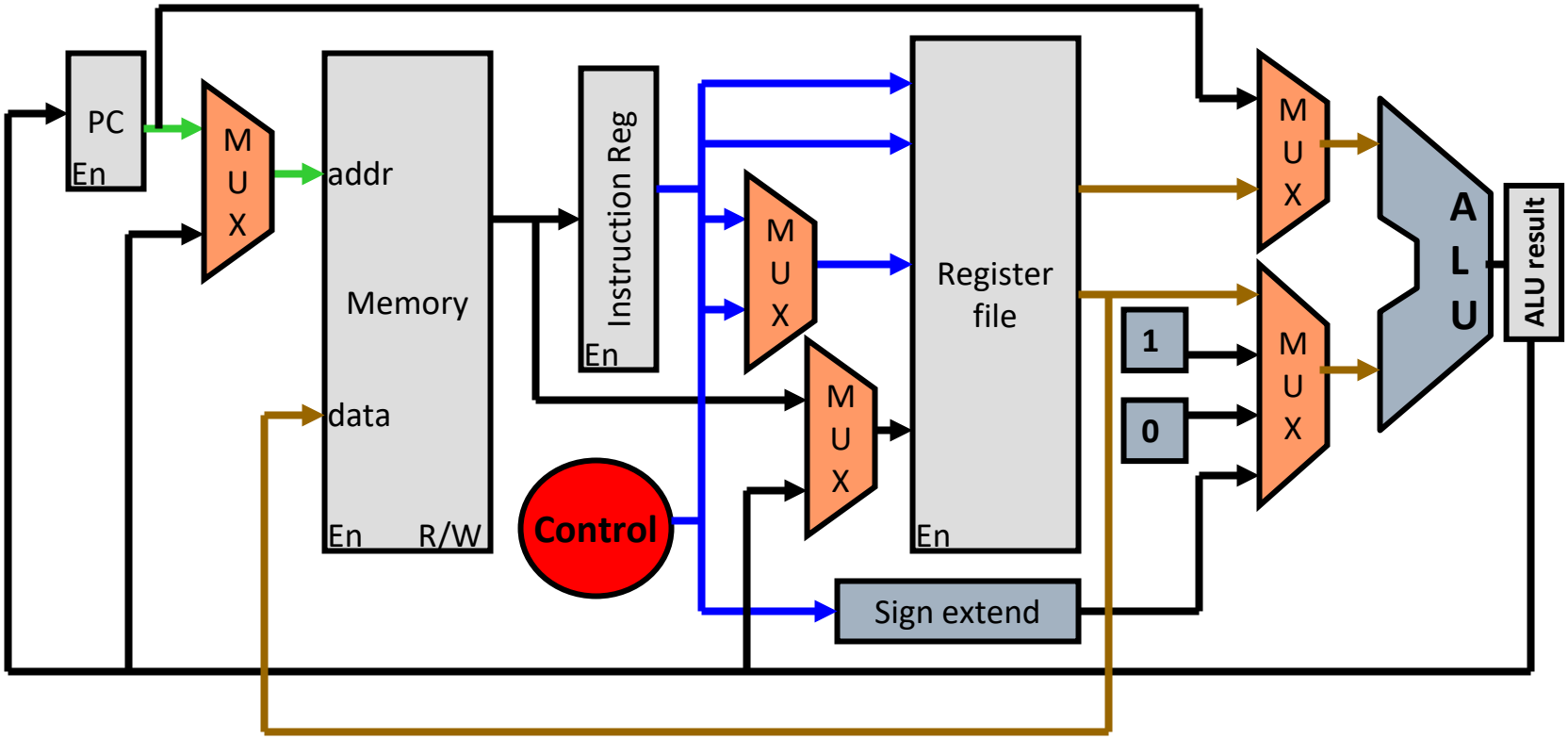
SC: 100 cycles * 8 ns = 800 ns

MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

Review: Single-Cycle LC2Kx Datapath



Review: Multi-Cycle LC2Kx Datapath



Review: Single and Multicycle Performance

1 ns – Register File read/write time
2 ns – ALU/adder
2 ns – memory access
0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

Single and Multicycle Performance

1 ns – Register read/write time
2 ns – ALU/adder
2 ns – memory access
0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

SC: 2 + 1 + 2 + 2 + 1 = 8 ns
MC: MAX(2, 1, 2, 2, 1) = 2ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

SC: 100 cycles * 8 ns = 800 ns
MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

Review: Single and Multicycle Performance—Questions from last time

1 ns – Register read/write time
2 ns – ALU/adder
2 ns – memory access
0 ns – MUX, PC access, sign extend, ROM

1. Assuming the above delays, what is the best cycle time that the LC2k multicycle datapath could achieve?

MC: MAX(2, 1, 2, 2, 1) = 2ns
SC: 2 + 1 + 2 + 2 + 1 = 8 ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

SC: 100 cycles * 8 ns = 800 ns
MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

3. So what good is MC???

Review: Single and Multicycle Performance—Questions from last time

2 ns – Register read/write time

2 ns – ALU/adder

2 ns – memory access

0 ns – MUX, PC access, sign extend, ROM

1.What if the register file access is increased to 2ns, does that change the answer to the previous question?

MC: MAX(2, 2, 2, 2, 2) = 2ns

SC: 2 + 2 + 2 + 2 + 2 = 10 ns

2. Assuming the above delays, for a program consisting of 25 LW, 10 SW, 45 ADD, and 20 BEQ, which is faster?

SC: 100 cycles * 10 ns = 1000 ns

MC: (25*5 + 10*4 + 45*4 + 20*4)cycles * 2ns = 850 ns

3. Balancing delays helps MC