10. Basic processor design – multicycle datapaths

EECS 370 – Introduction to Computer Organization

Fall 2013

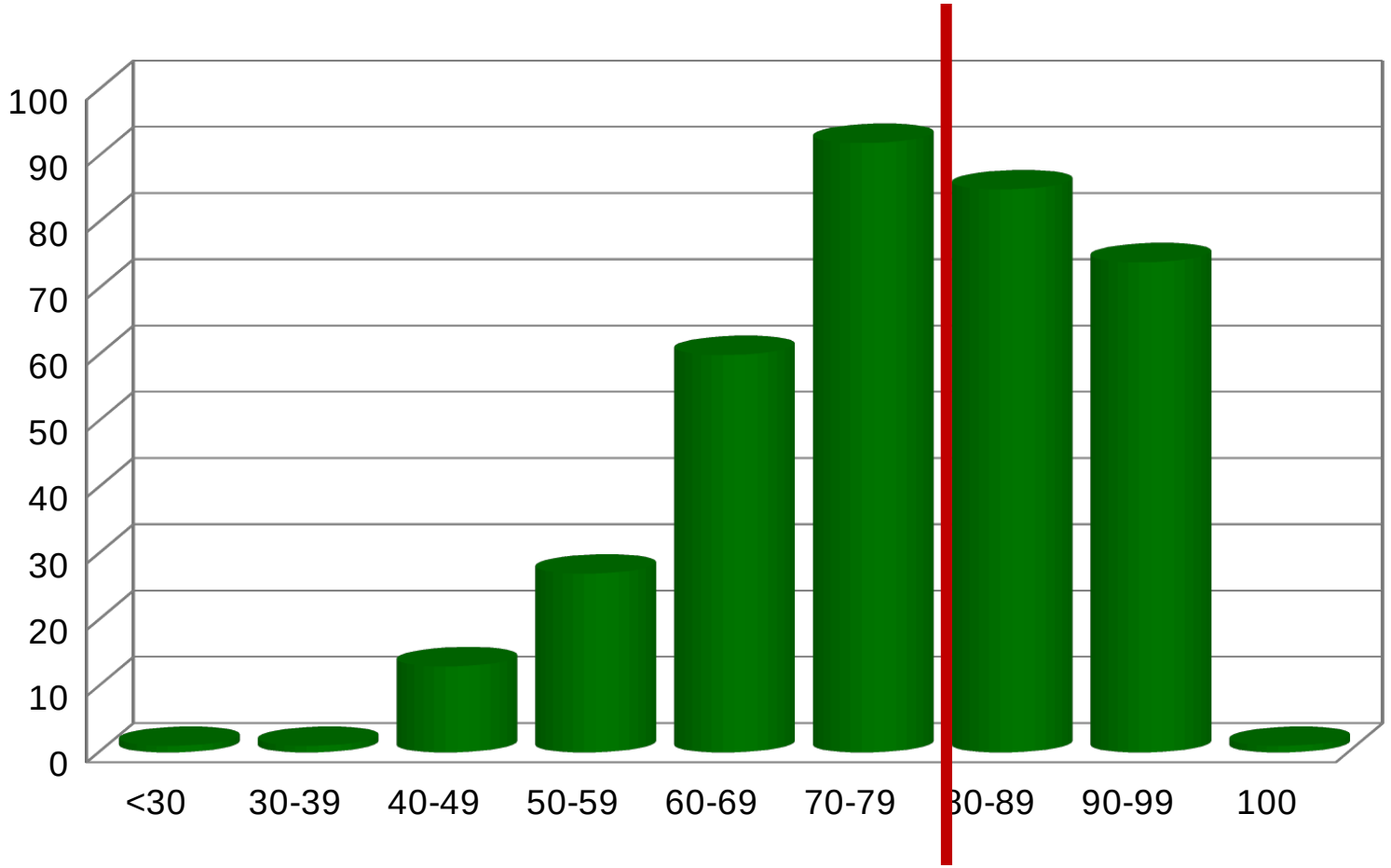
Profs. Bertacco, Robert Dick & Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

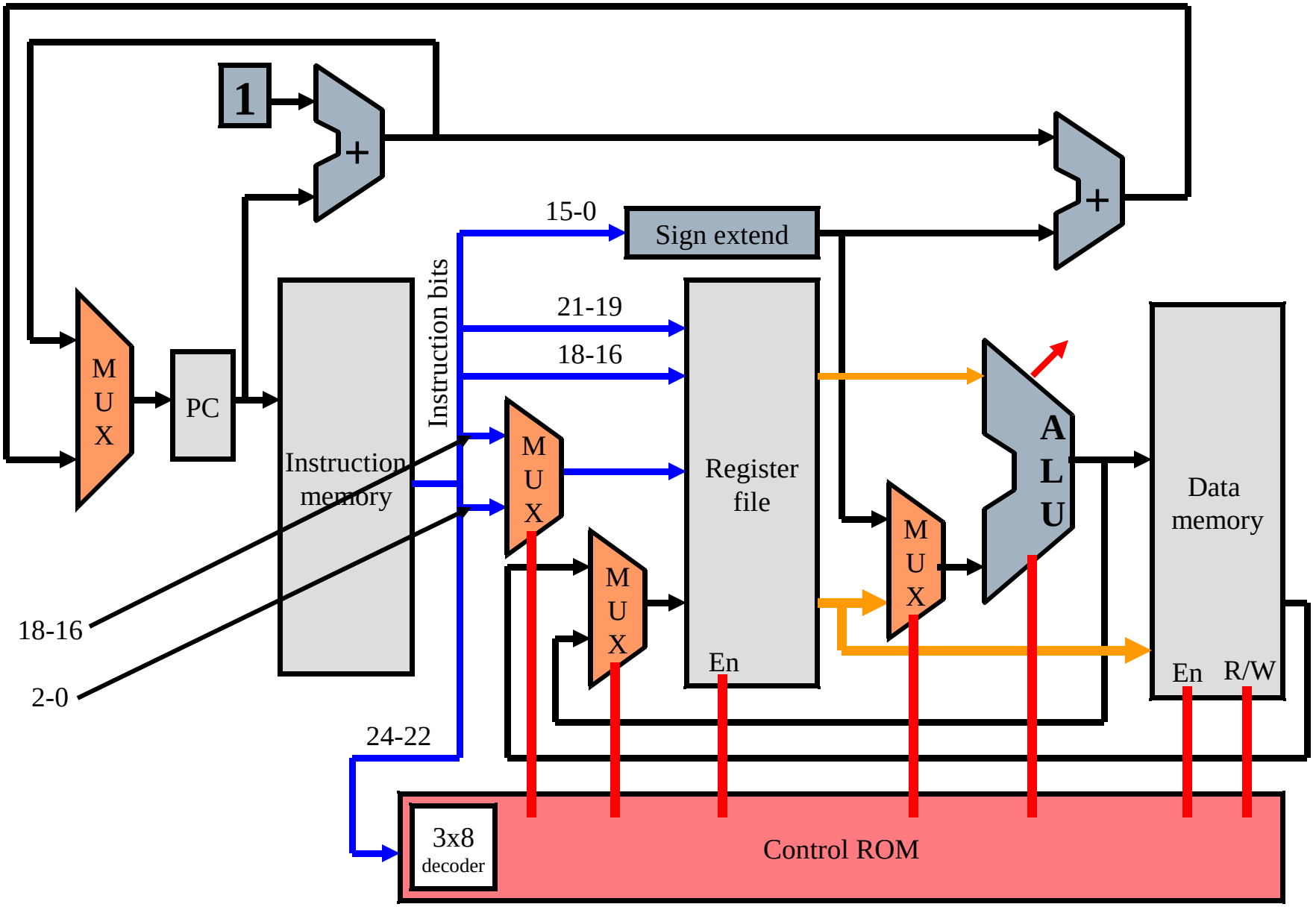
Announcements

- Homework 4 due Due 24 October
- Project 2 due 25 October
- Exam graded

Midterm Stats: avg 78 / median 79.5 / std. Dev. 14



Recap (LC2Kx single cycle datapath)



The problem with single cycle datapaths

1 ns – Register read/write time

2 ns – ALU/adder

2 ns – memory access

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

Instr.	Get instr.	Read reg.	ALU	Mem.	Write reg.	Total
add	2 ns	1 ns	2 ns	0 ns	1 ns	6 ns
beq	2 ns	1 ns	2 ns	0 ns	0 ns	5 ns
sw	2 ns	1 ns	2 ns	2 ns	0 ns	7 ns
lw	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns

Recap: execution time

Assume 100 instructions executed

- 25% of instructions are loads,
- 10% of instructions are stores,
- 45% of instructions are adds, and
- 20% of instructions are branches.

Single-cycle execution:

- $100 \cdot 8ns = \underline{800}$ ns

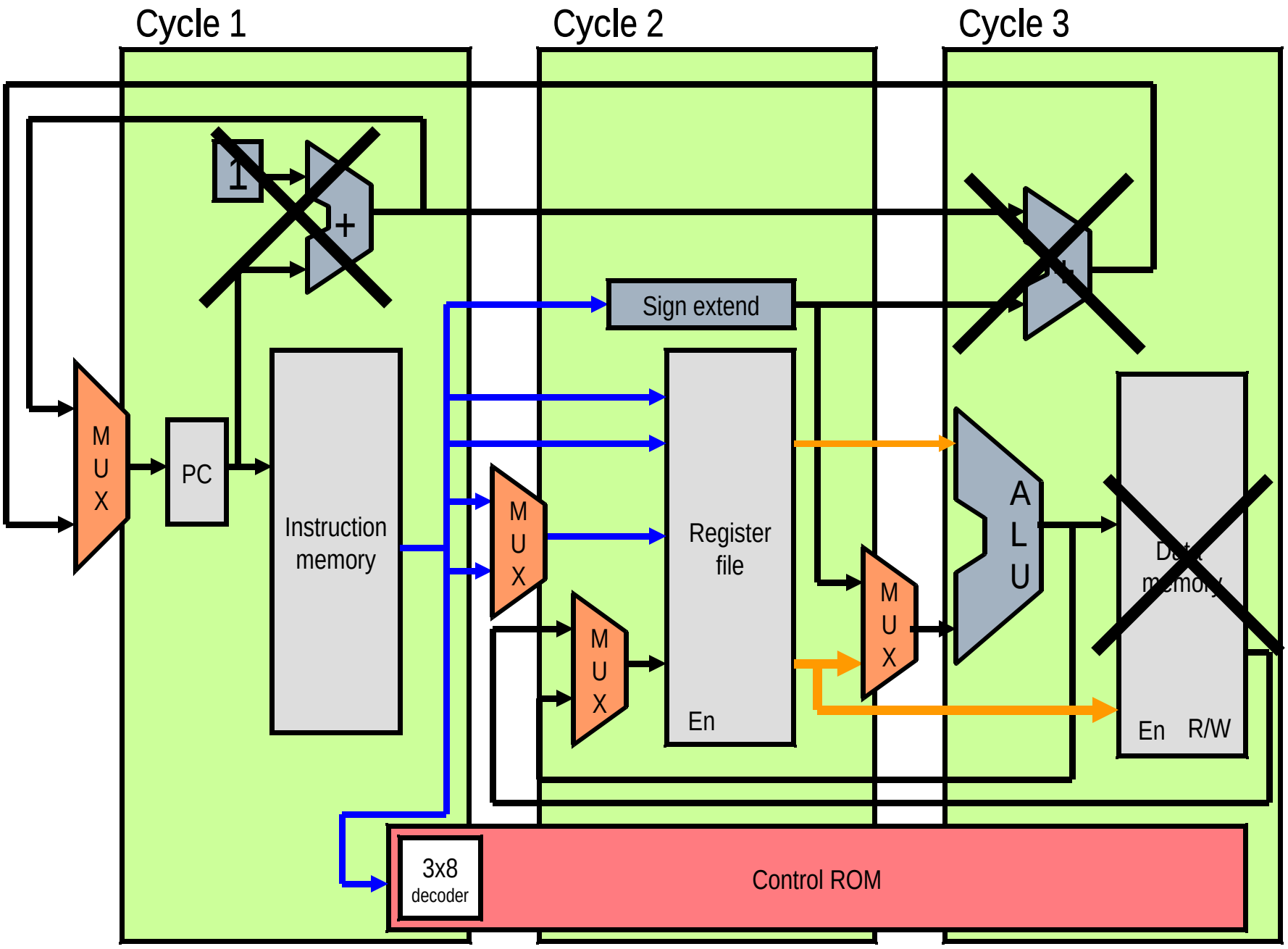
Optimal execution:

- $25 \cdot 8ns + 10 \cdot 7ns + 45 \cdot 6ns + 20 \cdot 5ns = \underline{640}$ ns

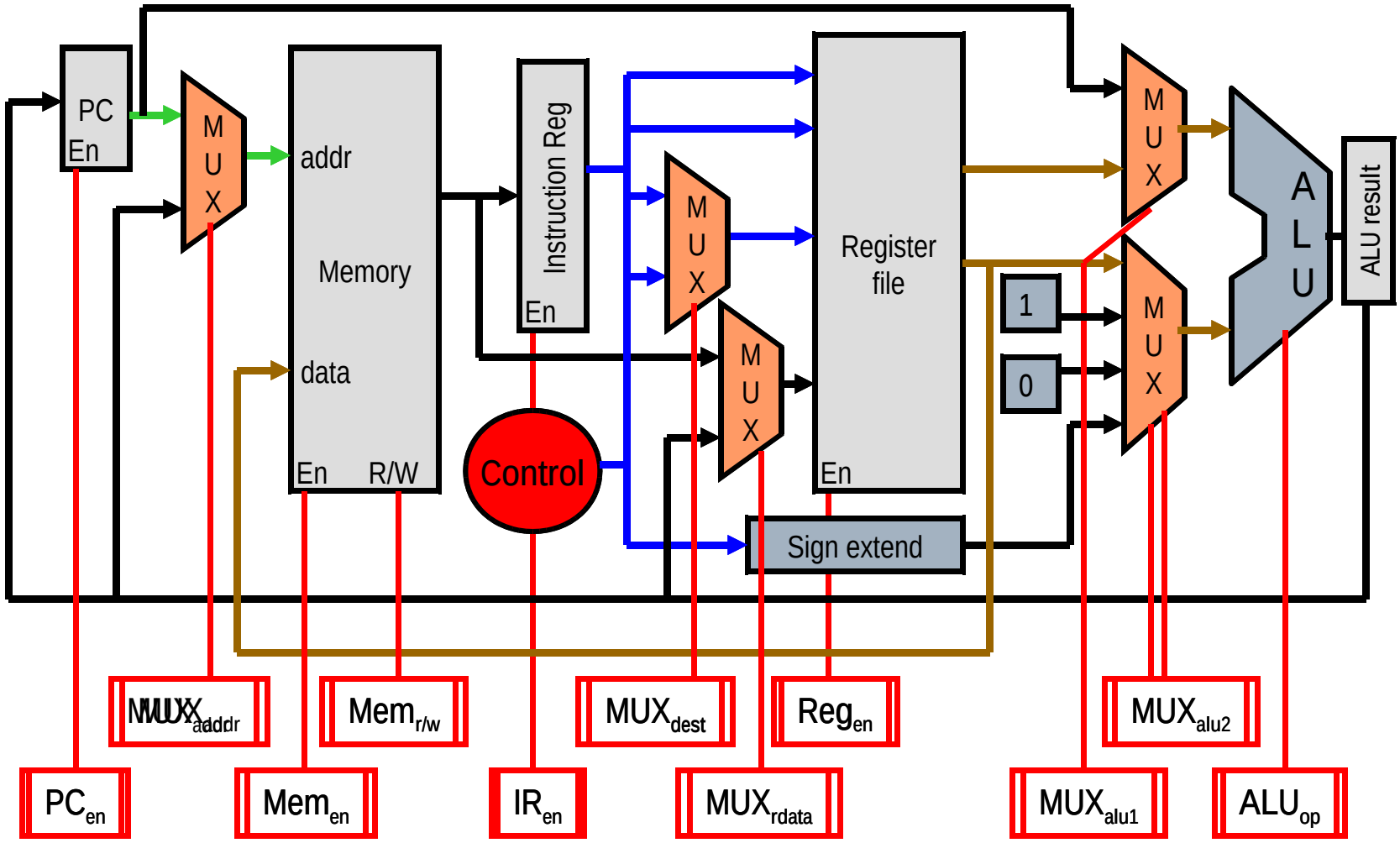
In reality, overhead to support multicycle.

Multiple-cycle execution

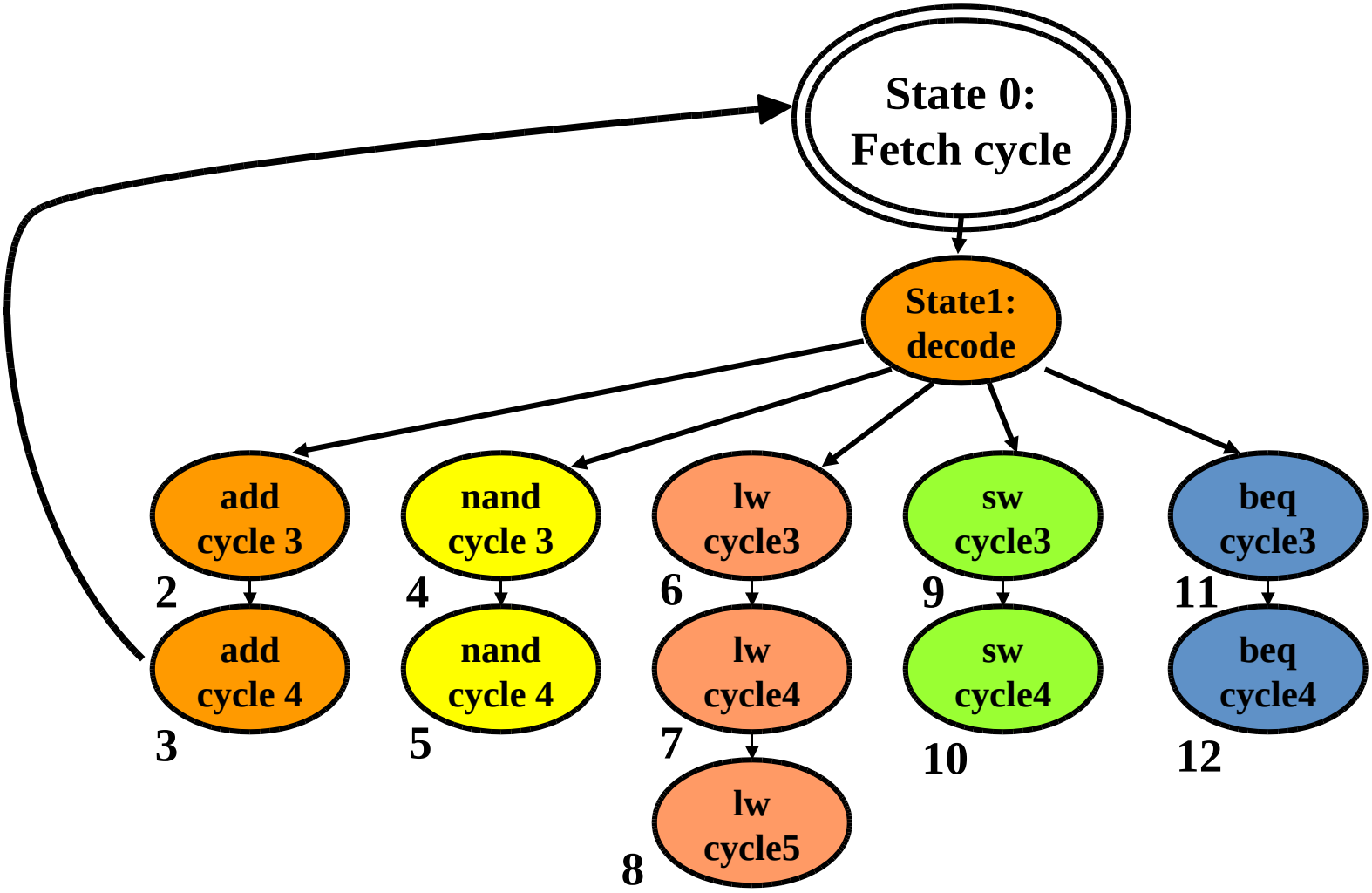
- Each instruction takes multiple cycles to execute
  - Cycle time is reduced
  - Slower instructions take more cycles
  - Can reuse datapath elements each cycle
- What is needed to make this work?
  - Since you are re-using elements for different purposes, you need more and/or wider MUXes.
  - You may 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
Instruction
memory
M
U
X
Register
file
En
M
U
X
M
U
X
A
L
U
En
R/W
3x8
decoder
Control ROM

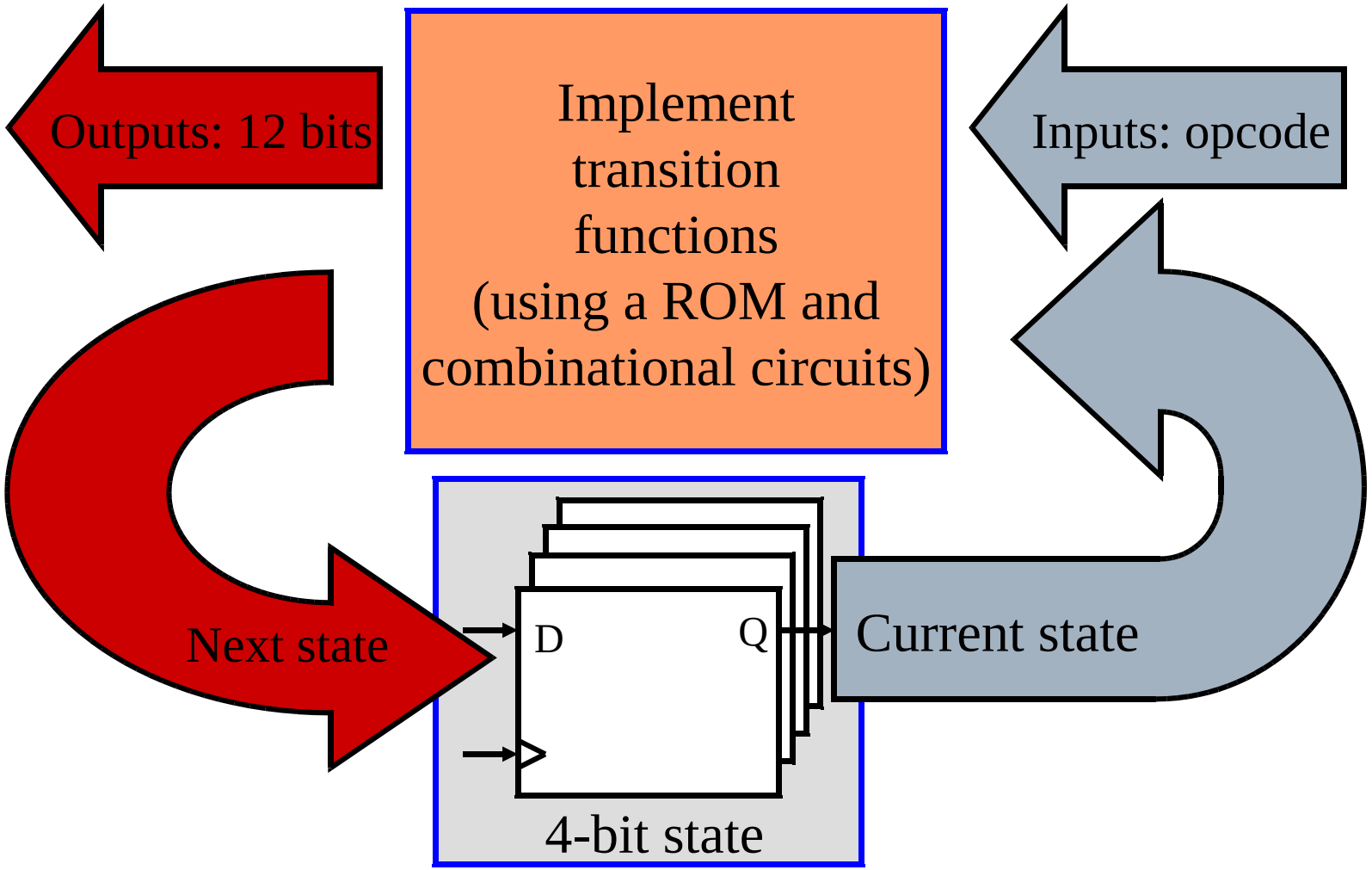
Multicycle LC2Kx datapath



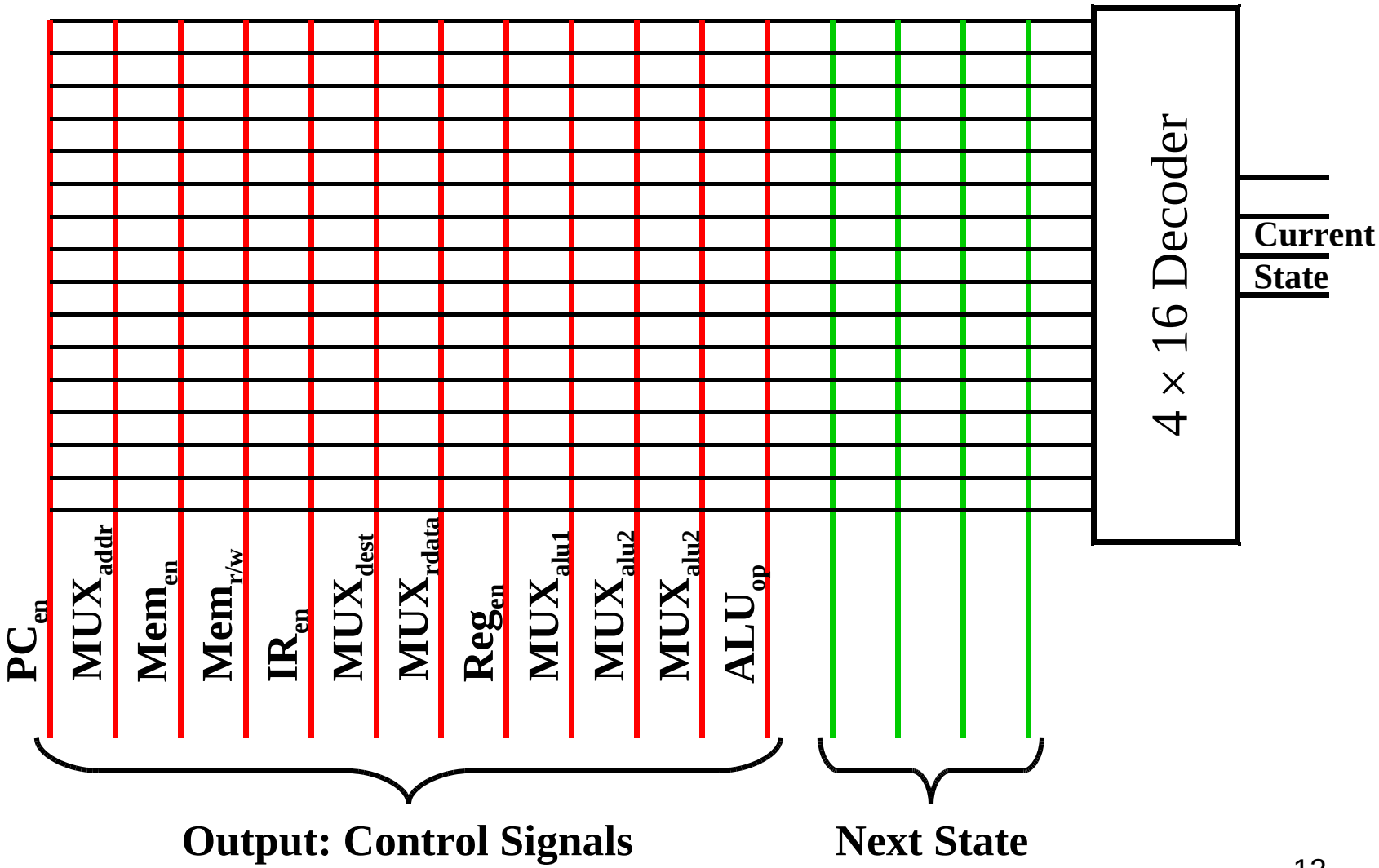
State machine for multi-cycle control signals (transition functions)



FSM design


Outputs: 12 bits
Implement transition functions (using a ROM and combinational circuits)
Inputs: opcode
Next state
D
Q
Current state
4-bit state

Setting control ROM contents

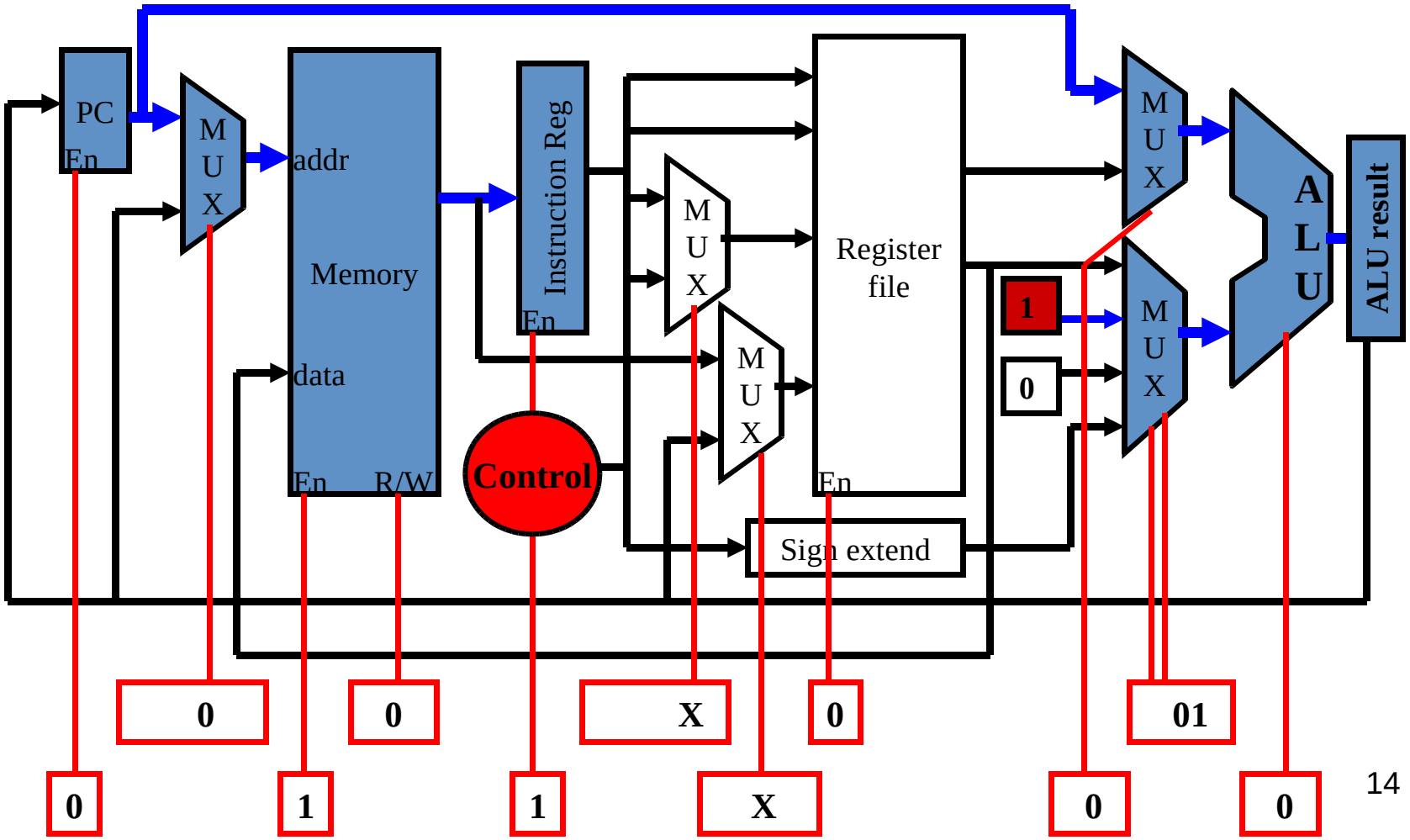


First cycle / state 0: fetch instruction

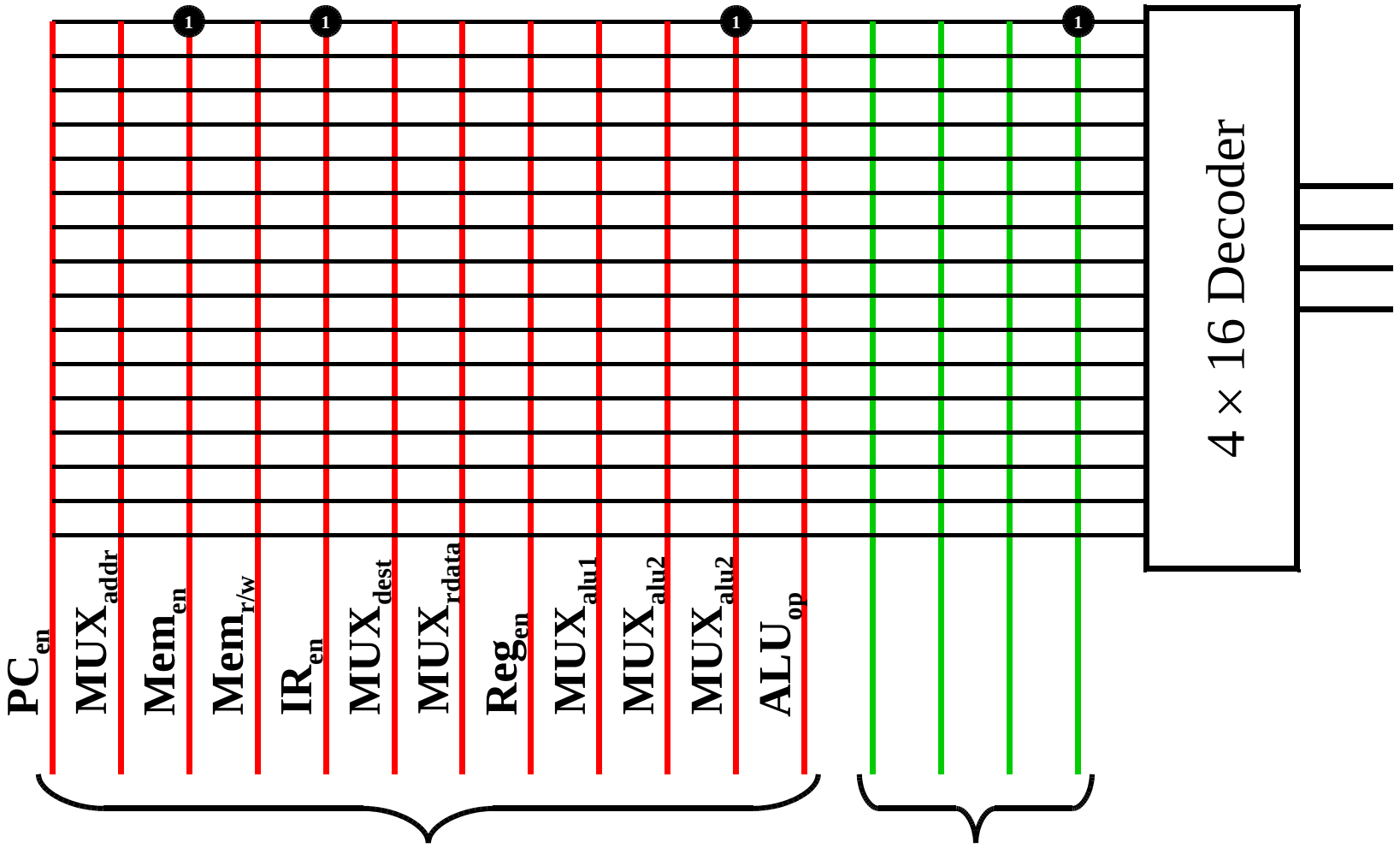
- What operations need to be done in the first cycle of executing any instruction?
  - Read memory[PC] and store into instruction register.
    - Must select PC in memory address MUX ($MUX_{addr} = 0$)
    - Enable memory operation ($Mem_{en} = 1$)
    - R/W should be (read) ($Mem_{r/w} = 0$)
    - Enable 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

Cycle 1 operation

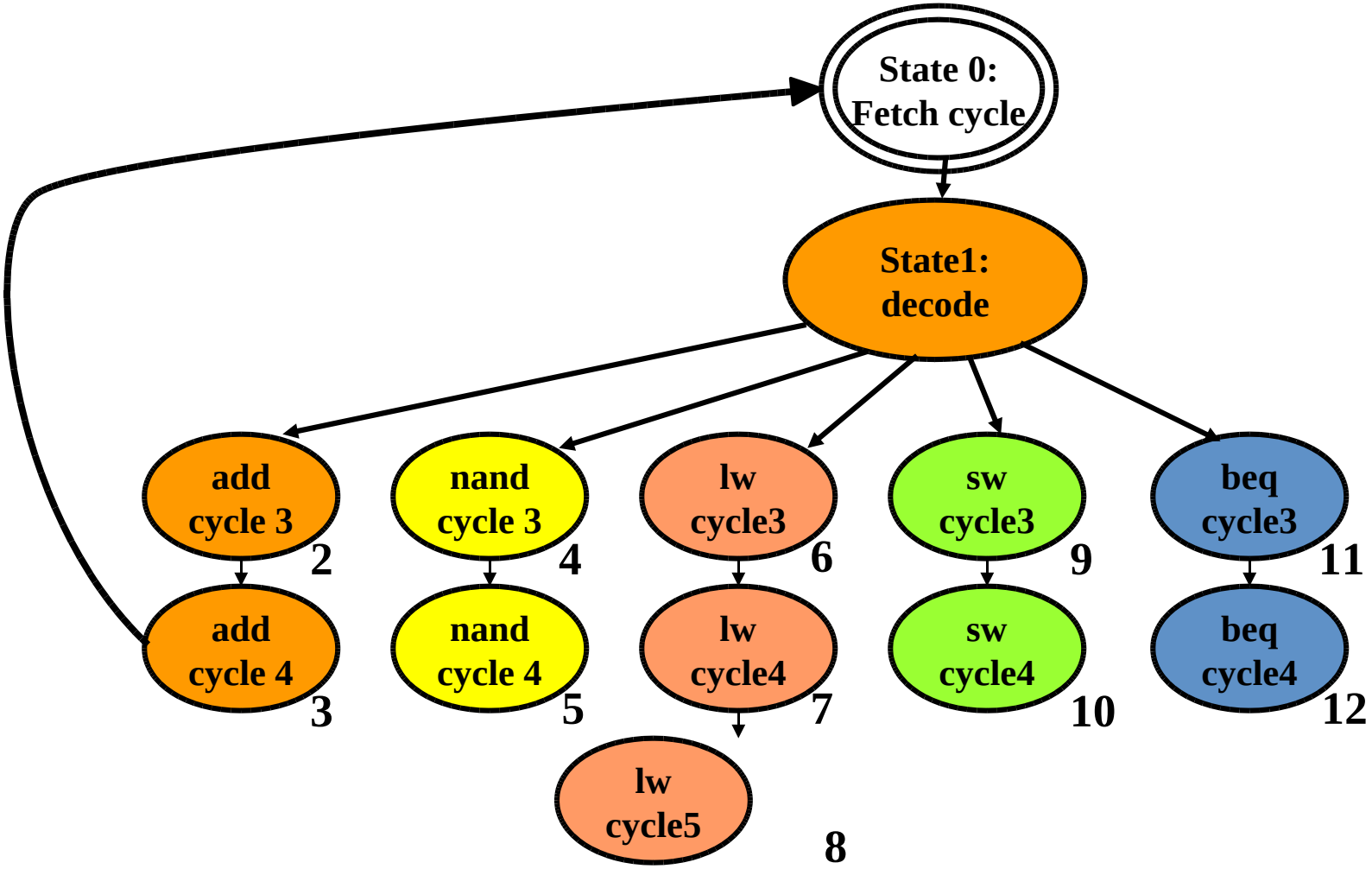
This is the same for all instructions
(since we don't know the instruction yet!)



Setting control ROM contents

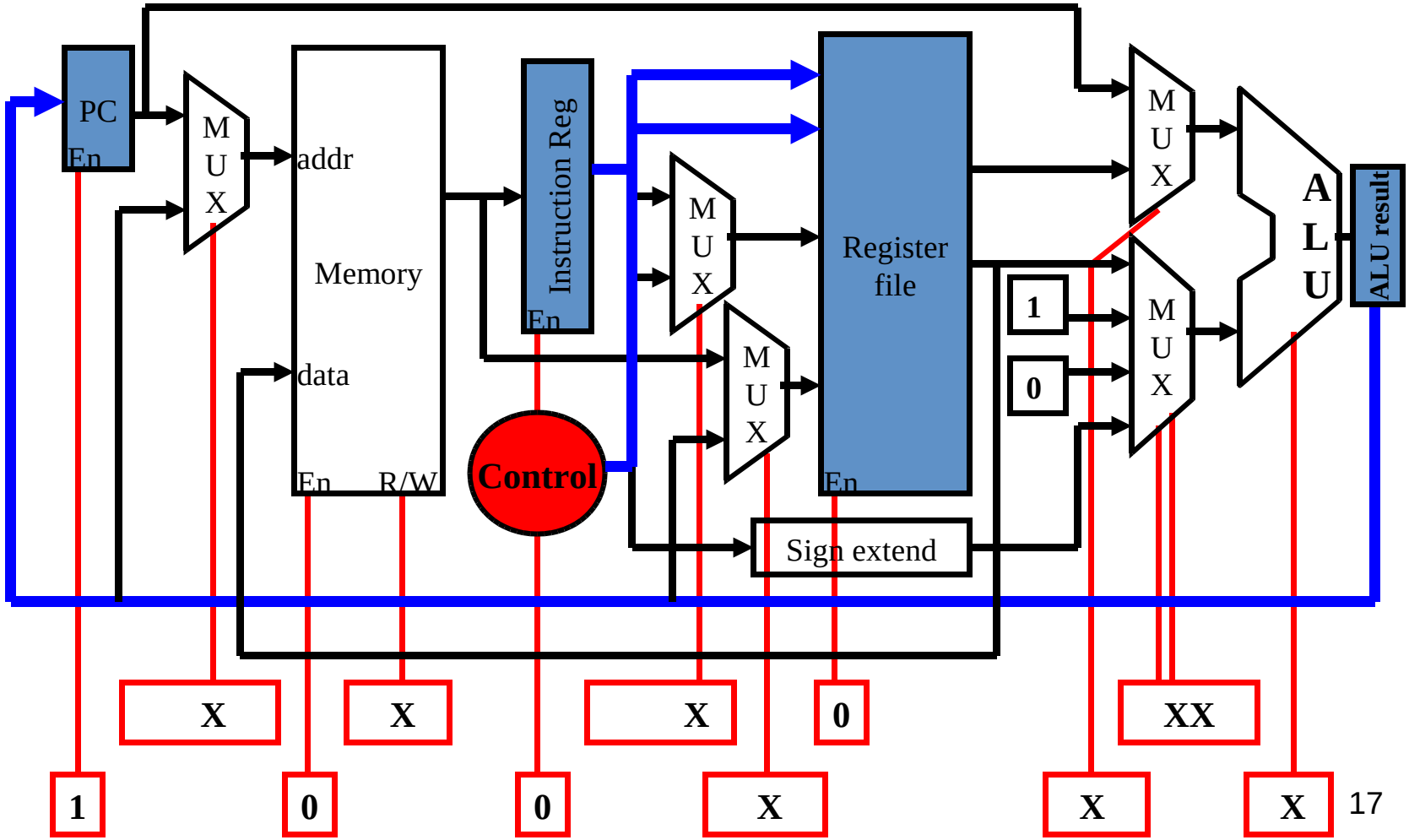


State 1: instruction decode

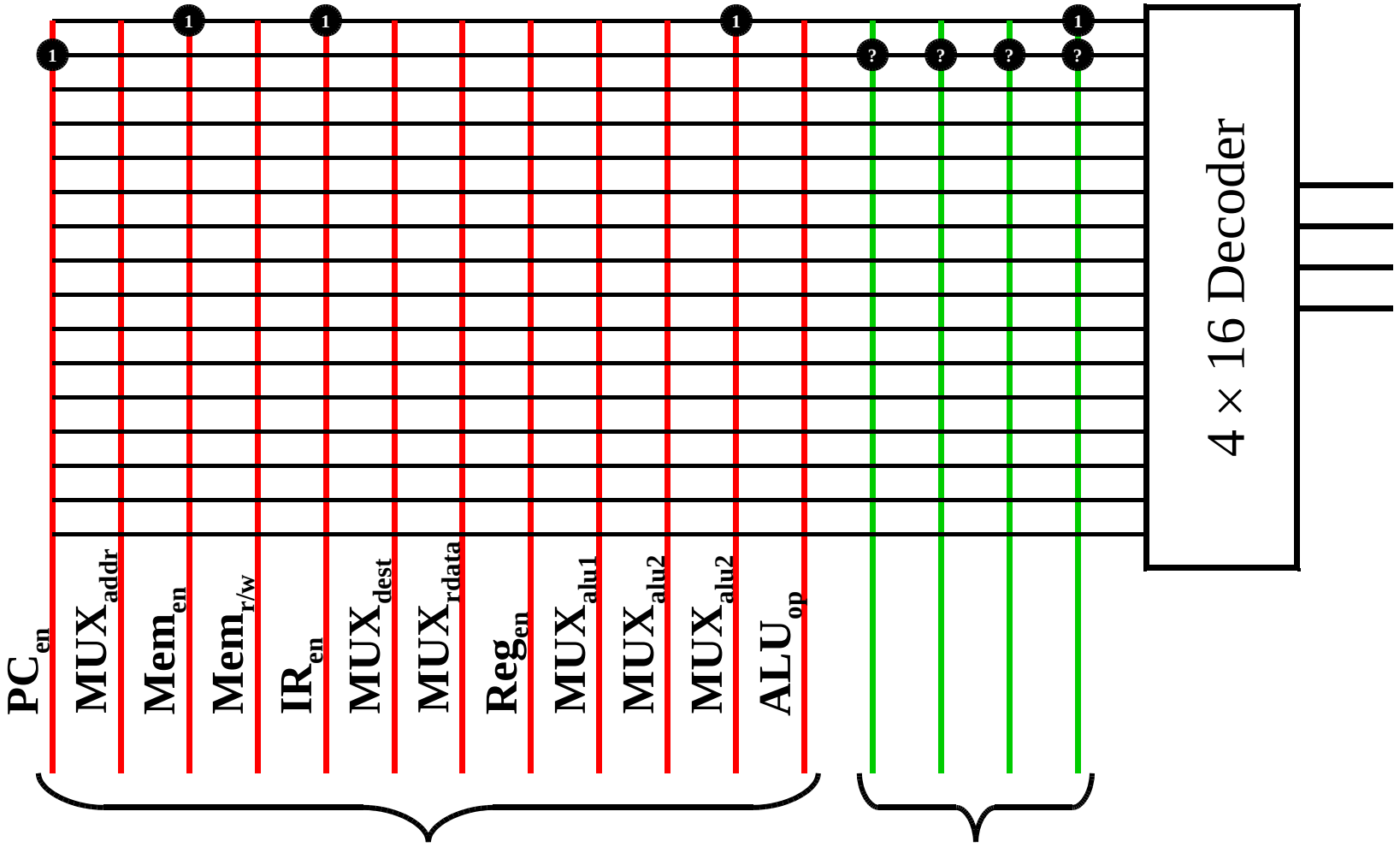


State 1: output function

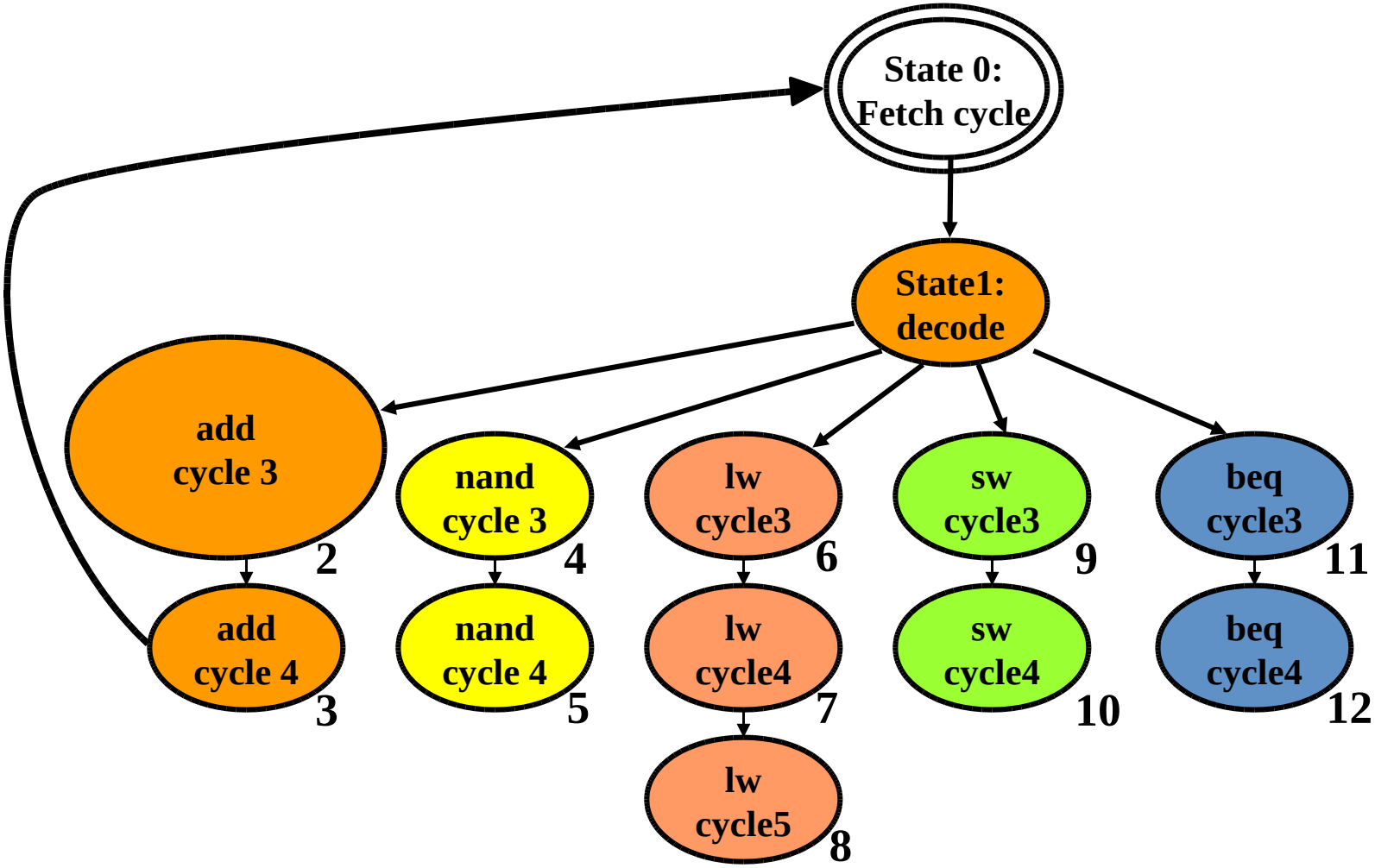
Update PC; read registers (regA and regB);
use opcode to determine next state



Setting control ROM contents

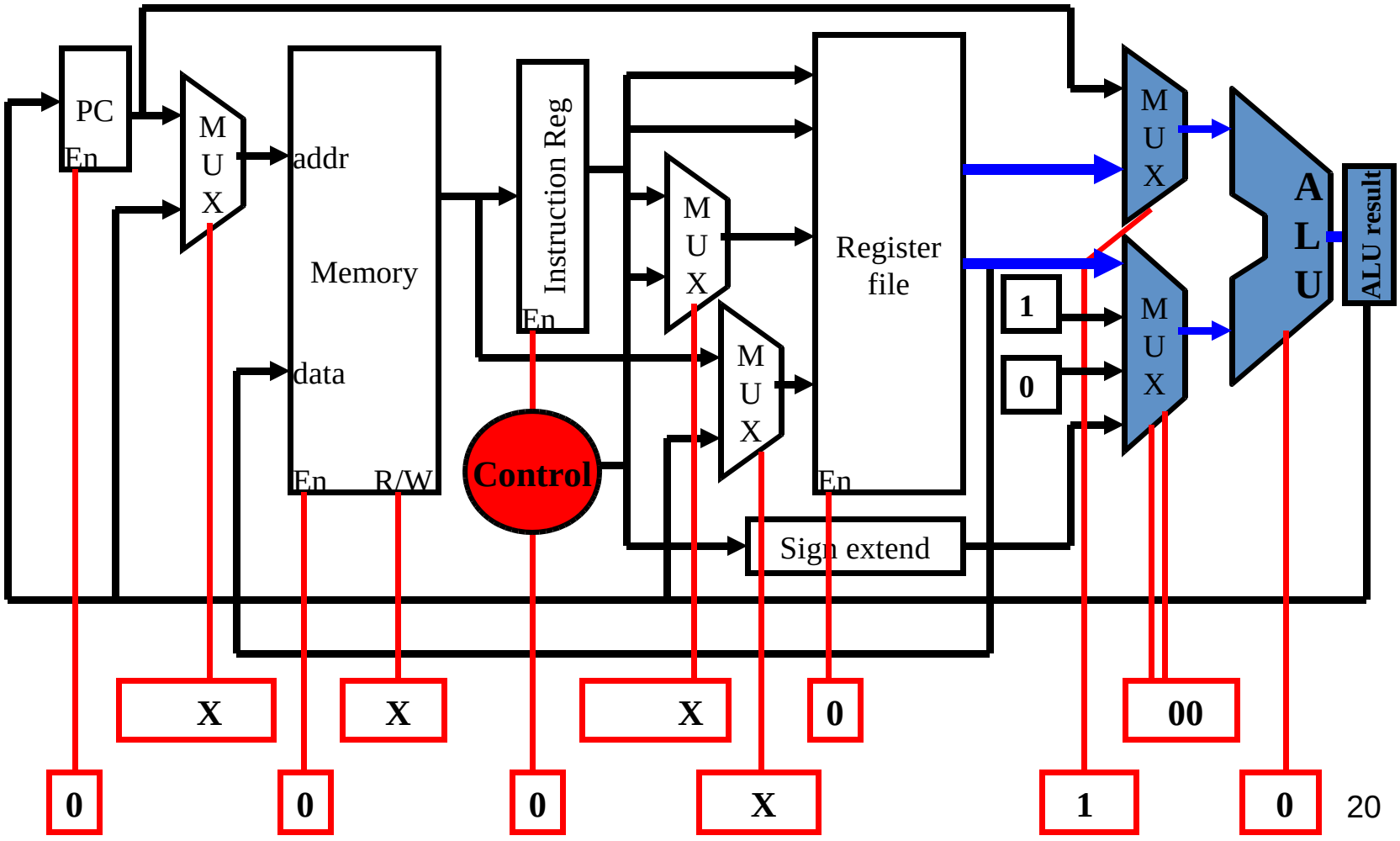


State 2: add cycle 3

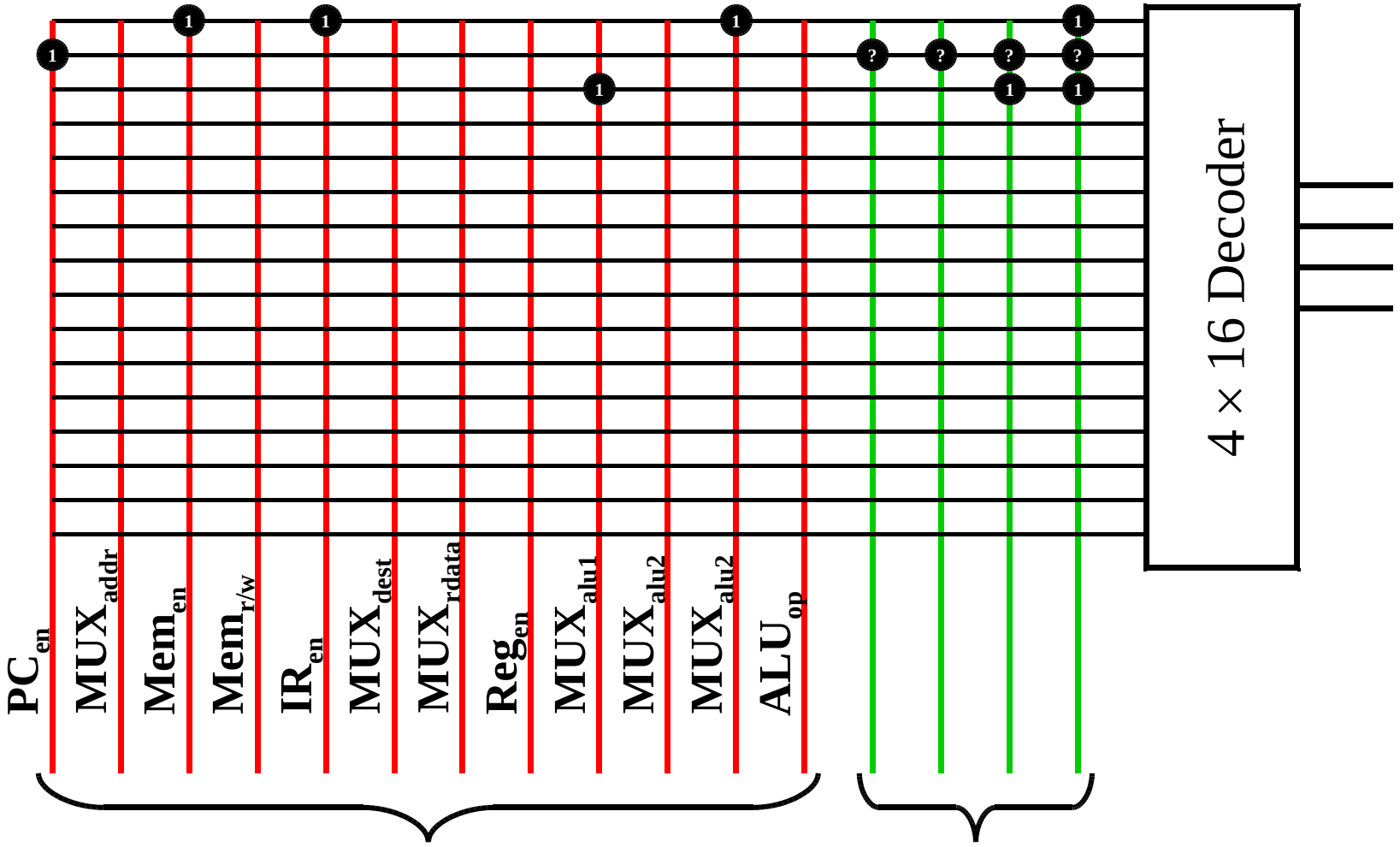


State 2: Add cycle 3 operation

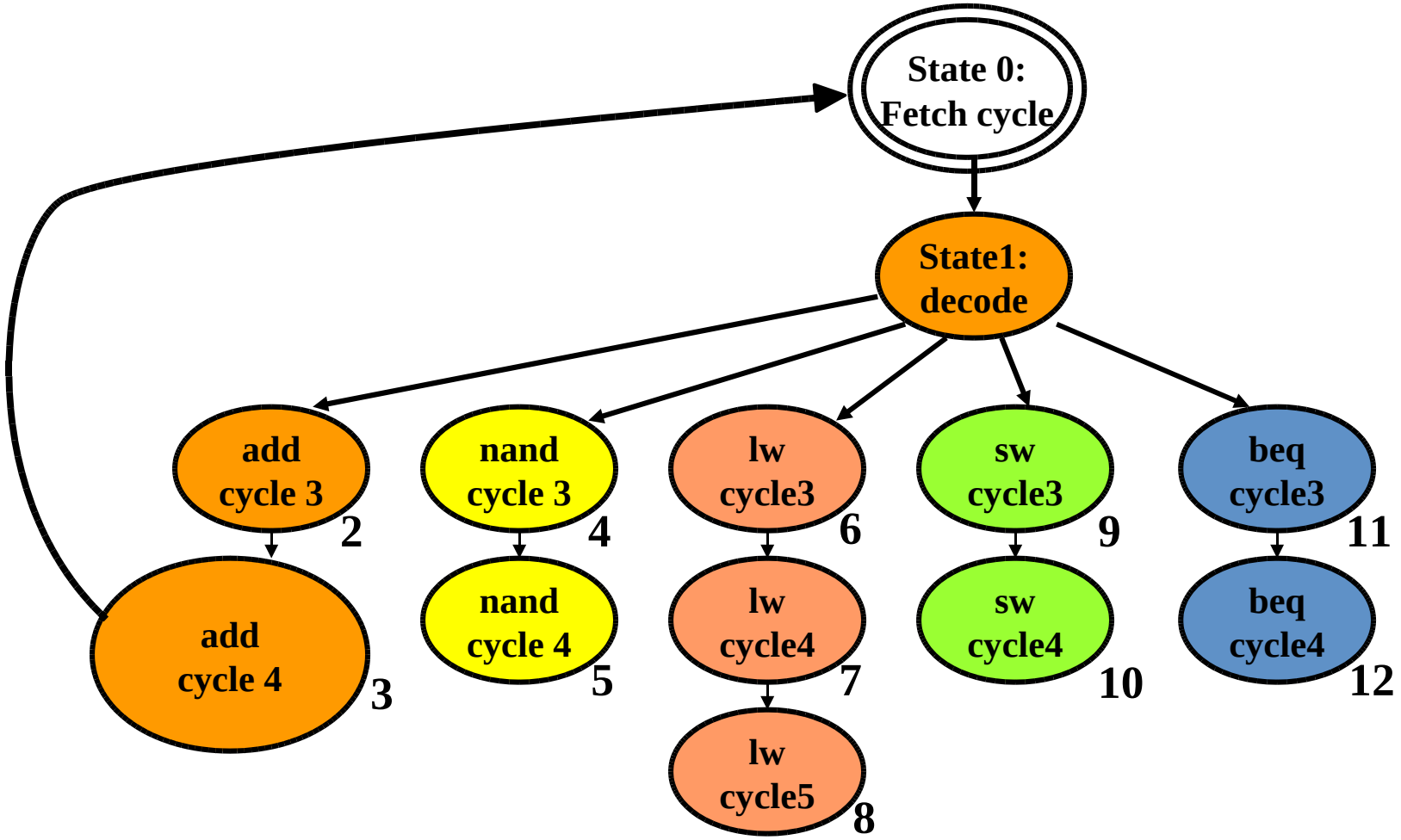
Send control signals to MUX to select values of regA and regB and control signal to ALU to add



Setting control ROM contents

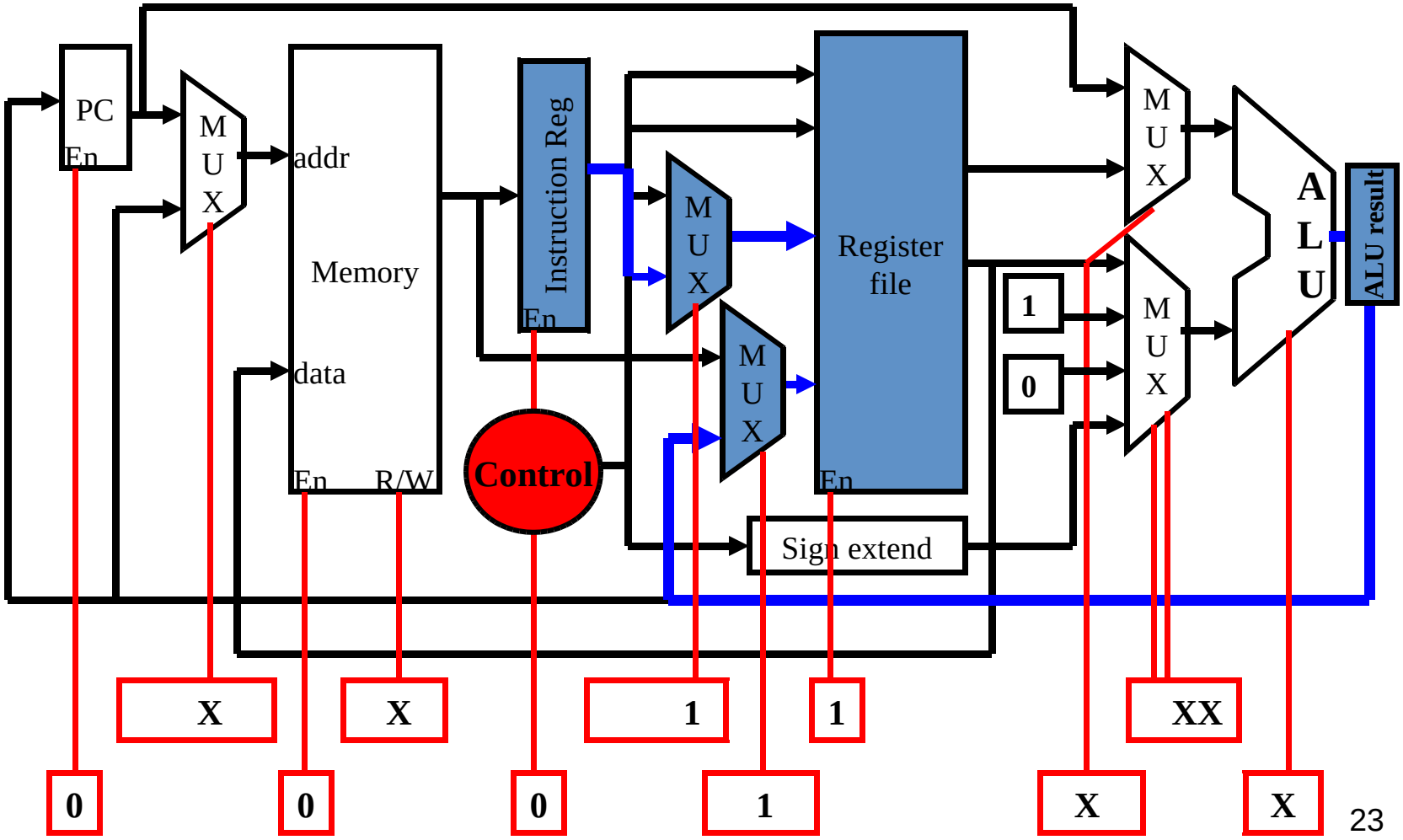


State 3: add cycle 4

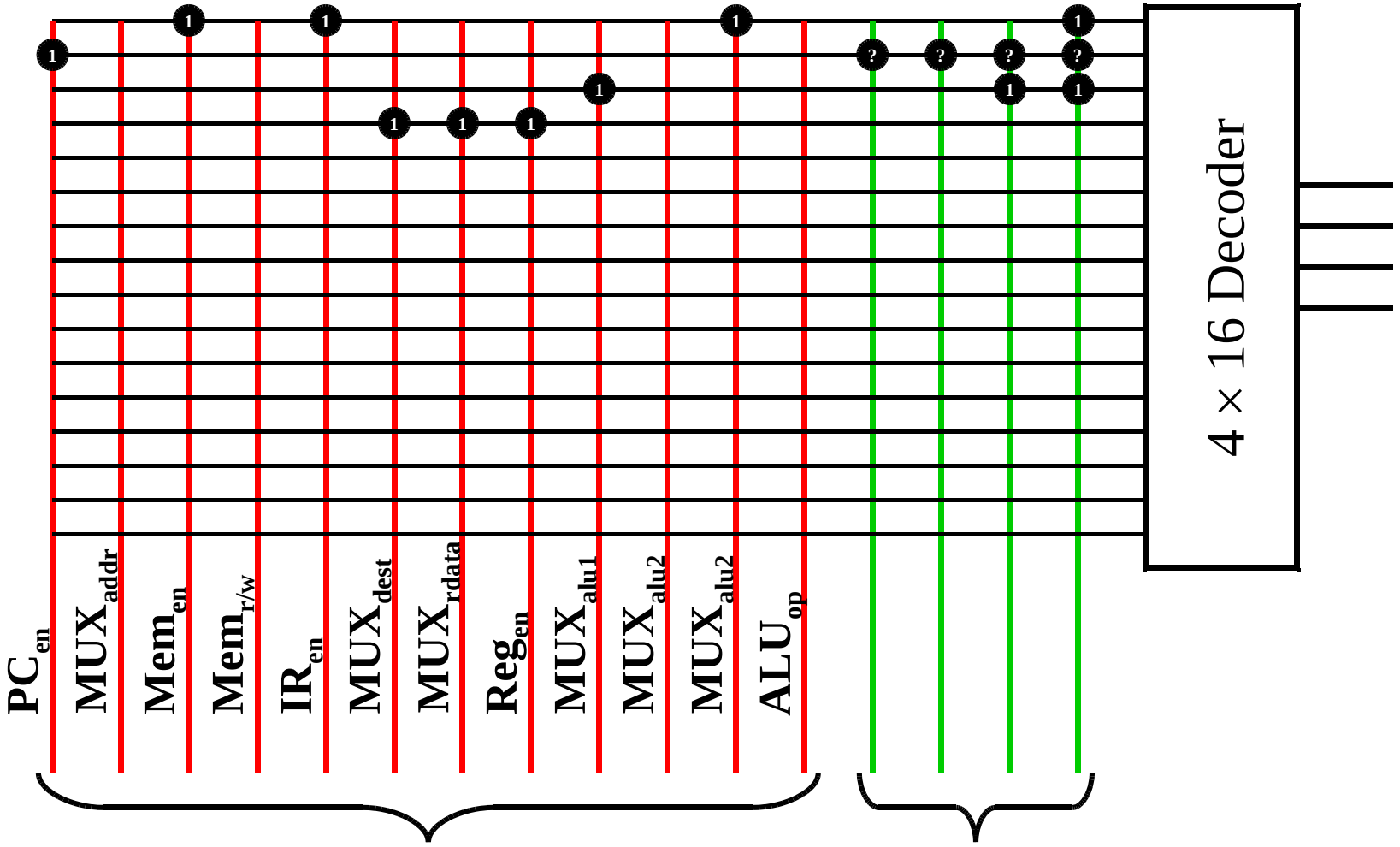


Add cycle 4 operation

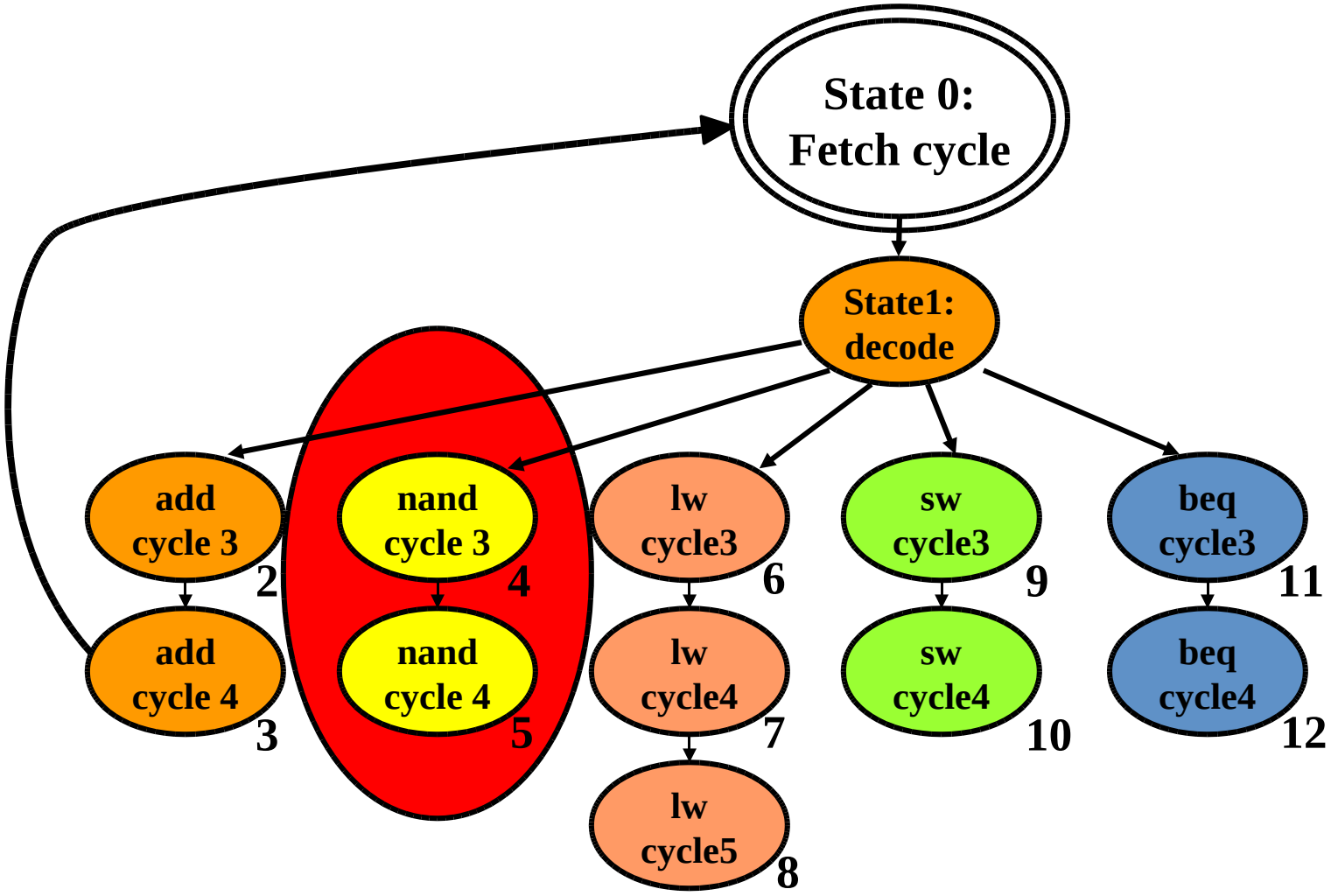
Send control signal to address MUX to select dest and to data MUX to select ALU output, then send write enable to register file.



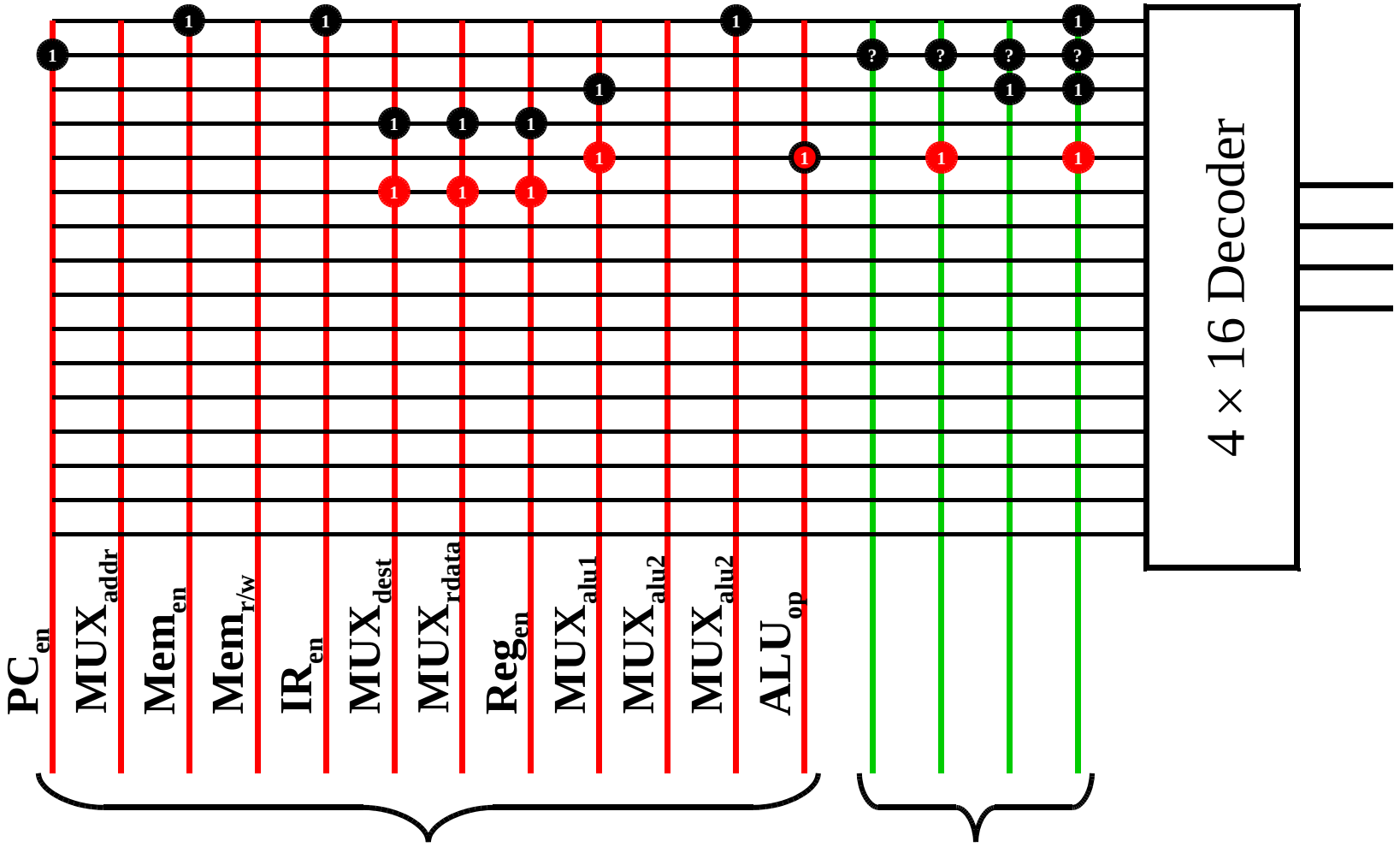
Setting control ROM contents



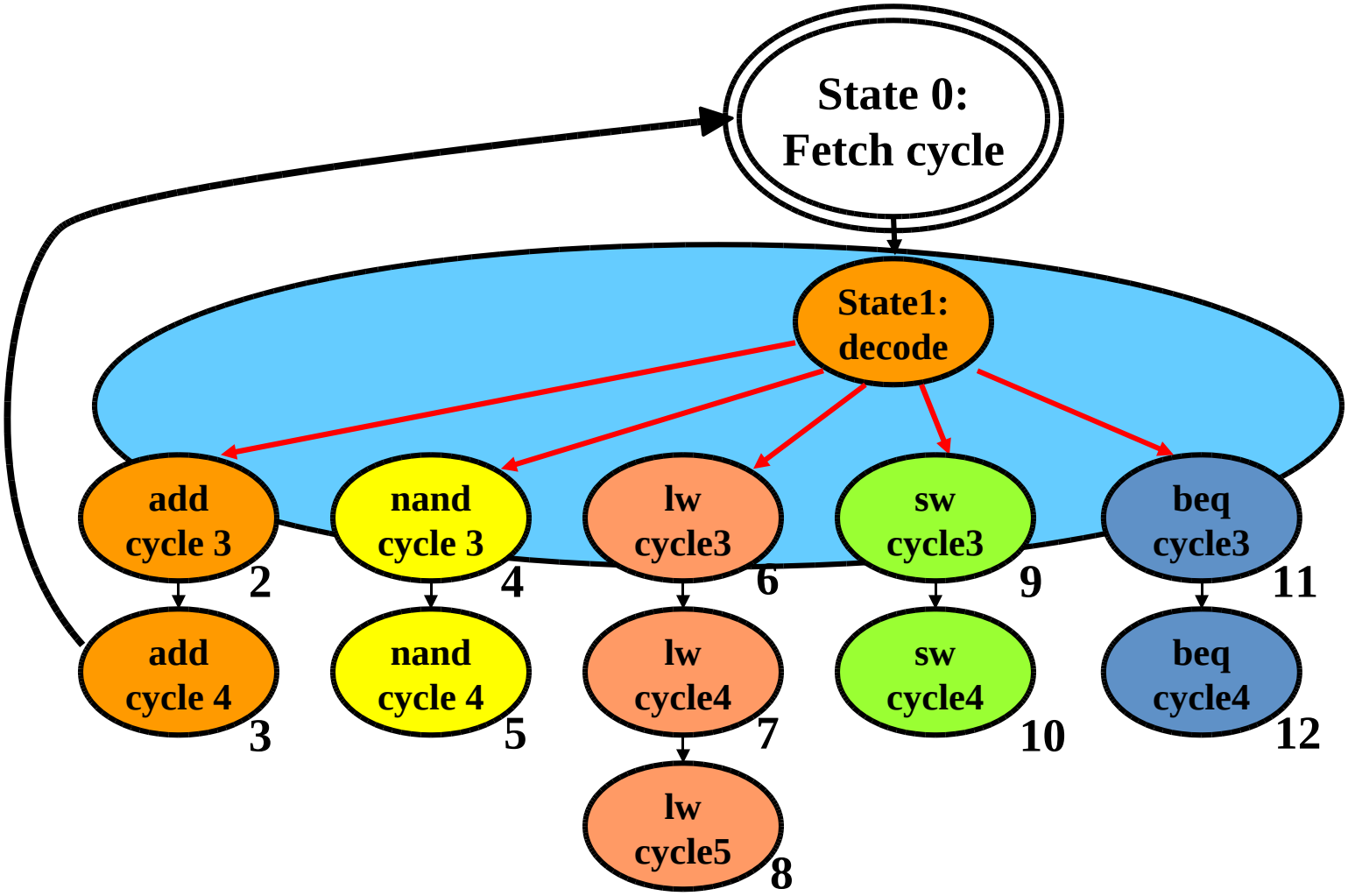
Return to State 0: fetch cycle to execute the next instruction



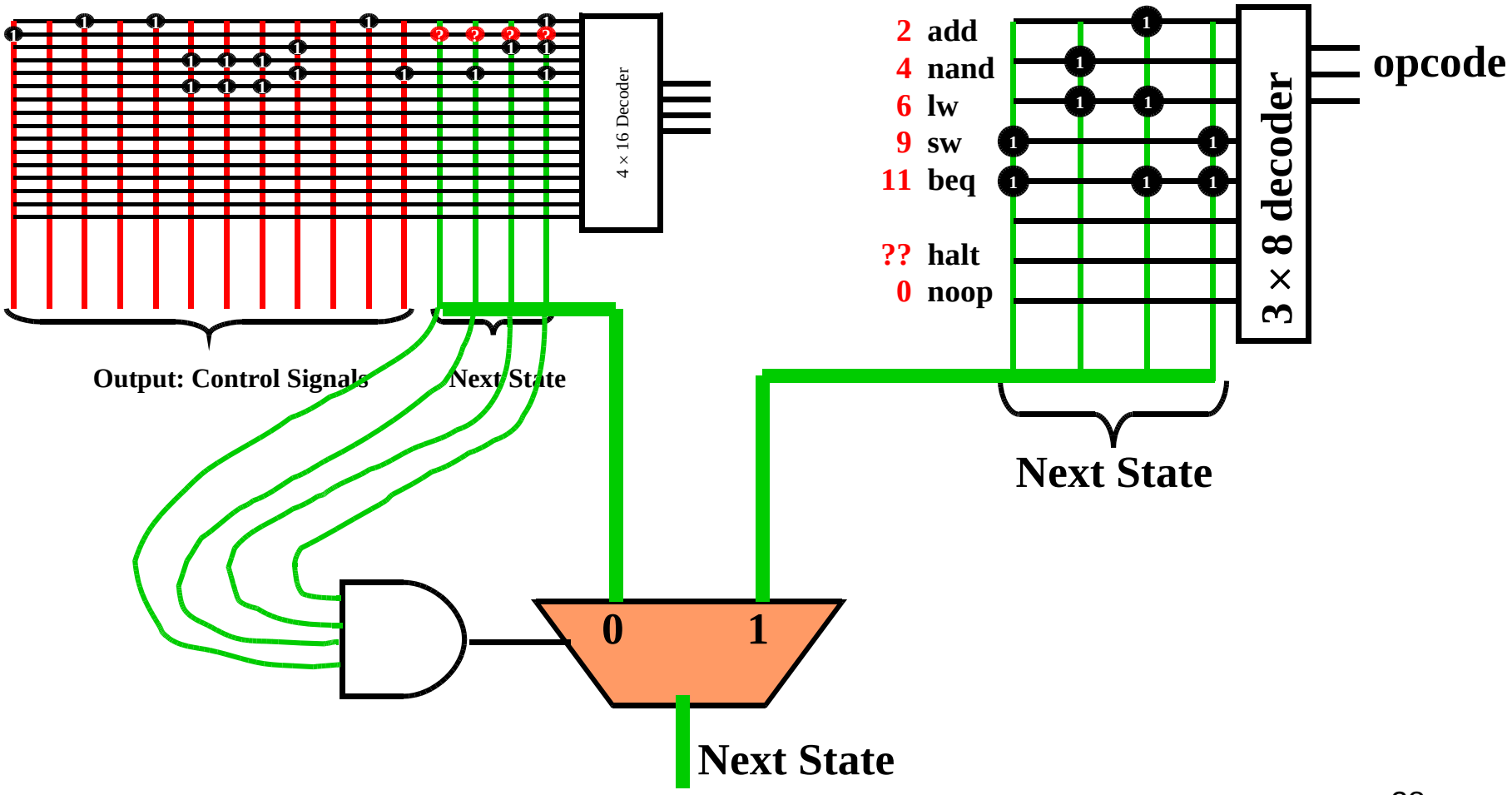
Control ROM settings for nand (4 and 5)



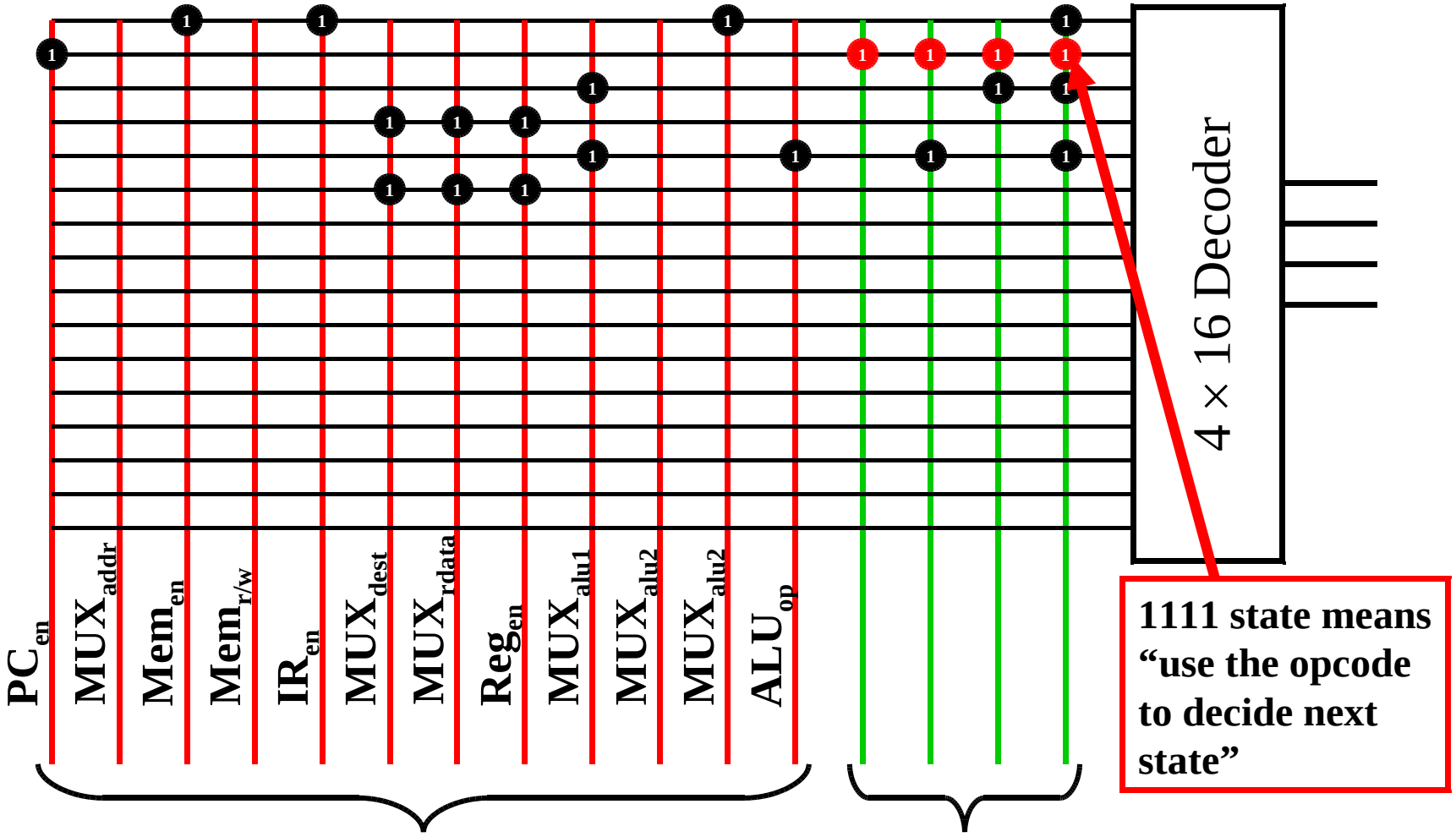
What about the transition from State 1?



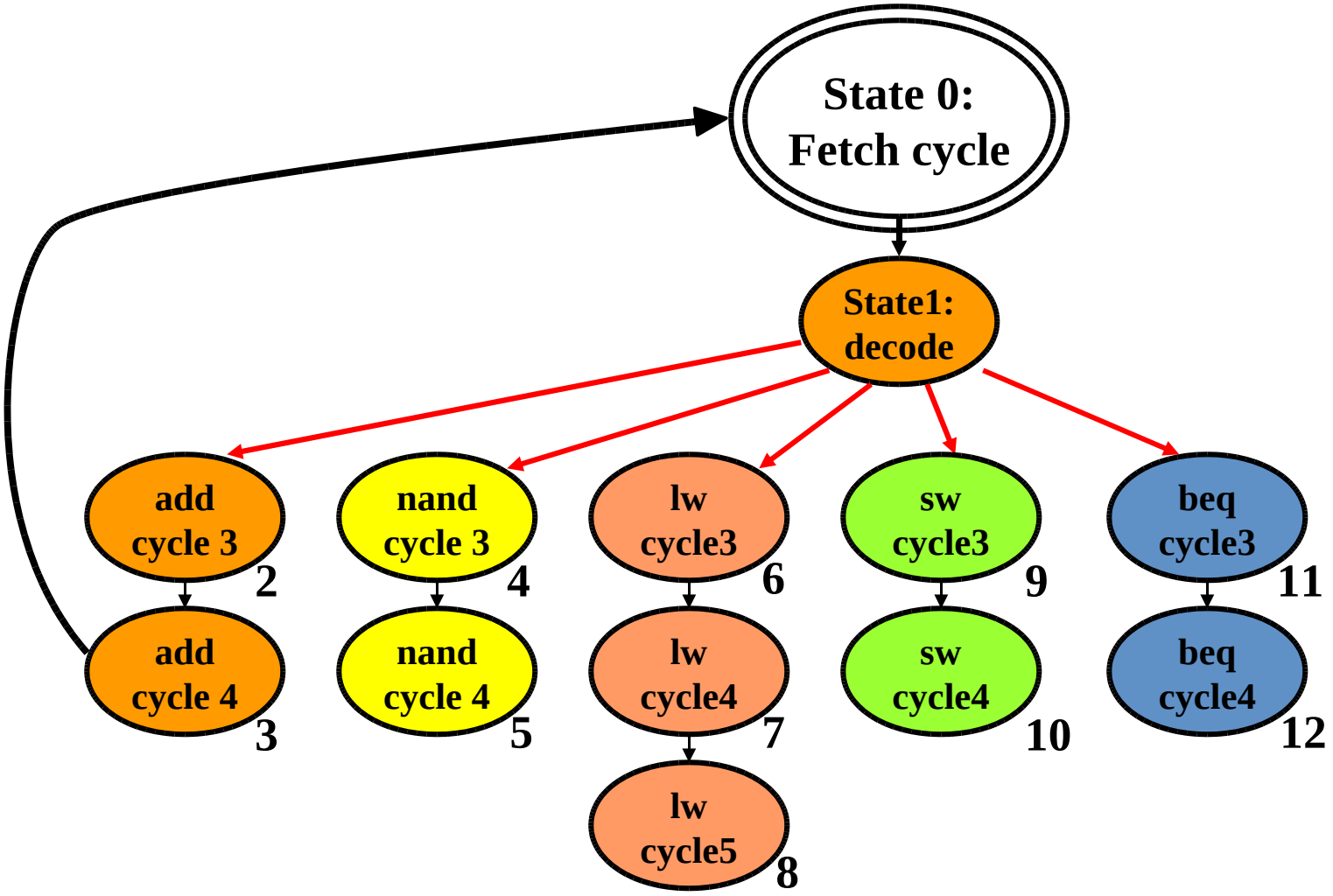
Complete transition function circuit



Control ROM (use of 1111 state)

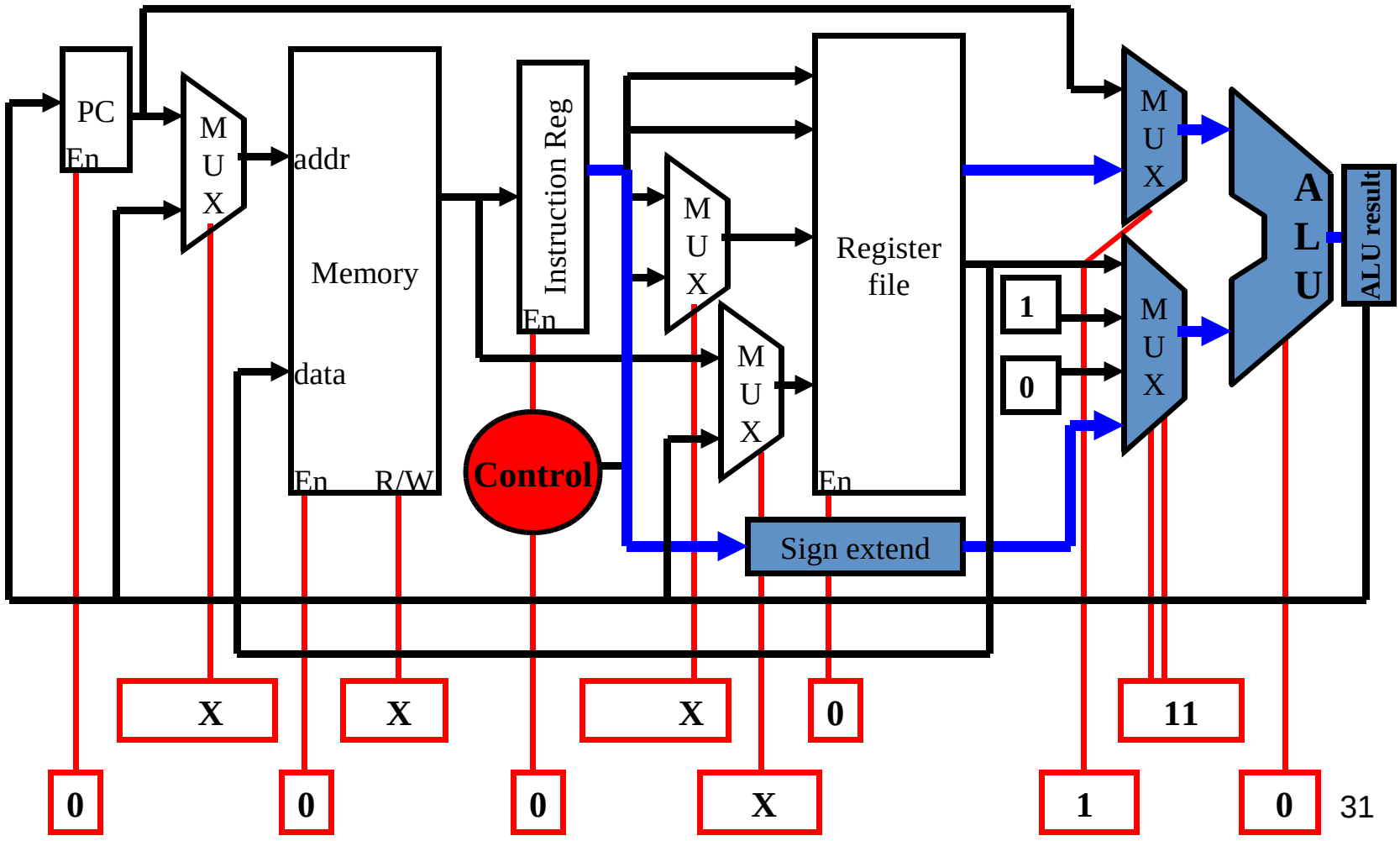


Return to State 0: fetch cycle to execute the next instruction

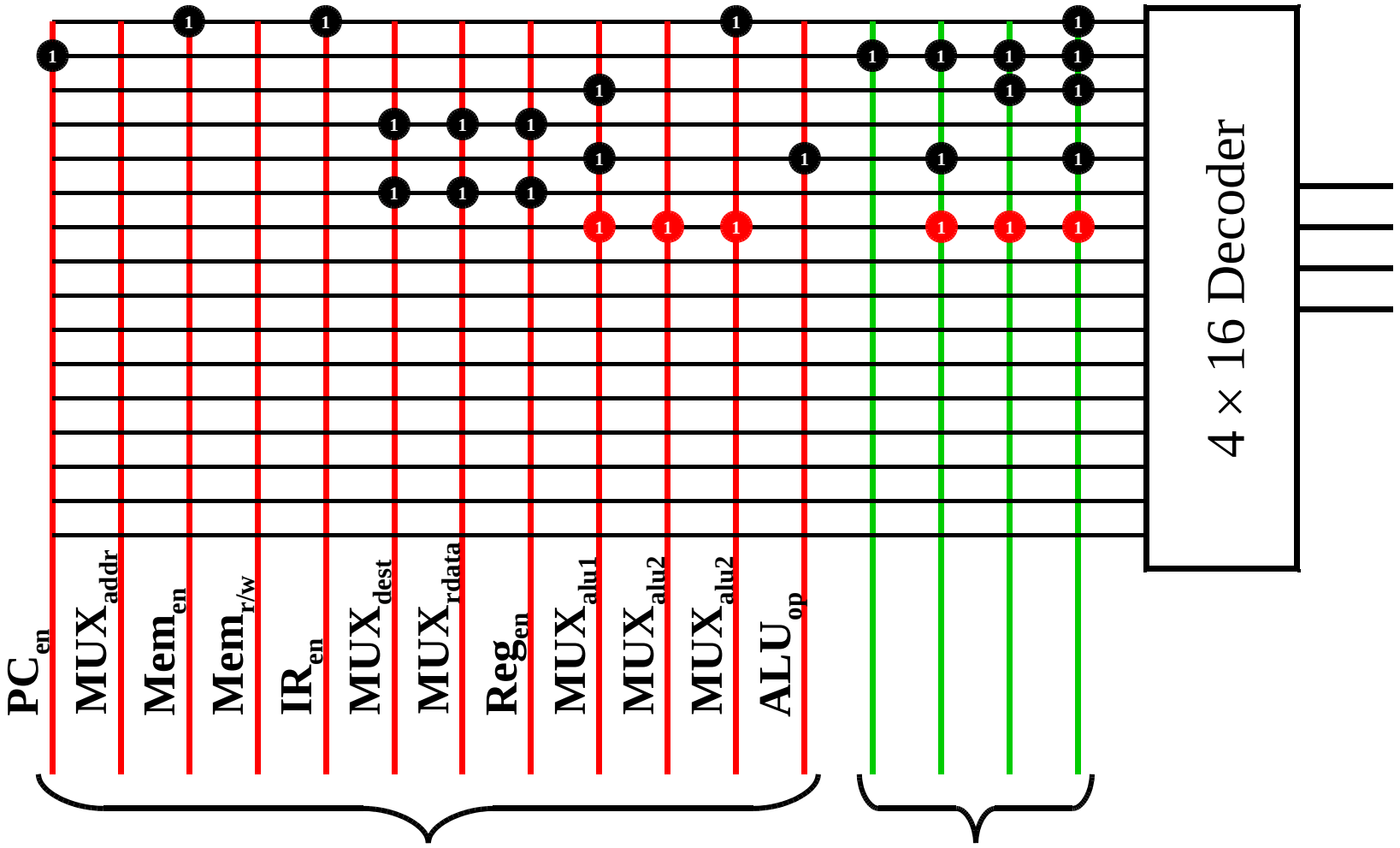


State 6: lw cycle 3

Calculate address for memory reference

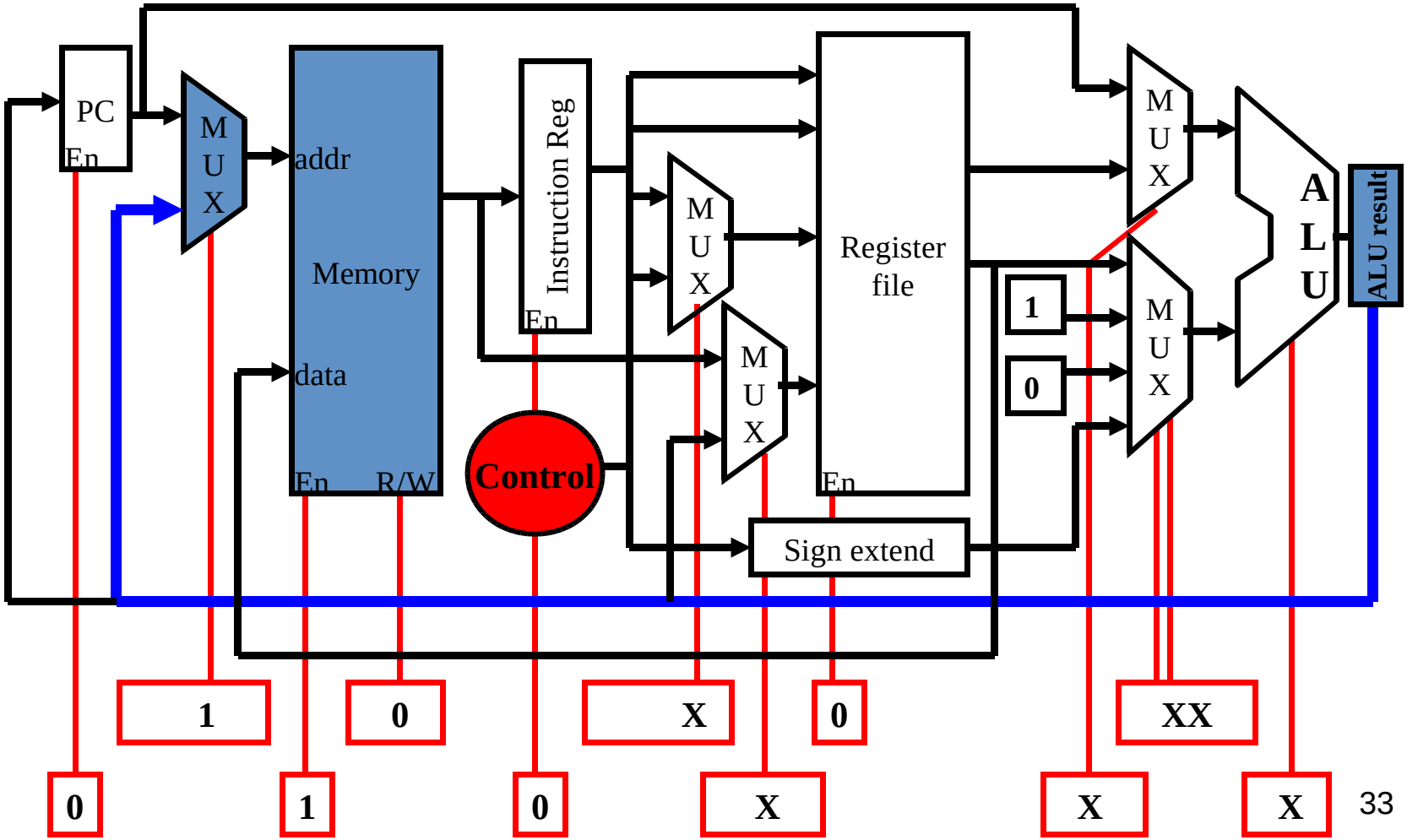


Control ROM settings (lw cycle 3)

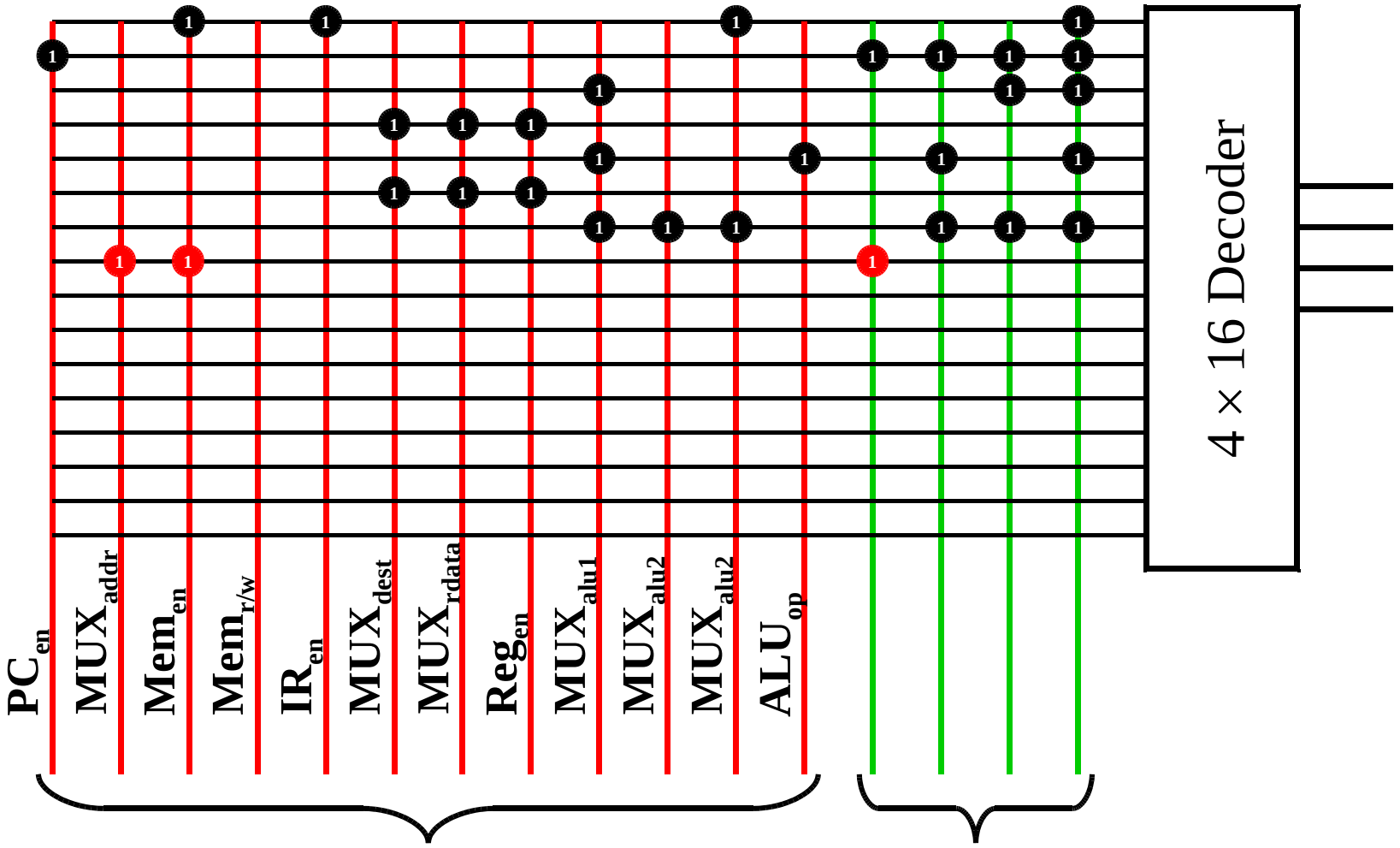


State 7: lw cycle 4

Read memory location

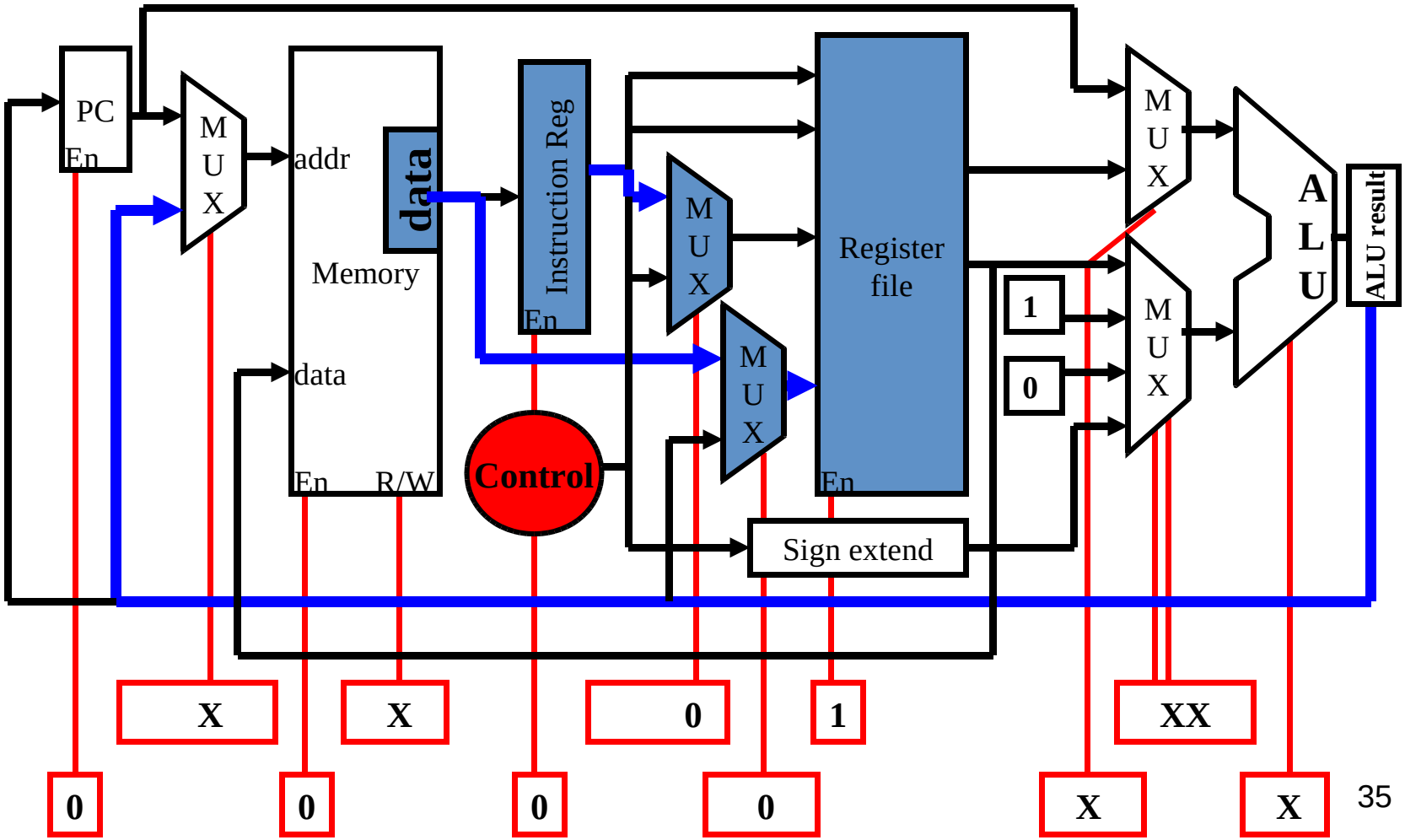


Control ROM settings (lw cycle 4)

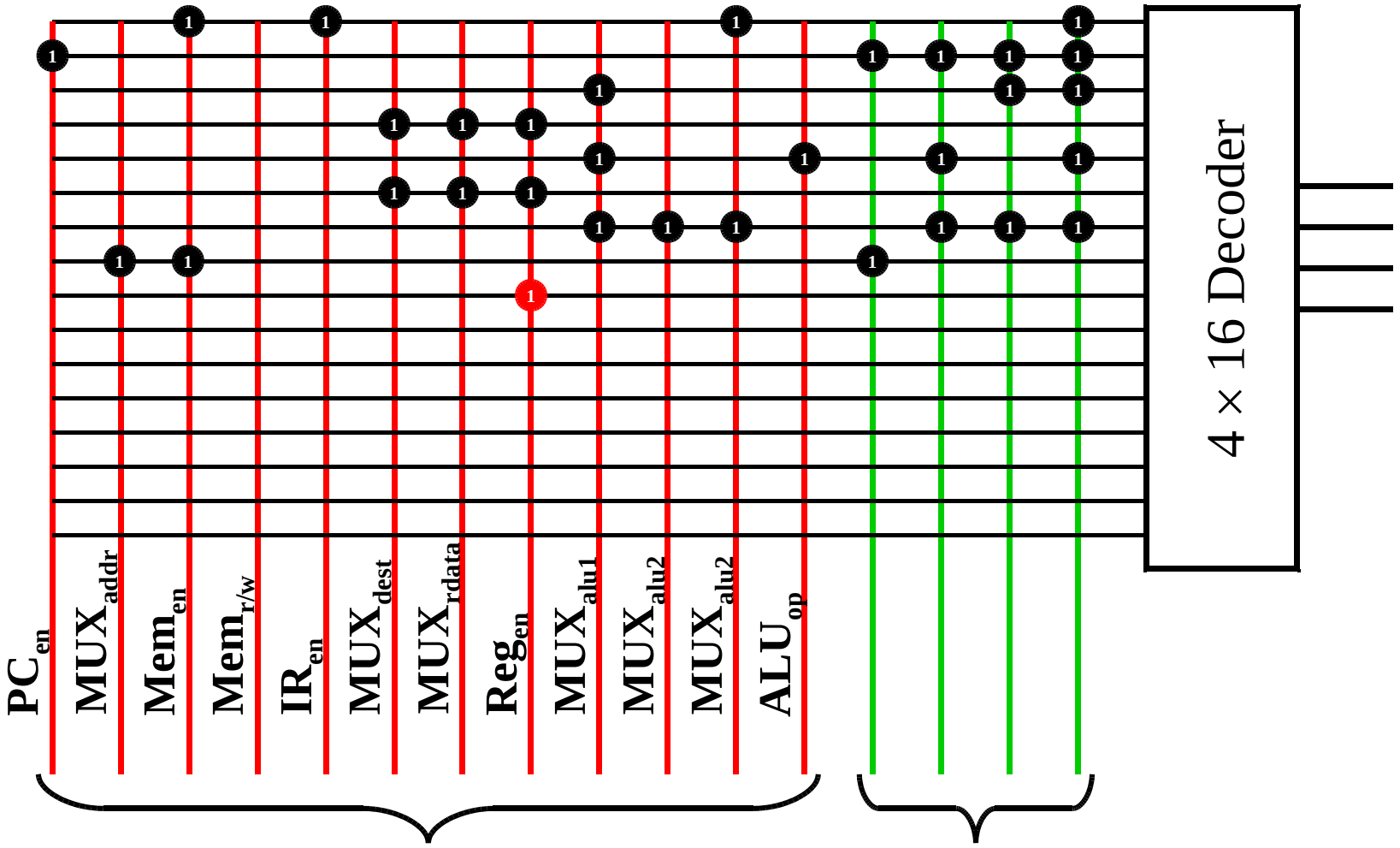


State 8: lw cycle 5

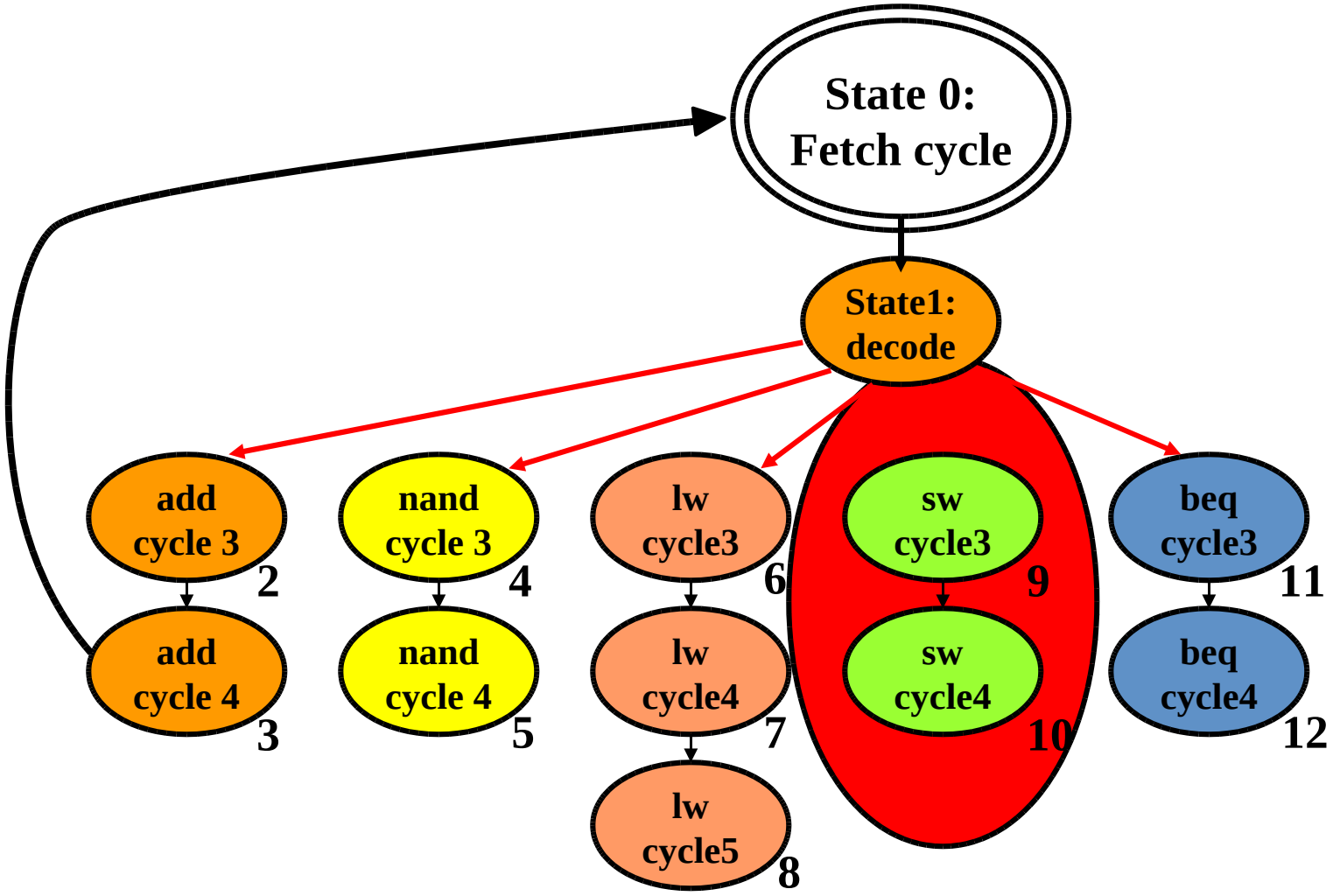
Write memory value to register file



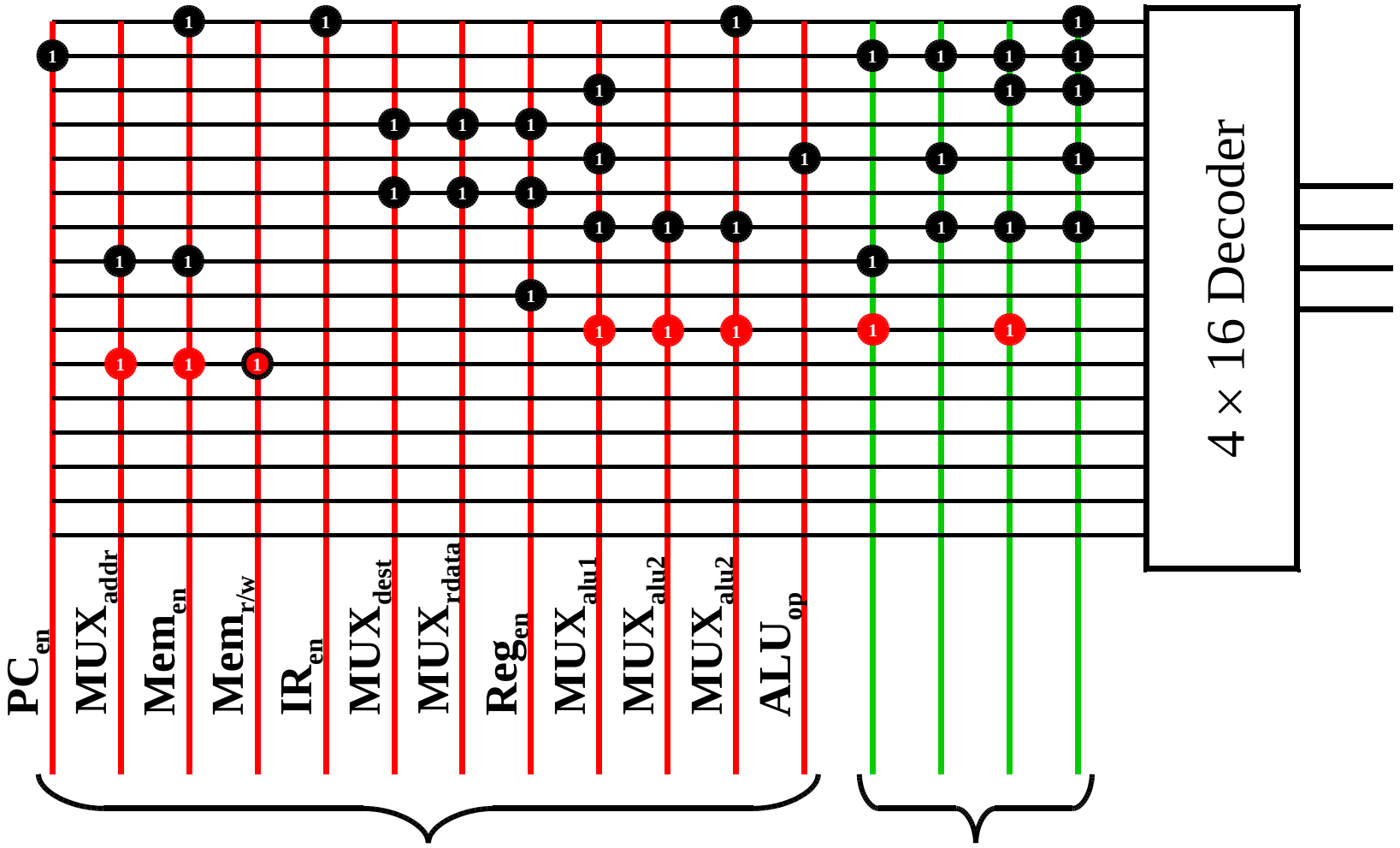
Control ROM settings (lw cycle 5)



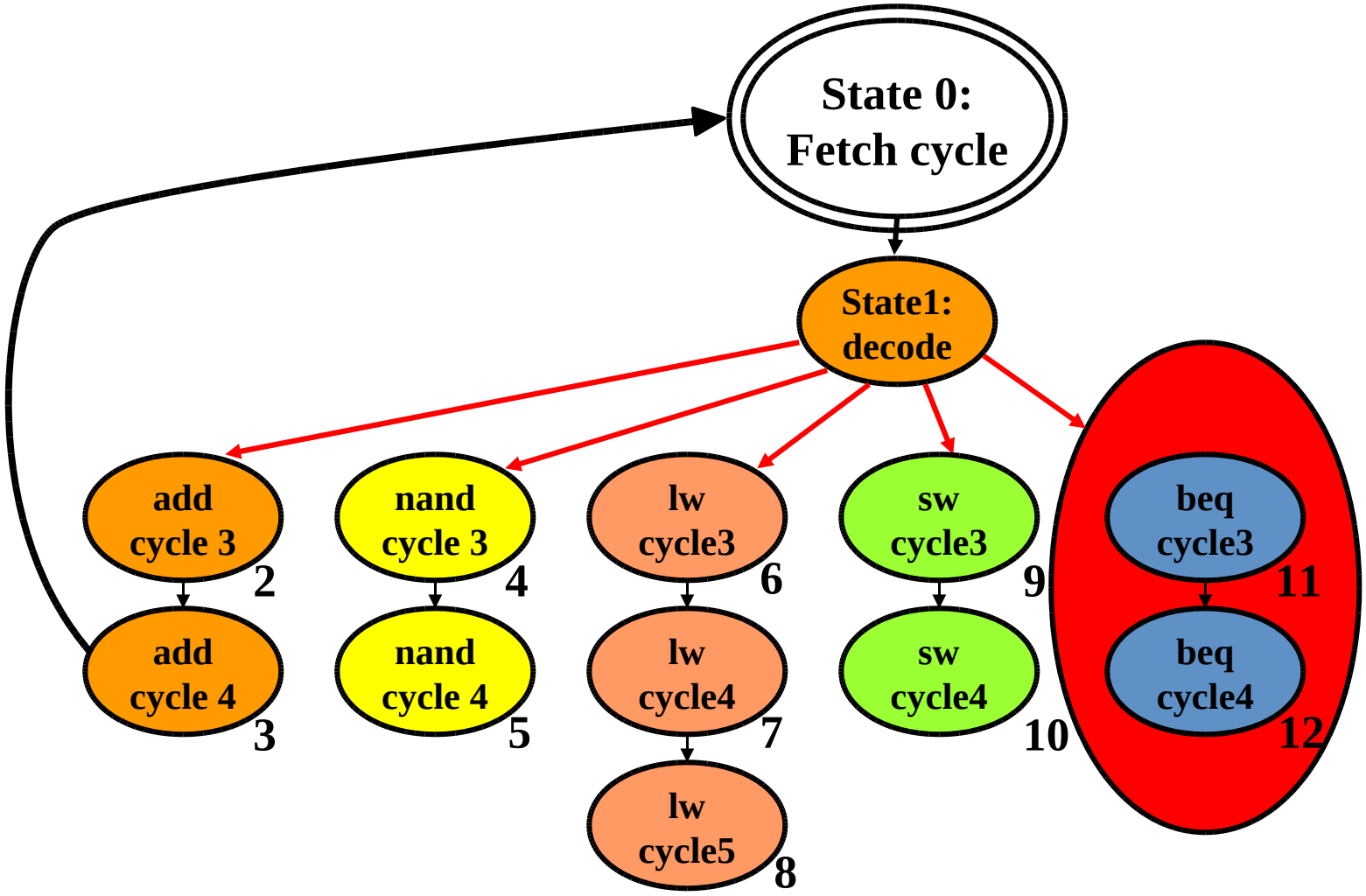
Return to State 0: fetch cycle to execute the next instruction



Control ROM settings (sw cycles 3 and 4)

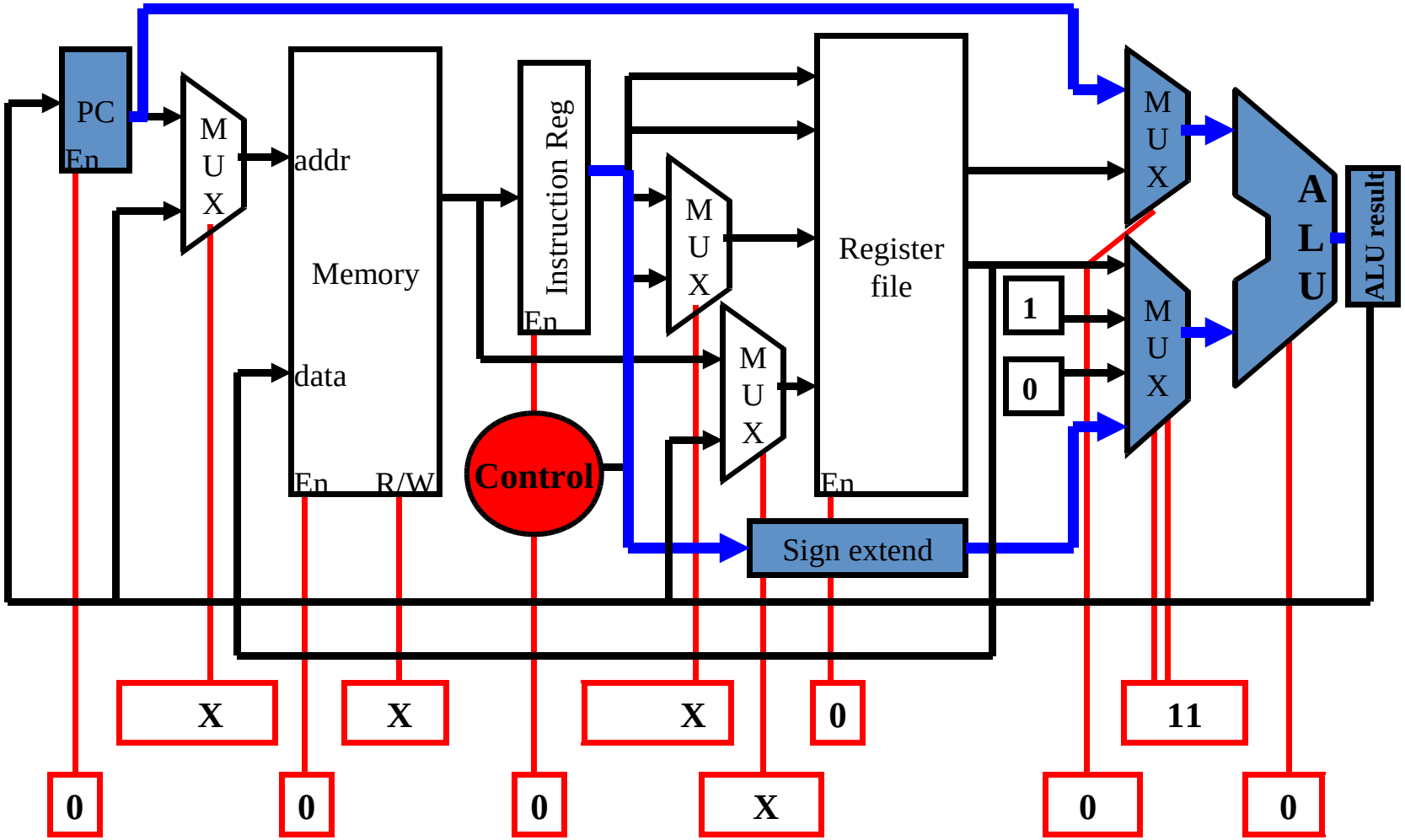


Return to State 0: fetch cycle to execute the next instruction

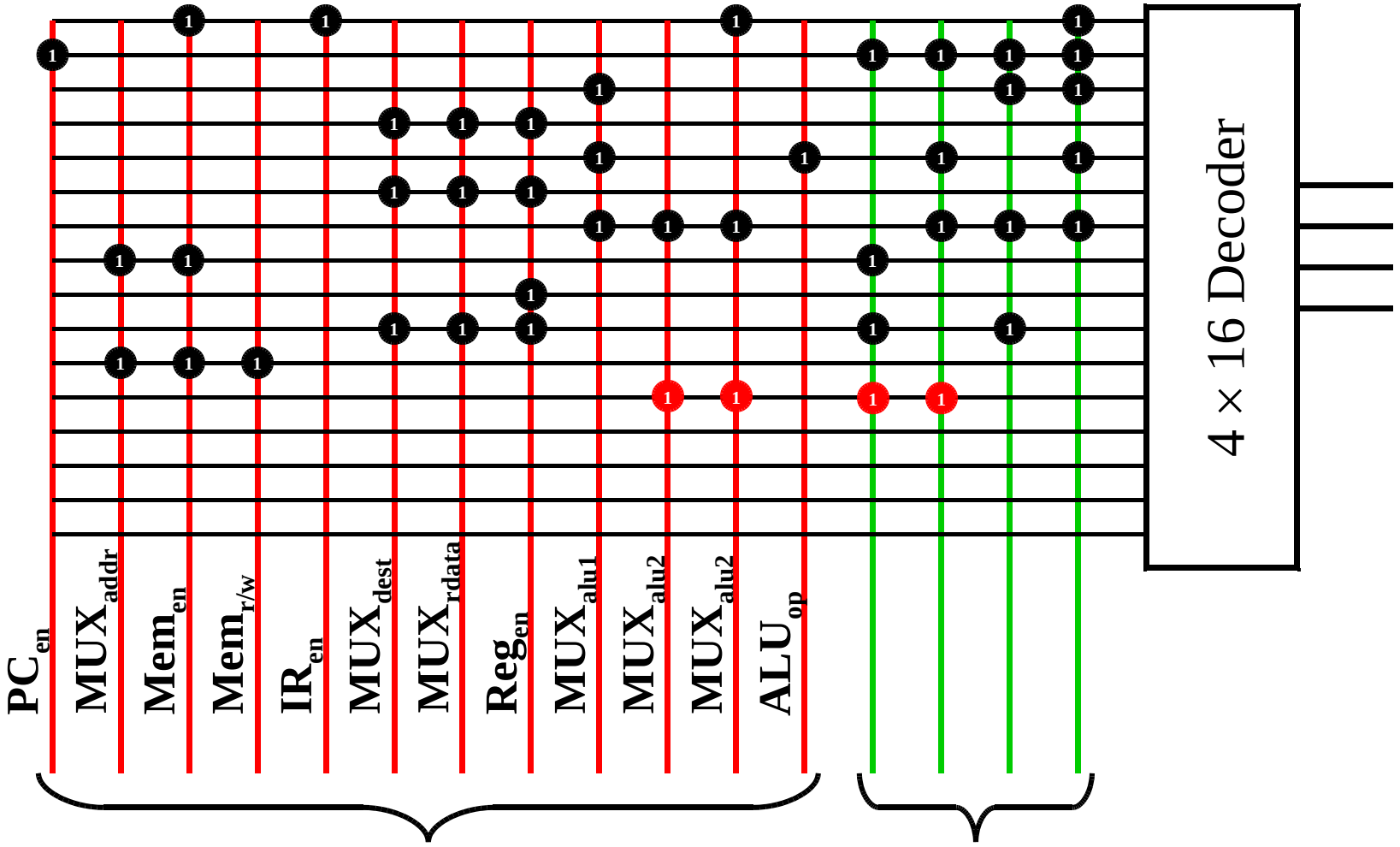


State 11: beq cycle 3

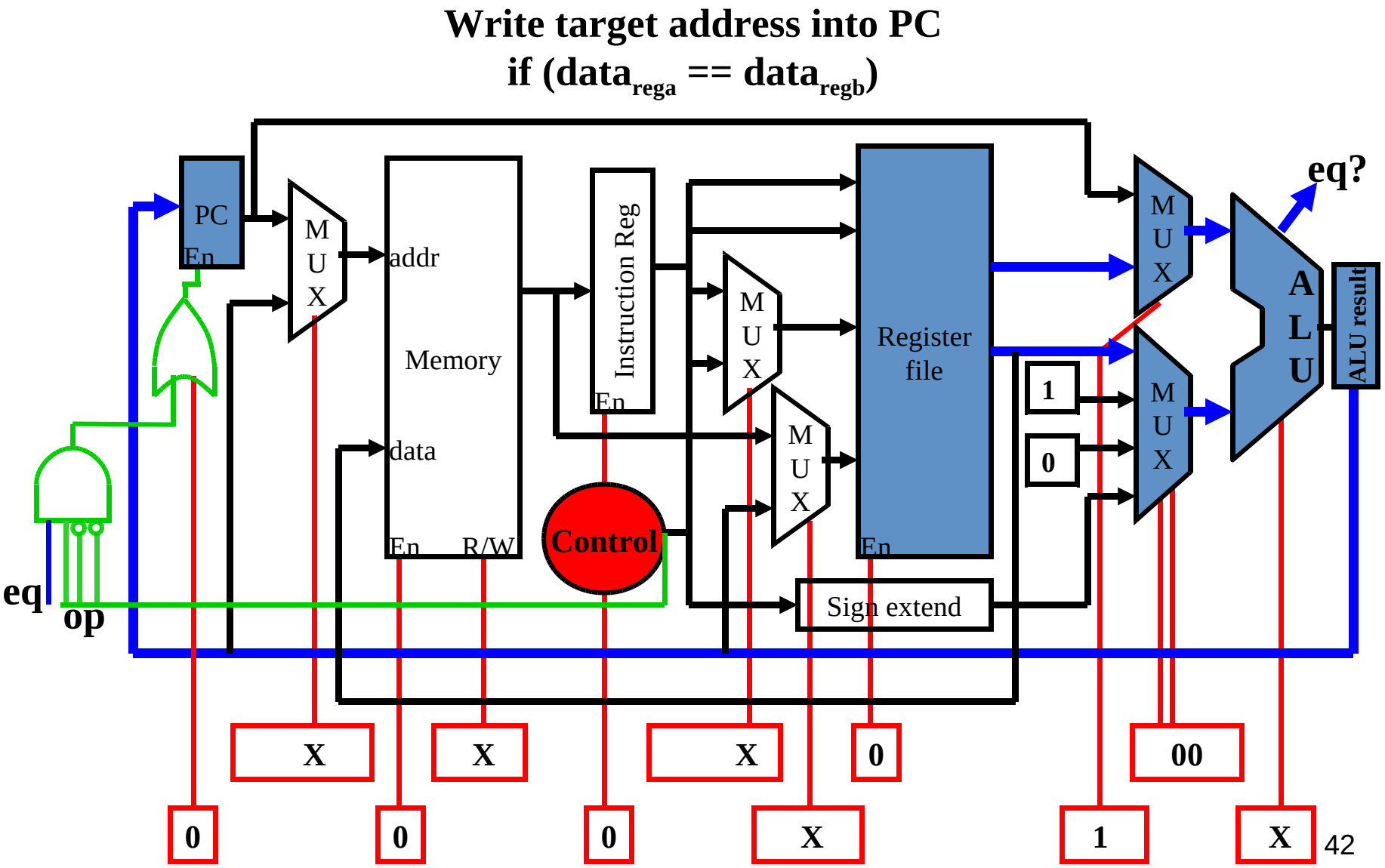
Calculate target address for branch



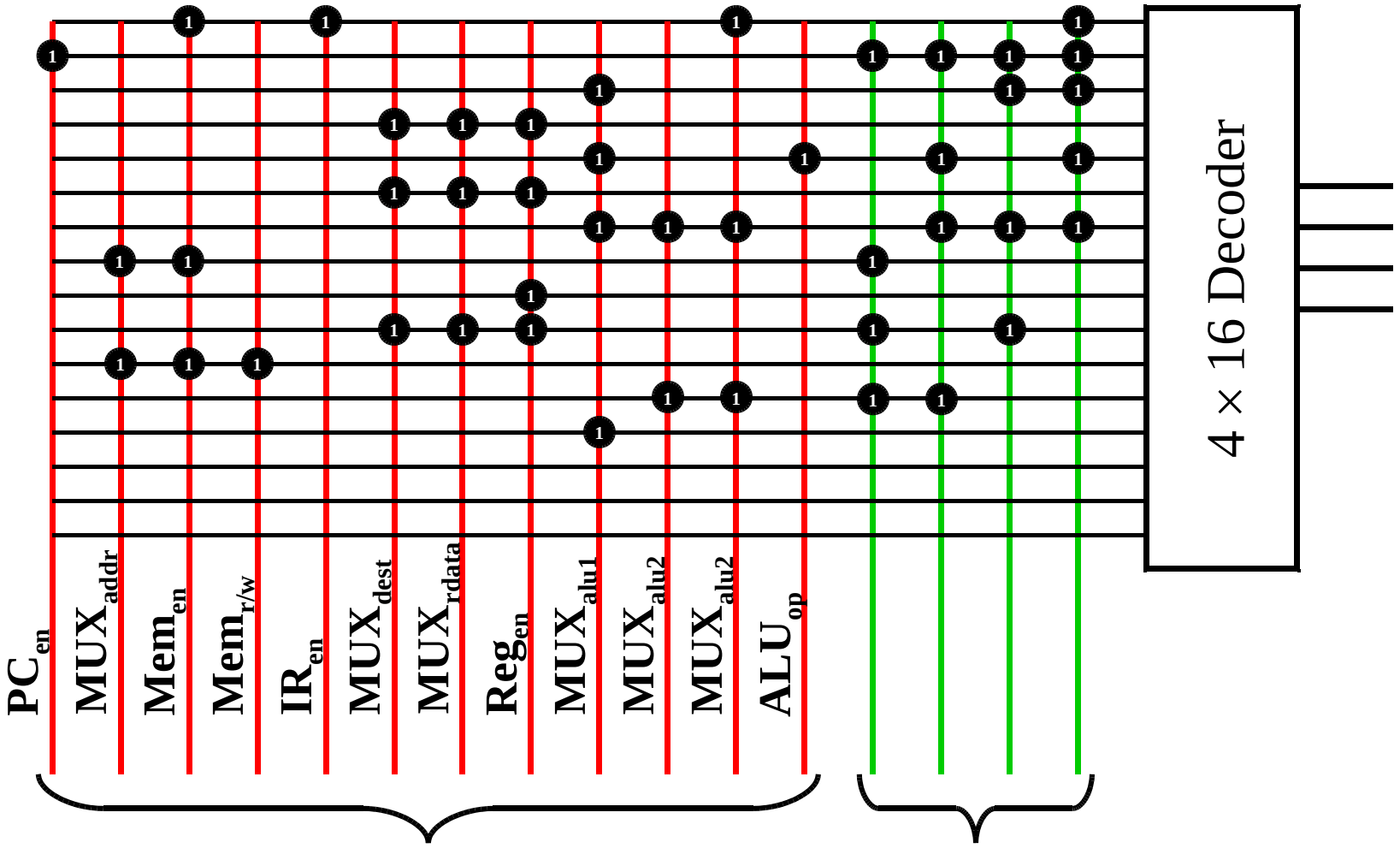
Control ROM settings (beq cycle 3)



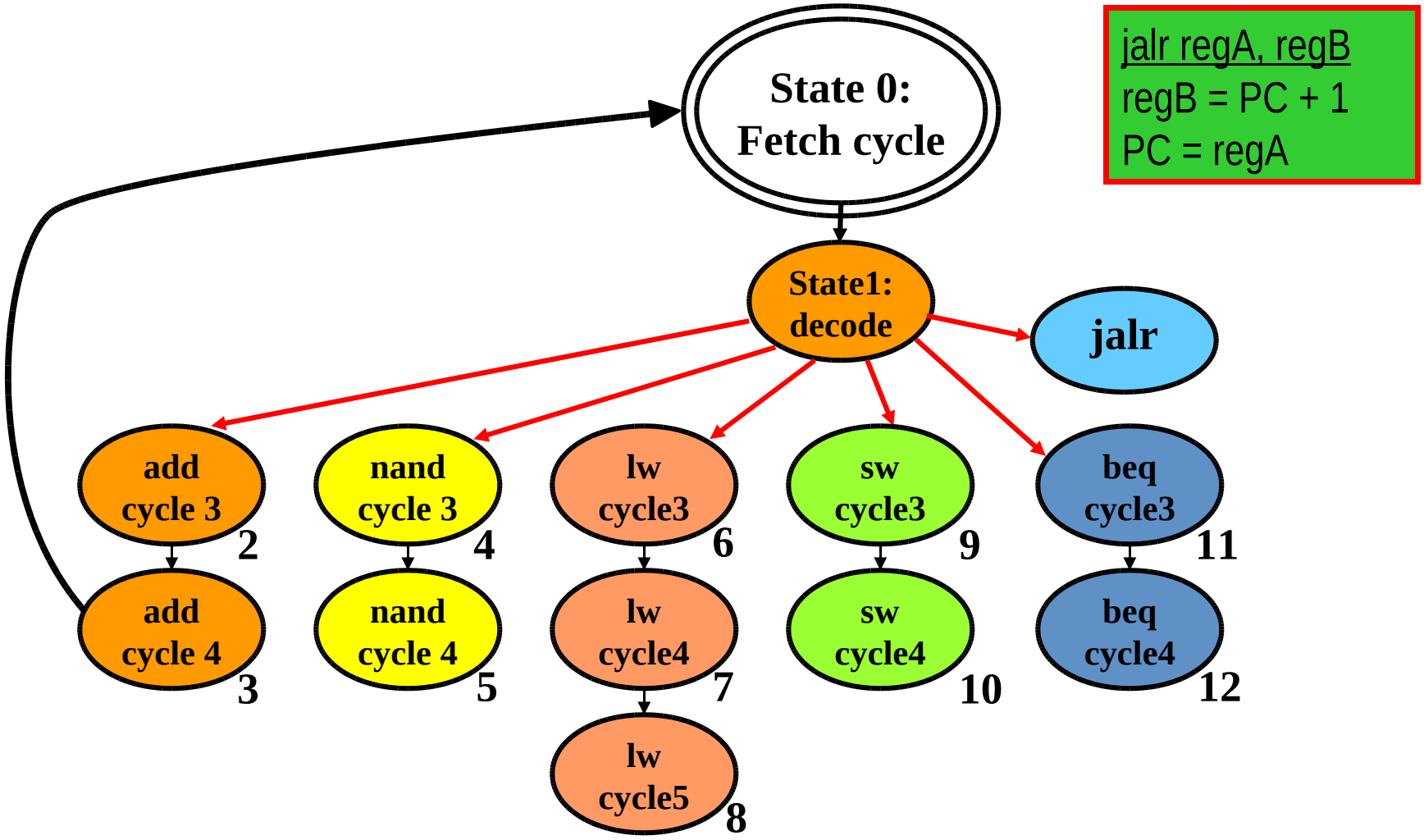
State 12: beq cycle 4



Control ROM settings (beq cycle 4)

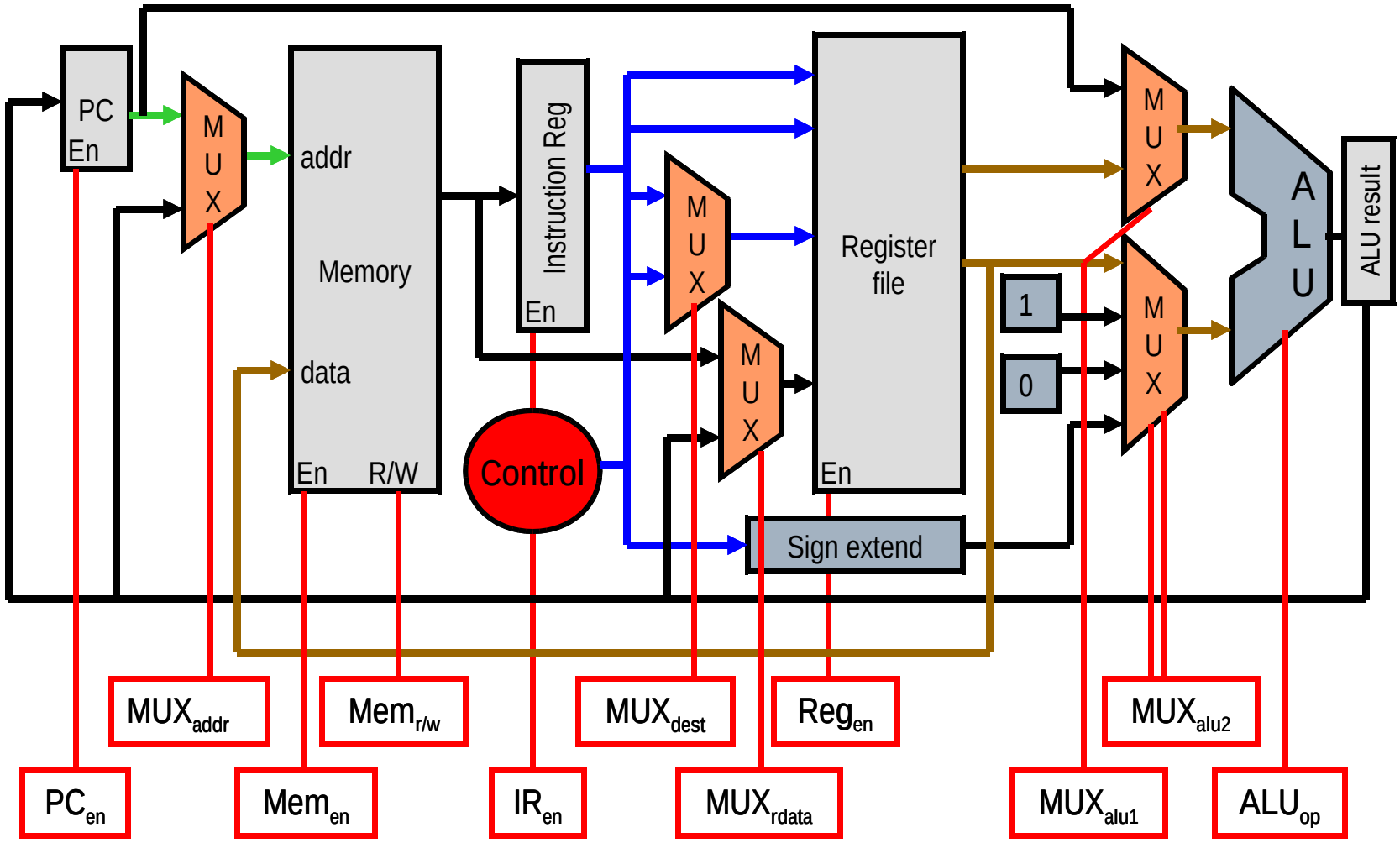


What about the jalr instruction?

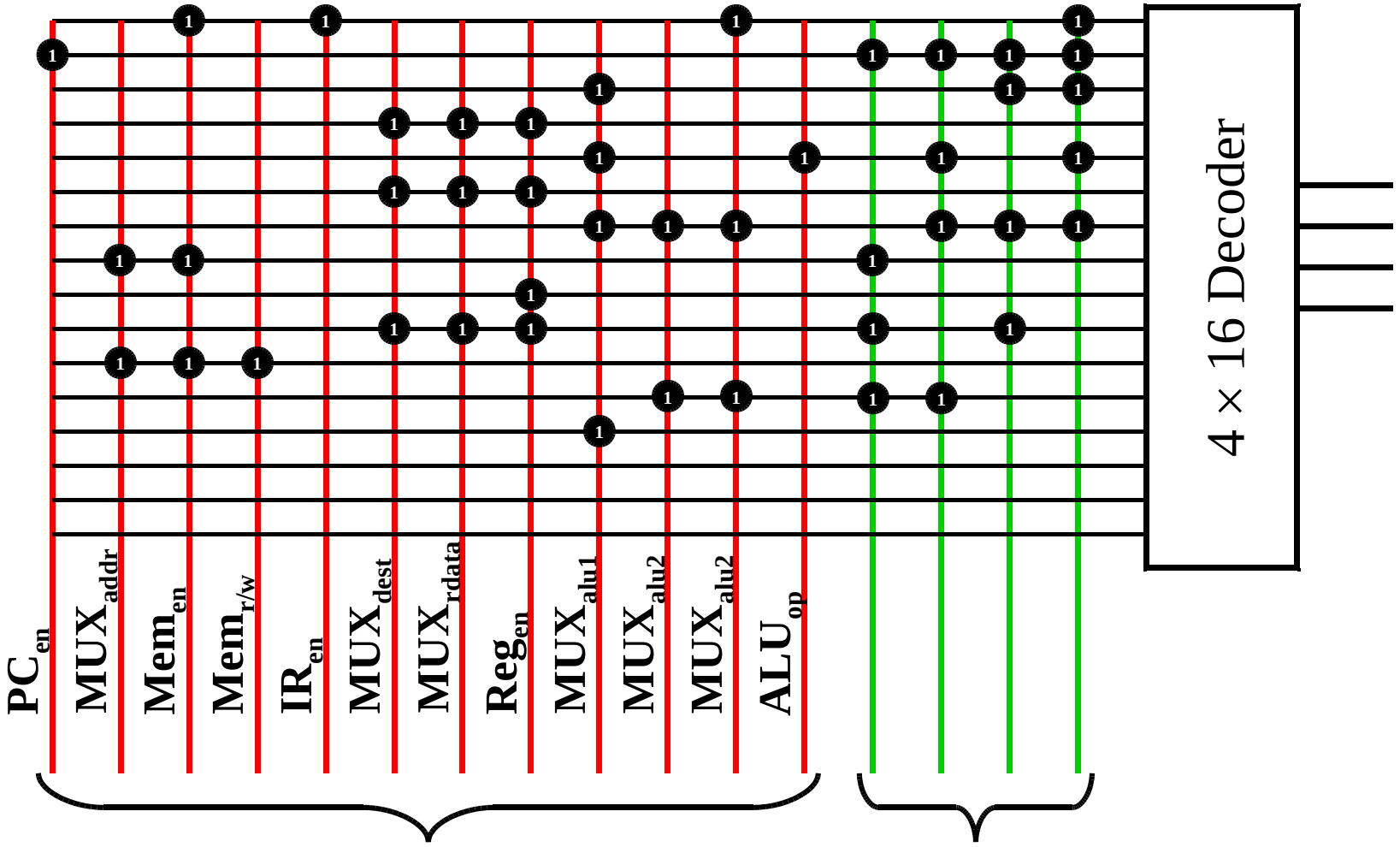


jalr regA, regB
regB = PC + 1
PC = regA

Multicycle LC2Kx datapath



Control ROM settings



Multicycle performance

Assume: 100 instructions executed

- 25% of instructions are loads,
- 10% of instructions are stores,
- 45% of instructions are adds, and
- 20% of instructions are branches.

How many cycles to execute this program on MC datapath?

Multicycle cycle time

Instr.	Get instr.	Read reg.	ALU	Mem.	Write reg.	Total
add	2 ns	1 ns	2 ns	0 ns	1 ns	6 ns
beq	2 ns	1 ns	2 ns	0 ns	0 ns	5 ns
sw	2 ns	1 ns	2 ns	2 ns	0 ns	7 ns
lw	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns

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

2. Assuming the above delays, is the example on the previous slide faster on the SC or MC design?

How to improve on this?

- Pipelining.
- Let additional instructions start before previous instructions have finished.