Basic Processor Design – Multicycle Processors

EECS 370 – Introduction to Computer Organization – Winter 2015

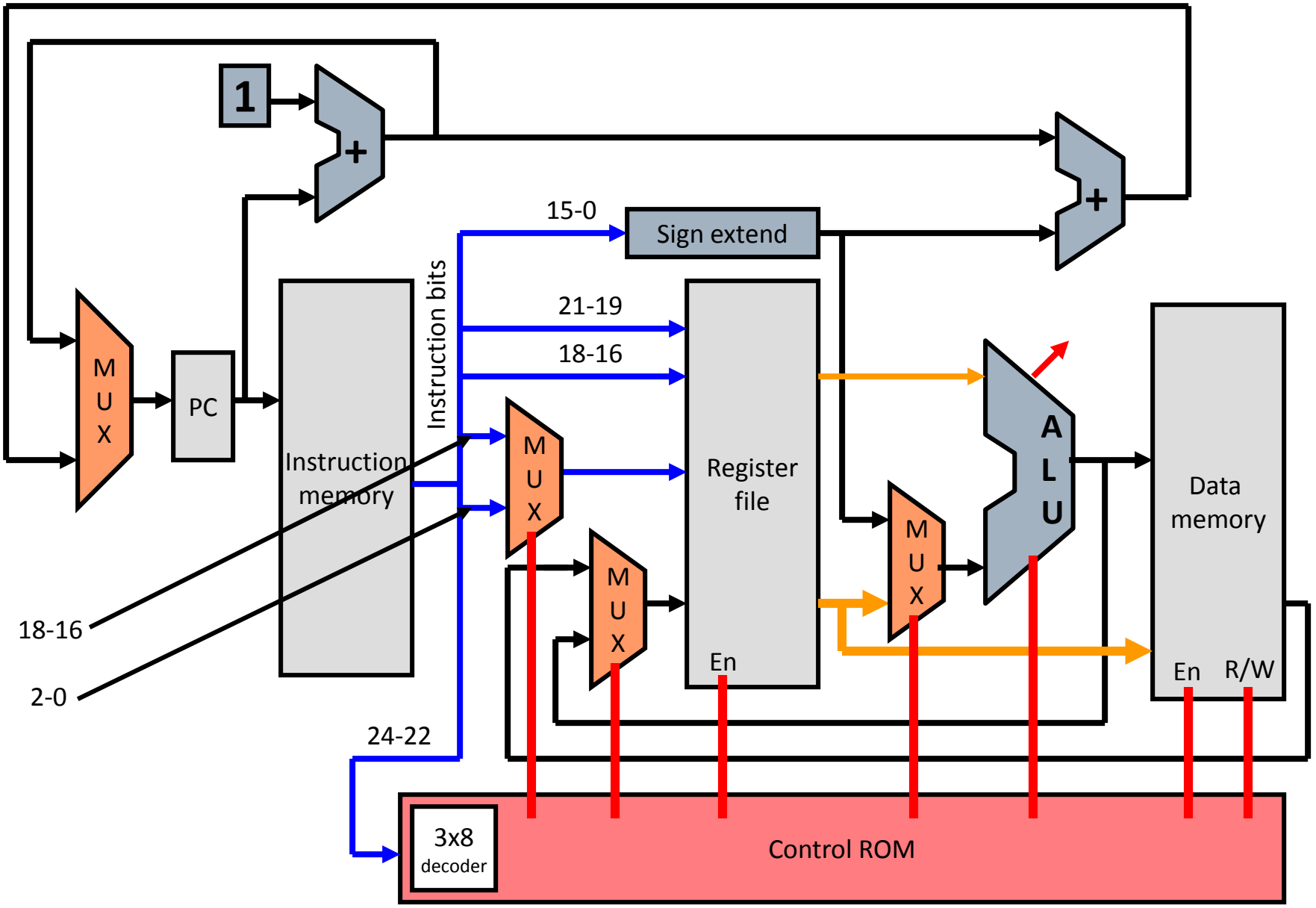
Robert Dick, Andrew Lukefahr, and Satish Narayanasamy

EECS Department
University of Michigan in Ann Arbor, USA

Midterm 1

- 7pm-9pm, 24 Feb
- Closed book and notes, with one exception
- Bring one double-sided 8.5”x11” sheet of paper with notes.
- No devices that allow wireless communication.
- Practice exams on website.
- Exam timing doesn’t line up.
- Topics this semester may differ from prior semesters, e.g., ARM vs. MIPS.
- List of topics covered so far has been posted to website.

Recap (LC2Kx single cycle datapath)



Recap: What's Wrong with Single Cycle?

- **All instructions run at the speed of the slowest instruction.**
- Adding a long instruction can hurt performance
  - What if you wanted to include multiply?
- You cannot reuse any parts of the processor
  - We have 3 different adders to calculate PC+1, PC+1+offset and the ALU
- No benefit in making the common case fast
  - Since every instruction runs at the slowest instruction speed
    - This is particularly important for loads as we will see later

Recap: What's Wrong with Single Cycle?

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

	Get Instr	**read reg**	**ALU oper.**	**mem**	**write reg**	
• add:	2ns	+ 1ns	+ 2ns		+ 1 ns	= 6 ns
• beq:	2ns	+ 1ns	+ 2ns			= 5 ns
• sw:	2ns	+ 1ns	+ 2ns	+ 2ns		= 7 ns
• lw:	2ns	+ 1ns	+ 2ns	+ 2ns	+ 1ns	= 8 ns

Recap: Computing 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:

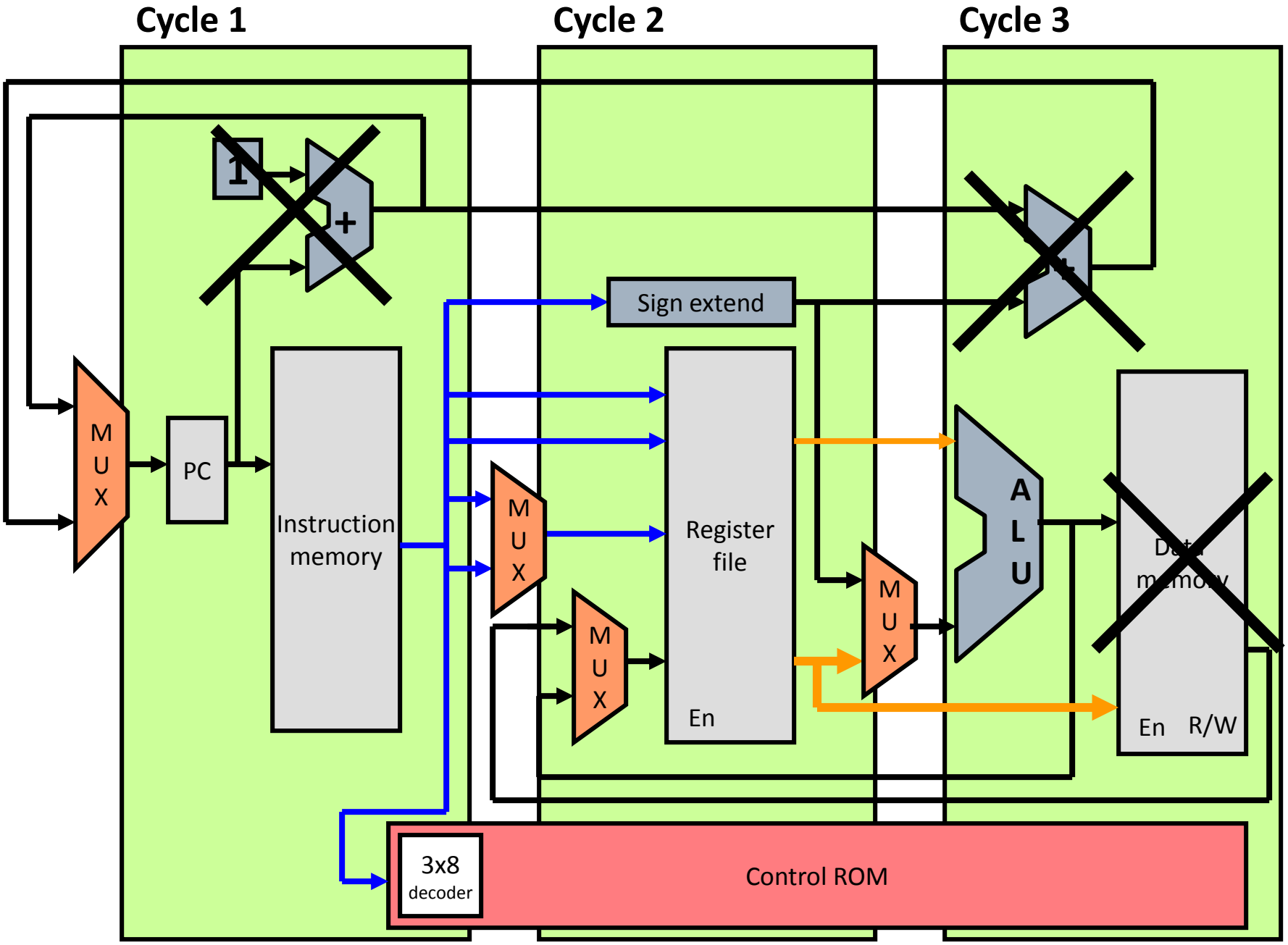
- 100 * 8ns = **800** ns

Optimal execution:

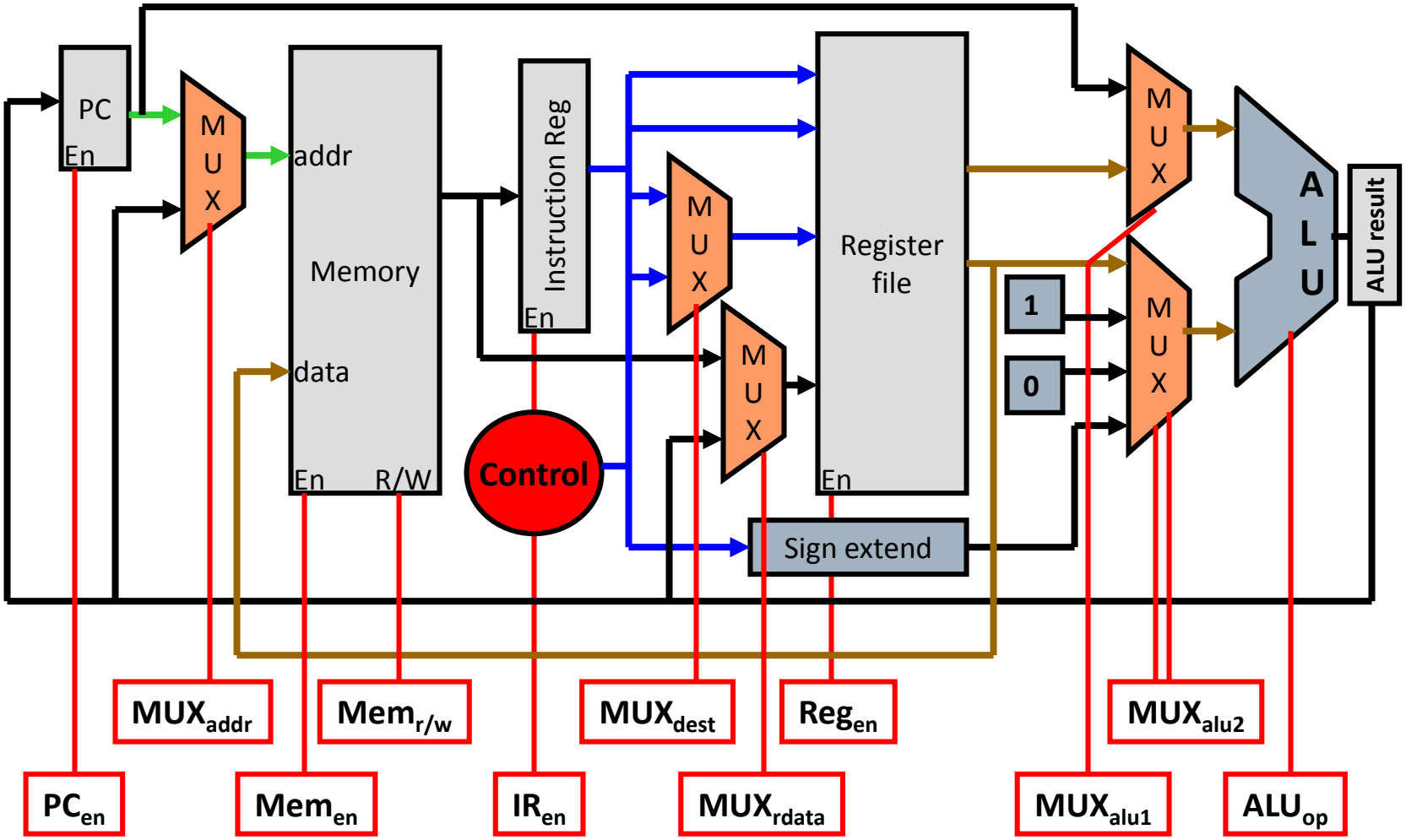
- 25*8ns + 10*7ns + 45*6ns + 20*5ns = **640** ns

Multiple-Cycle Execution

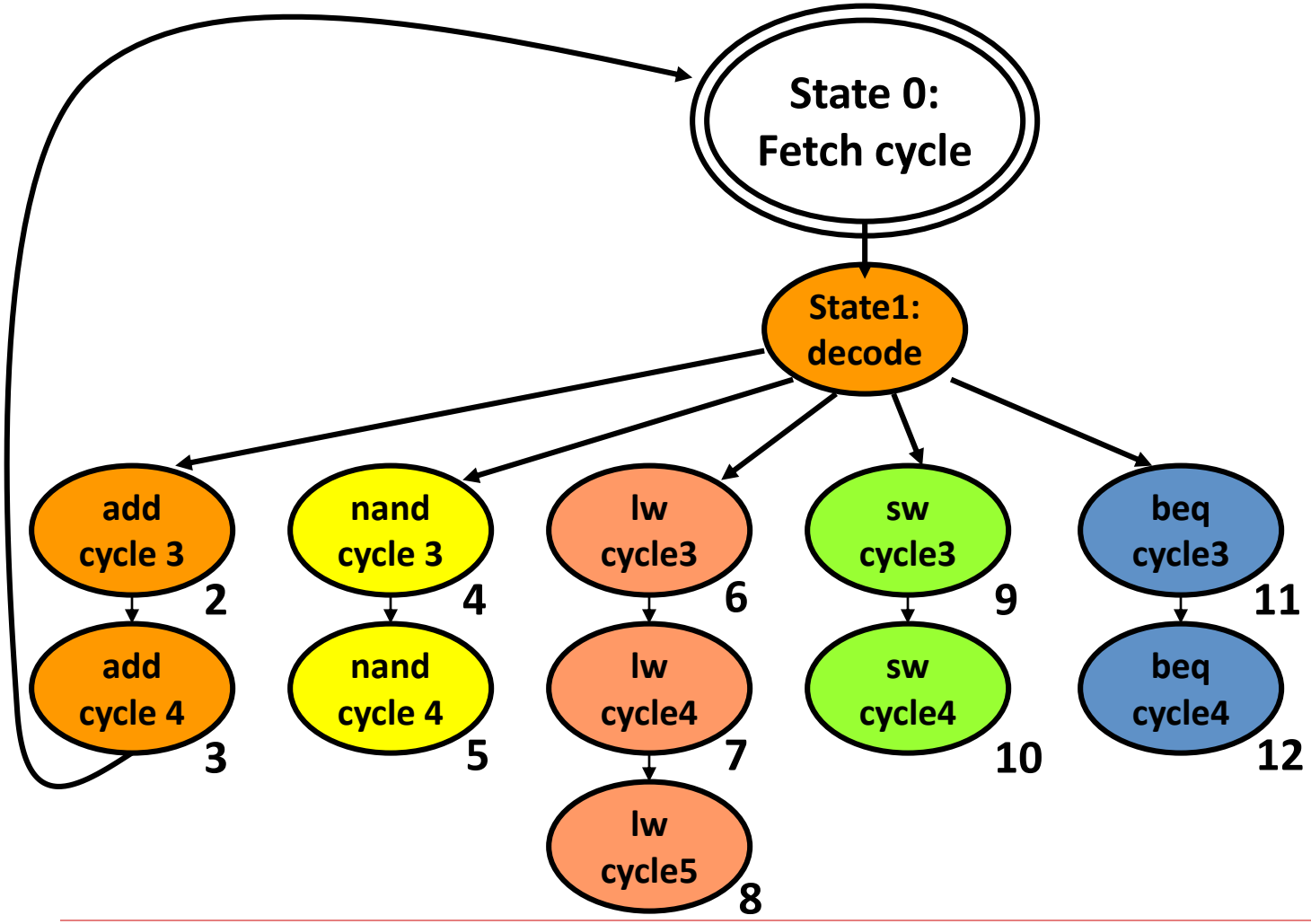
- Each instruction takes multiple cycles to execute
  - Cycle time is reduced
  - Slower instructions take more cycles
  - Can reuse datapath elements each cycle
- What is needed to make this work?
  - Since you are re-using elements for different purposes, you need more and/or wider MUXes.
  - You may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

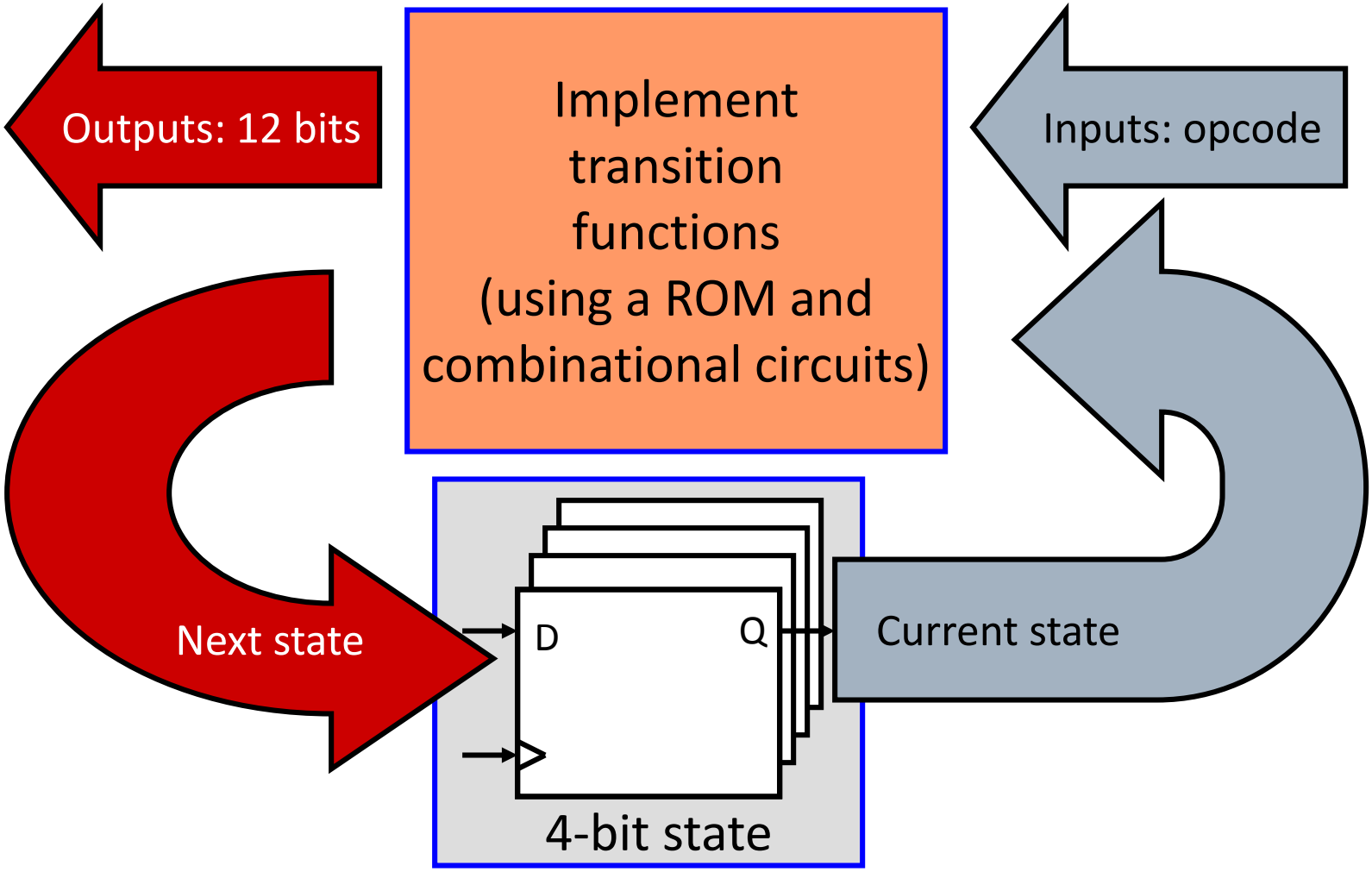
Multicycle LC2Kx Datapath



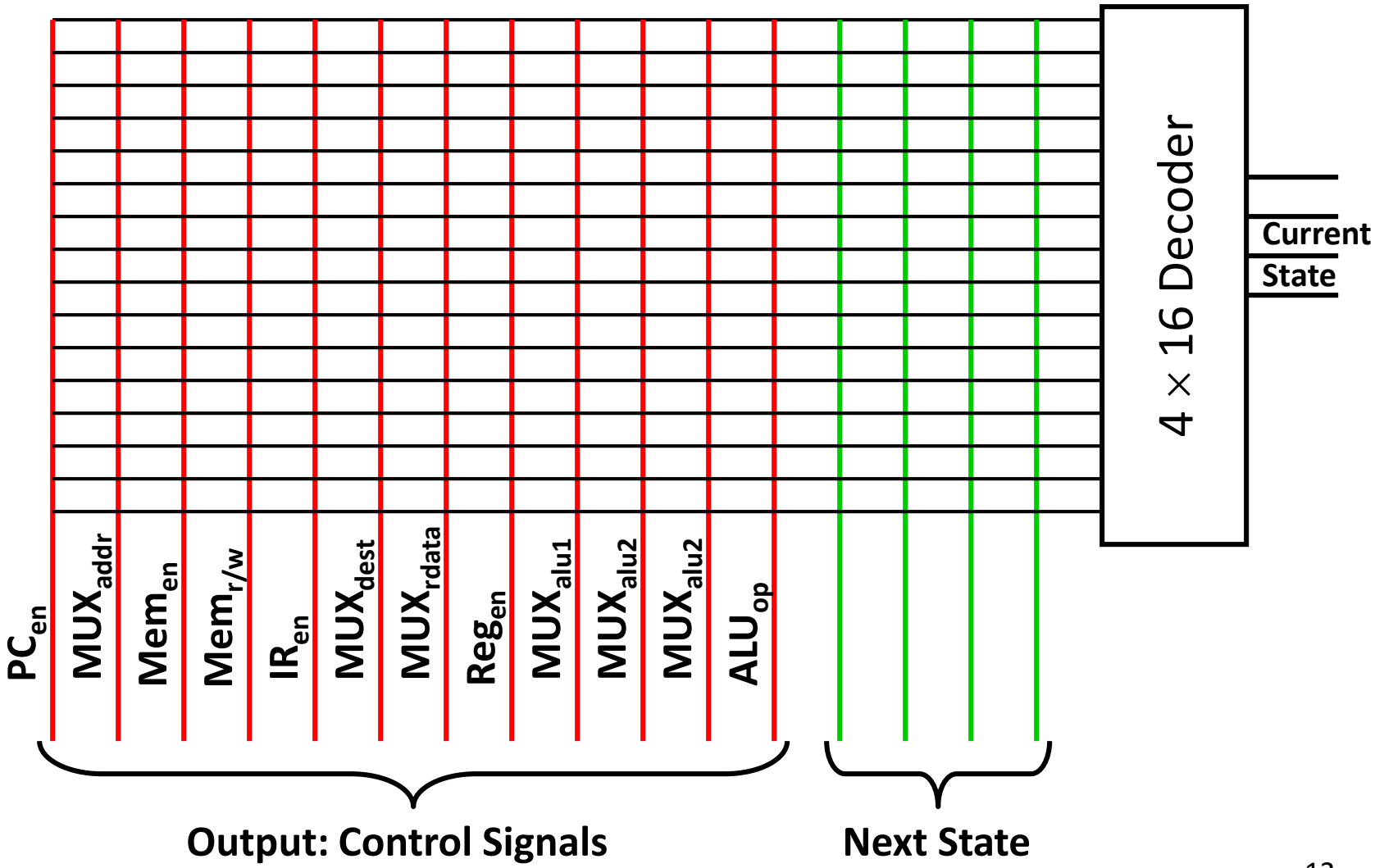
State machine for multi-cycle control signals (transition functions)



Implementing FSM



Building the Control Rom

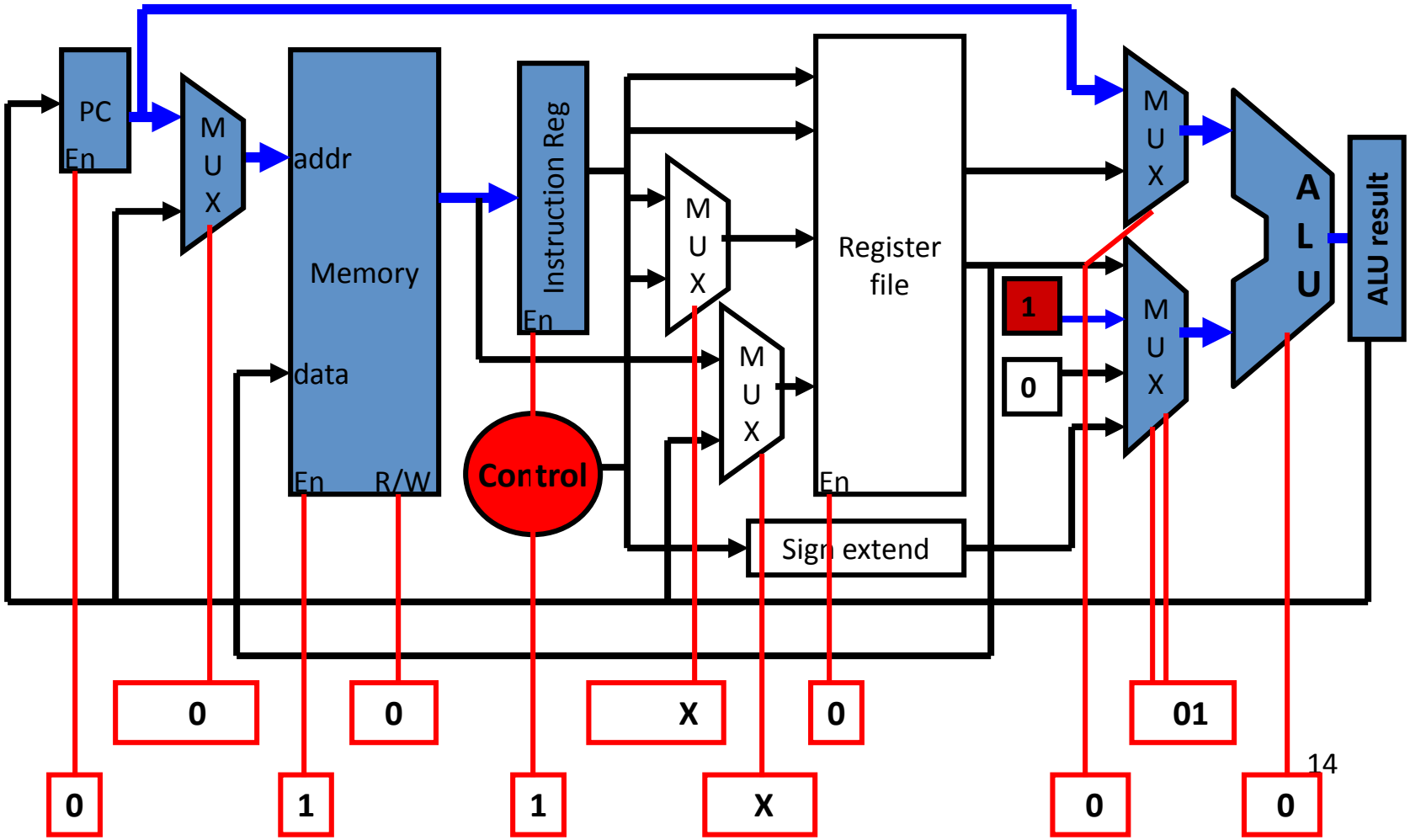


First Cycle (State 0) Fetch Instr

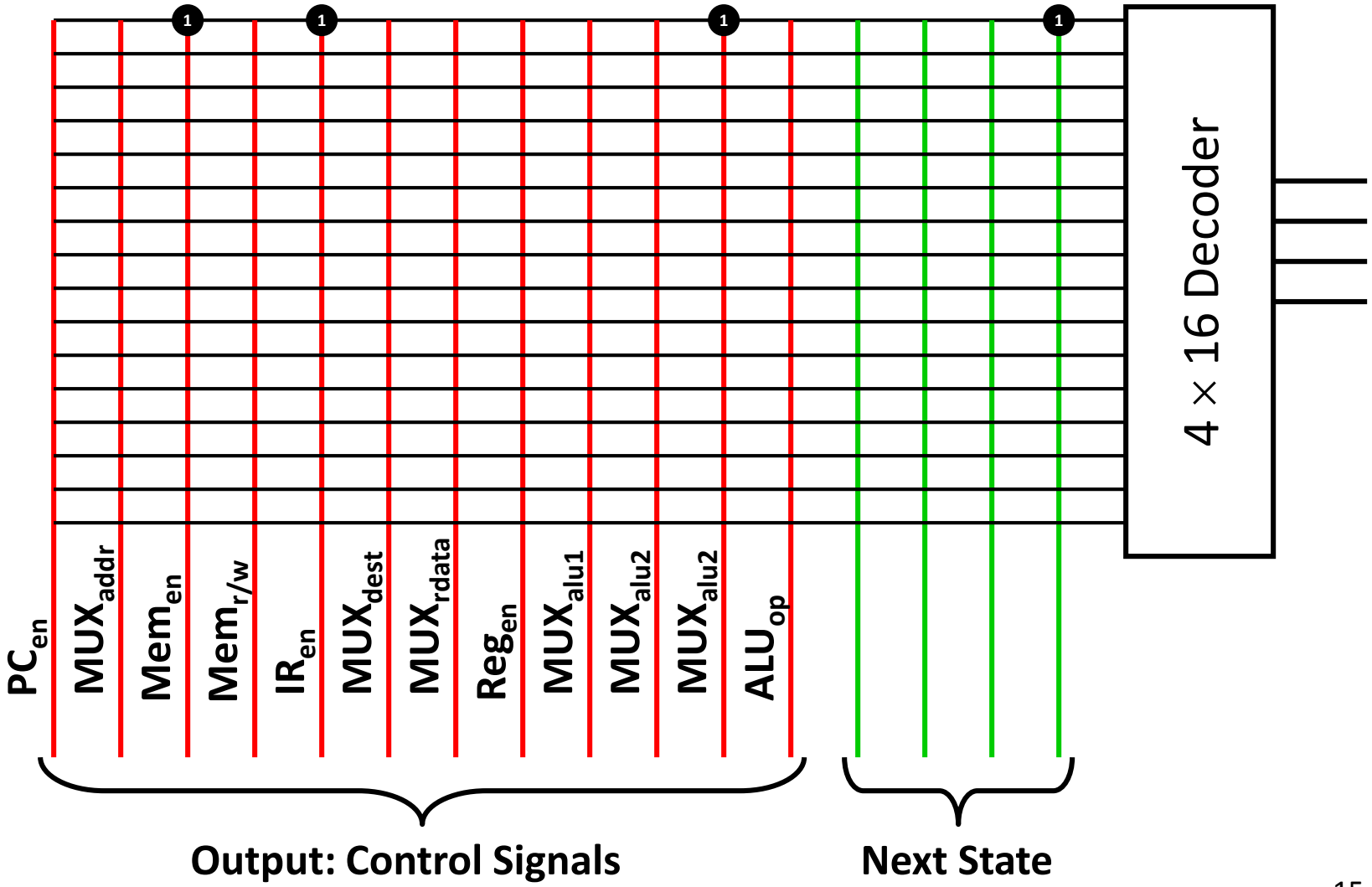
- What operations need to be done in the first cycle of executing any instruction?
  - Read memory[PC] and store into instruction register.
    - Must select PC in memory address MUX (MUX_{addr}= 0)
    - Enable memory operation (Mem_{en}= 1)
    - R/W should be (read) ($Mem_{r/w}$= 0)
    - Enable Instruction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

Cycle 1 Operation

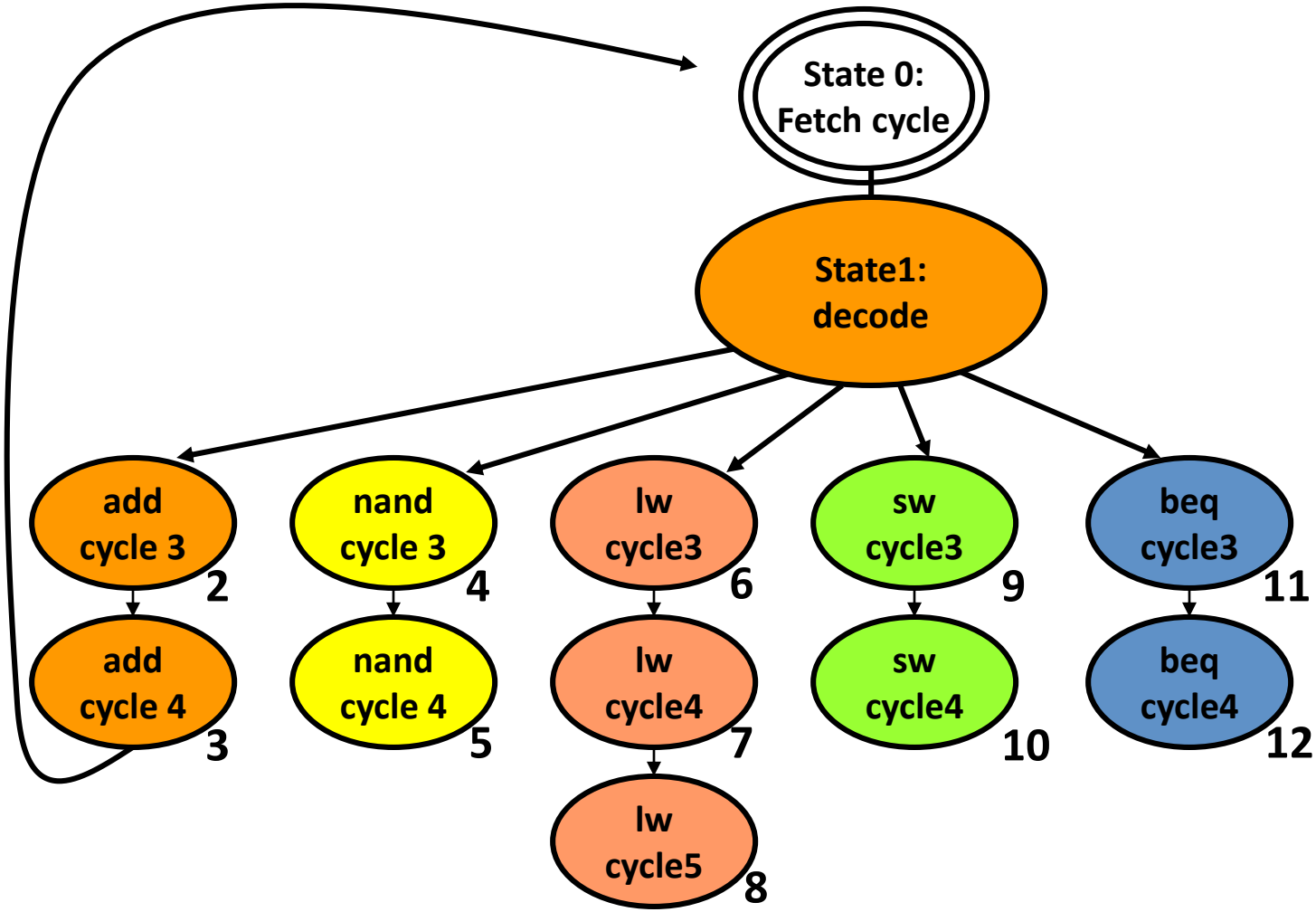
This is the same for all instructions
(since we don't know the instruction yet!)



Building the Control Rom

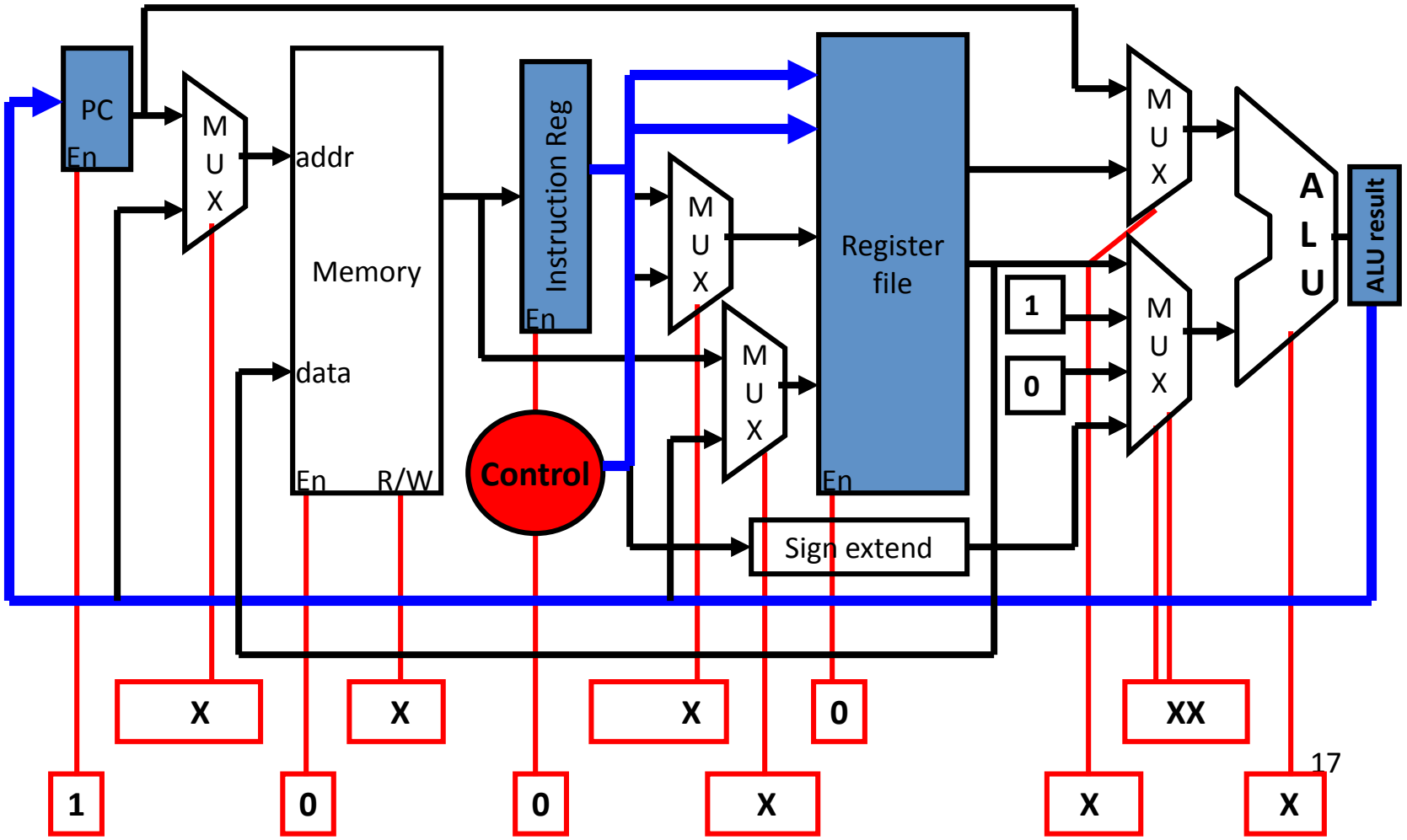


State 1: instruction decode

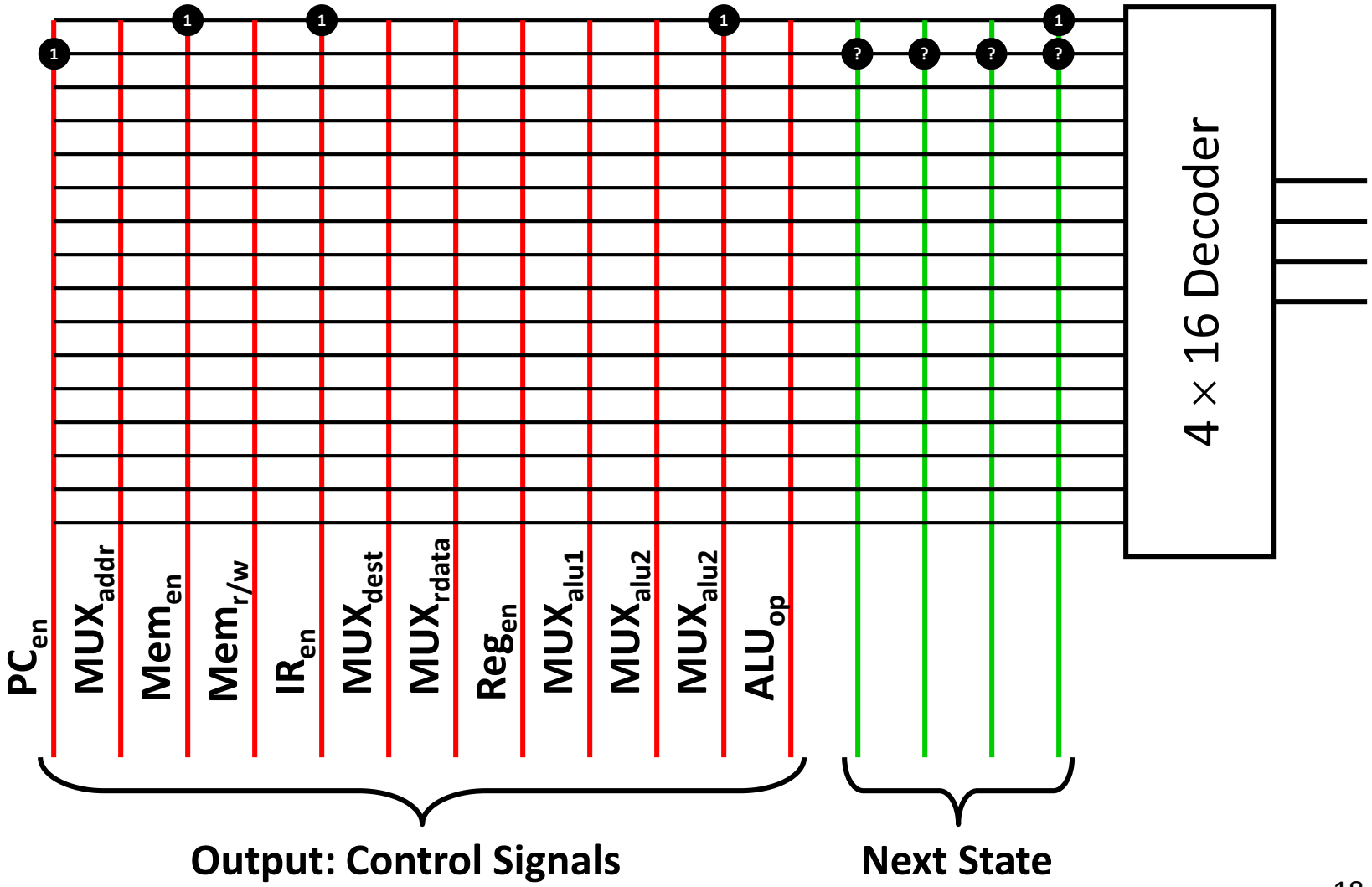


State 1: output function

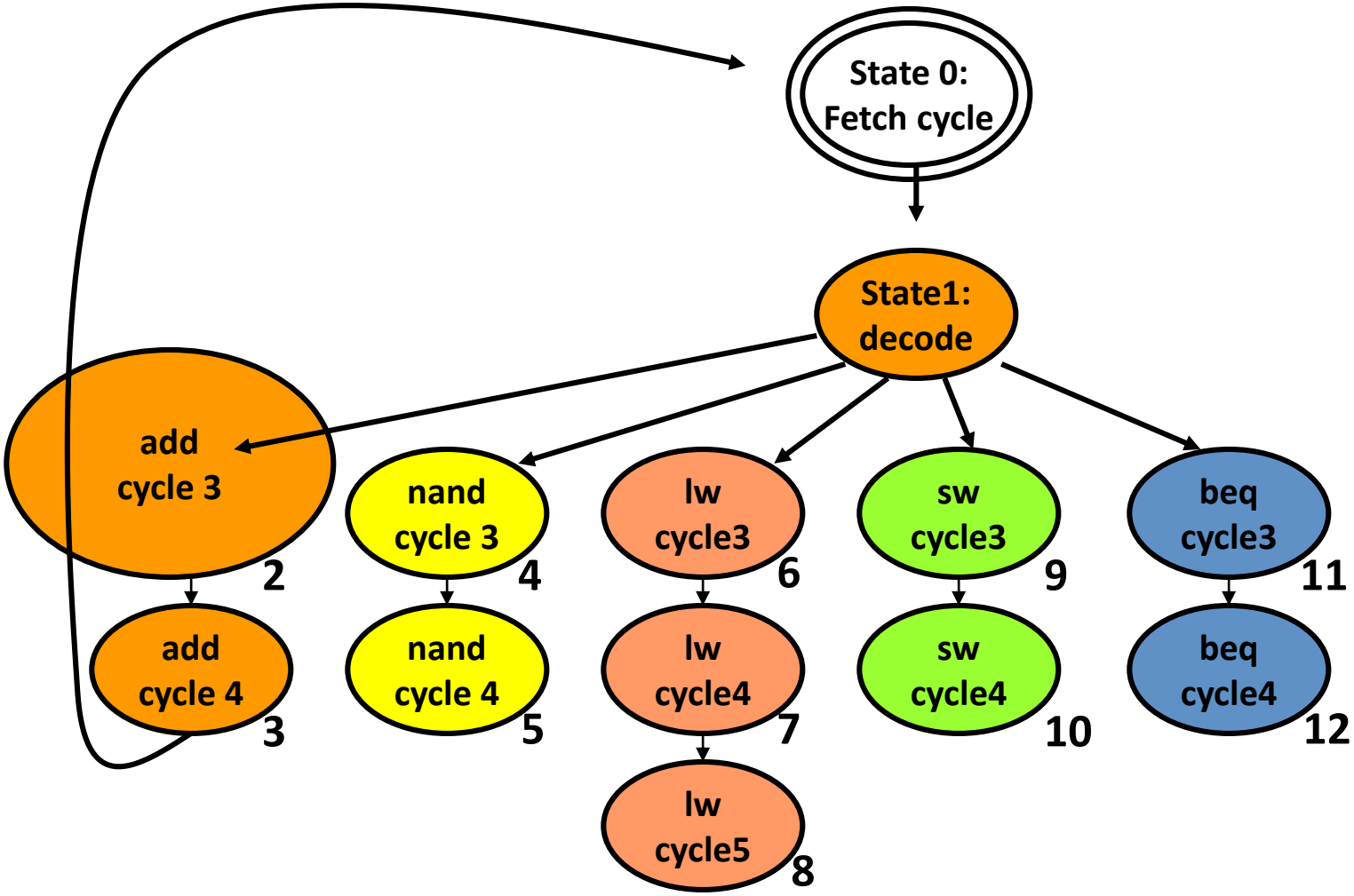
Update PC; read registers (regA and regB);
use opcode to determine next state



Building the Control Rom

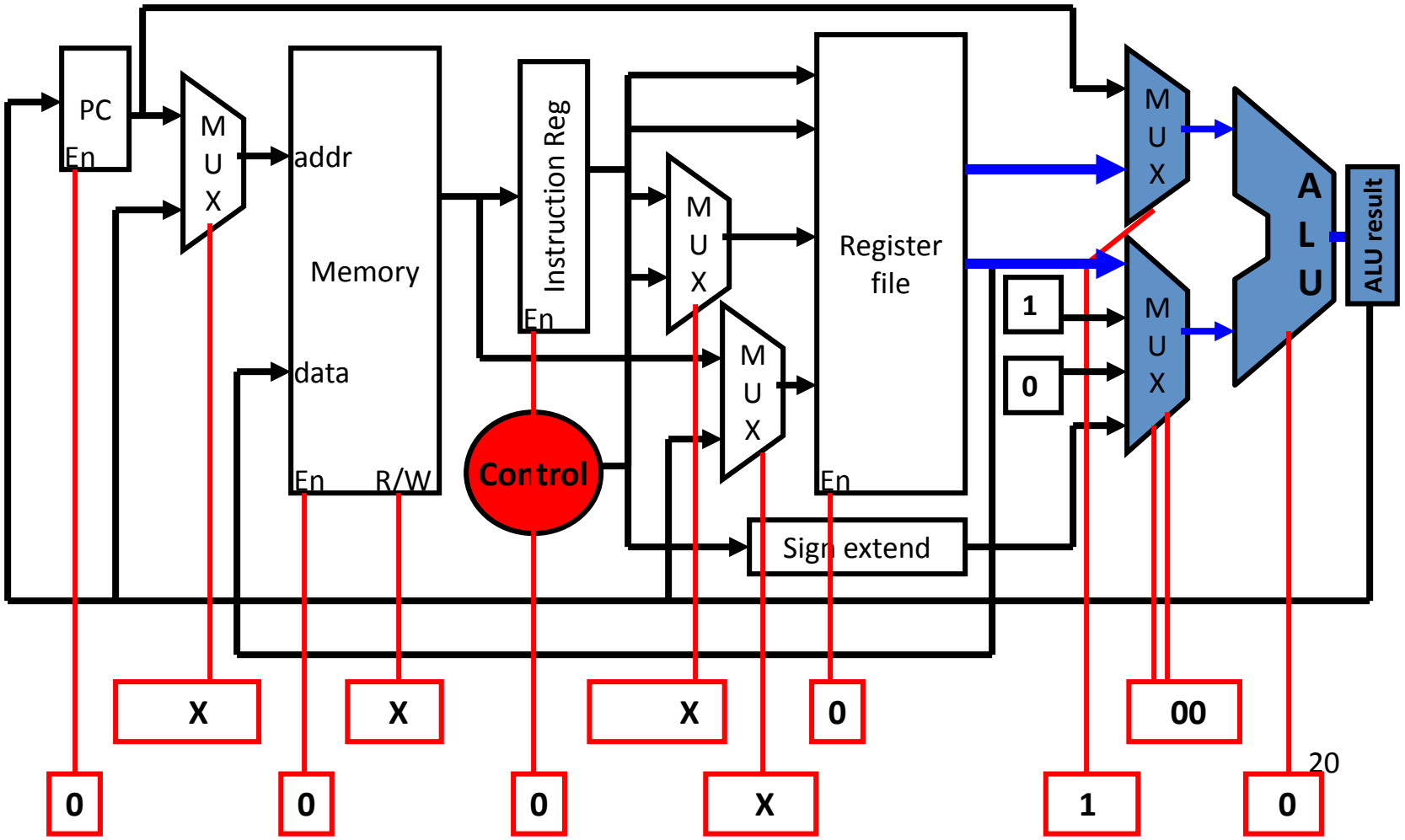


State 2: Add cycle 3

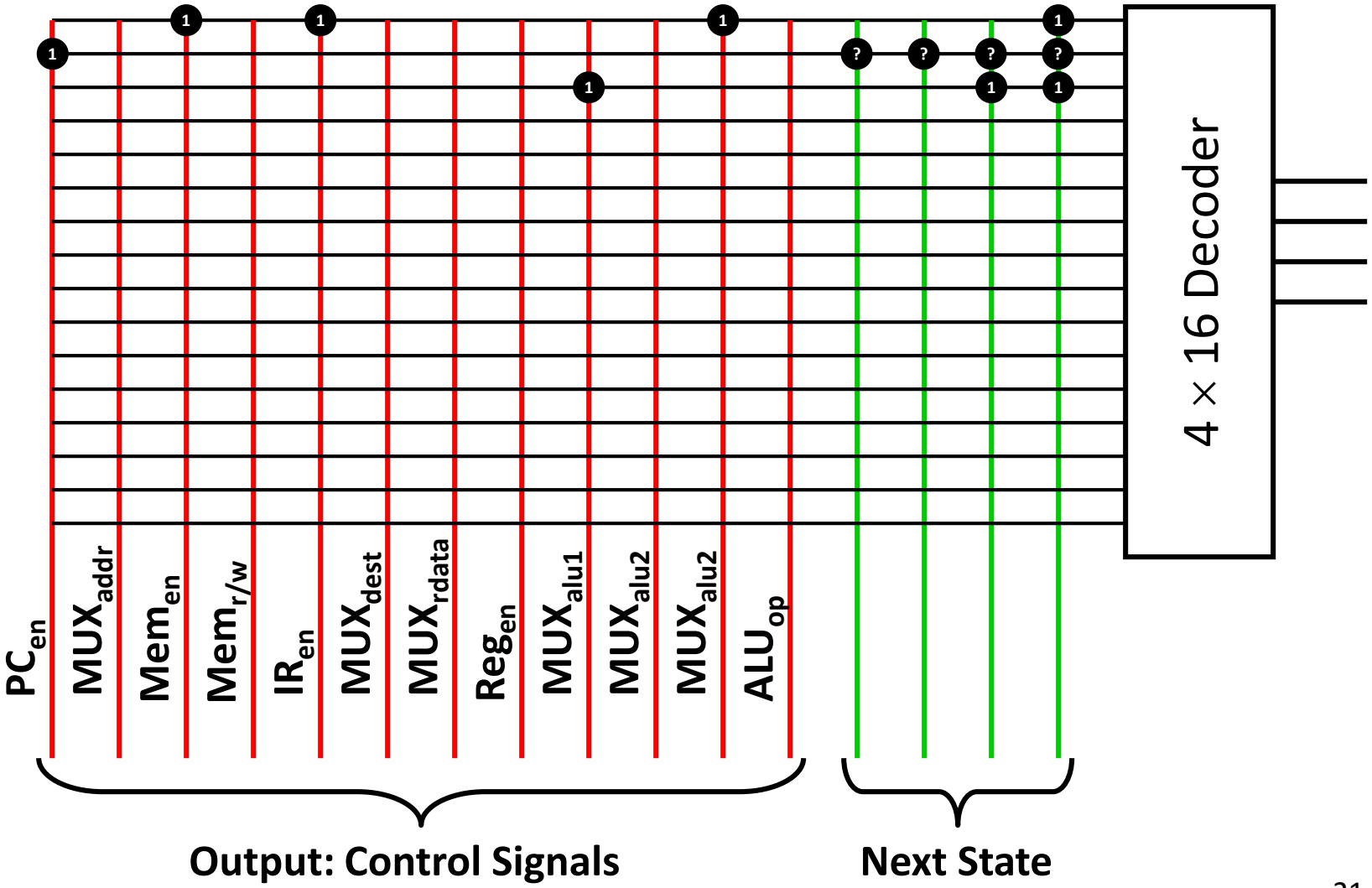


State 2: Add Cycle 3 Operation

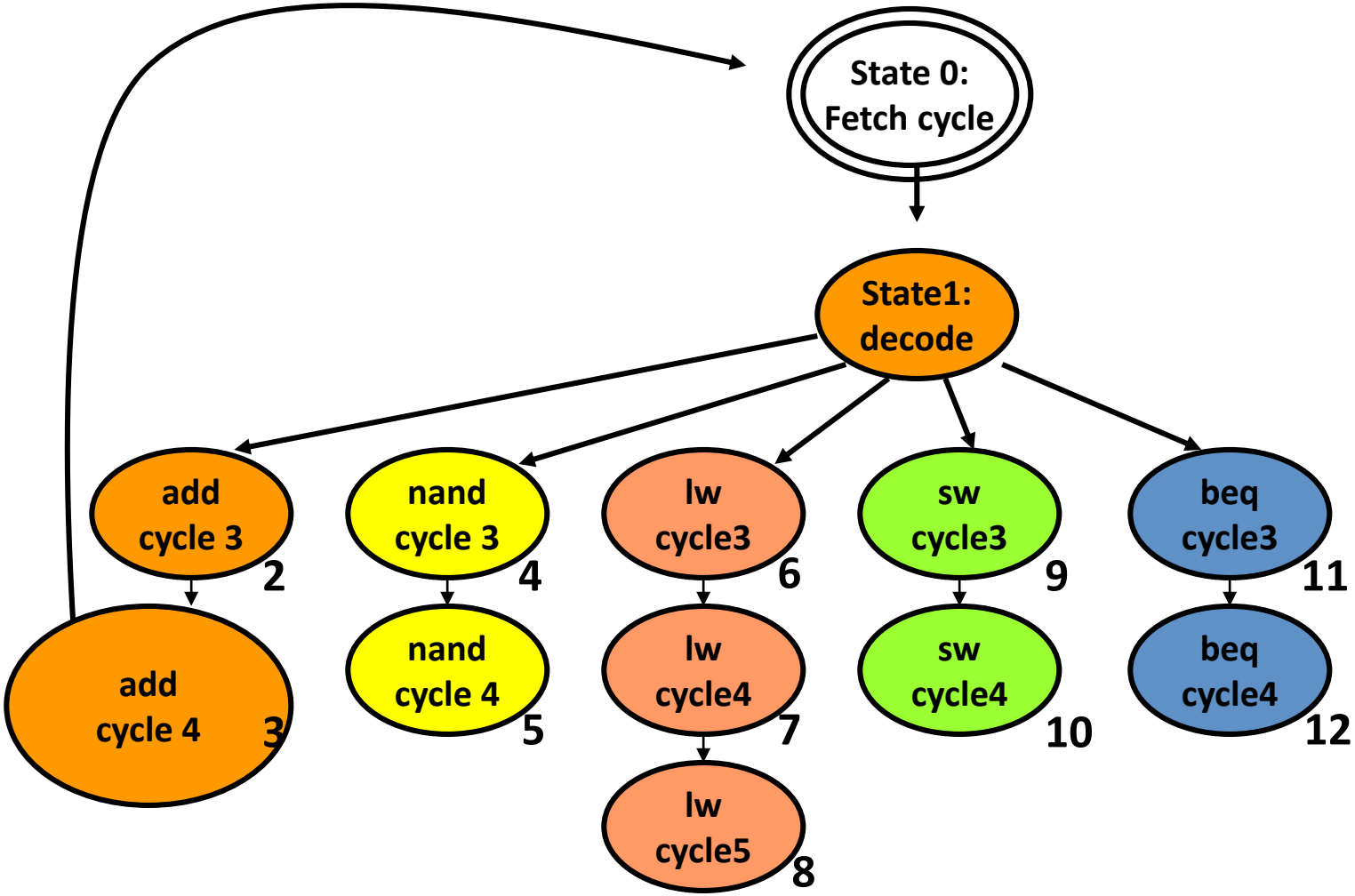
Send control signals to MUX to select values of regA and regB and control signal to ALU to add



Building the Control Rom

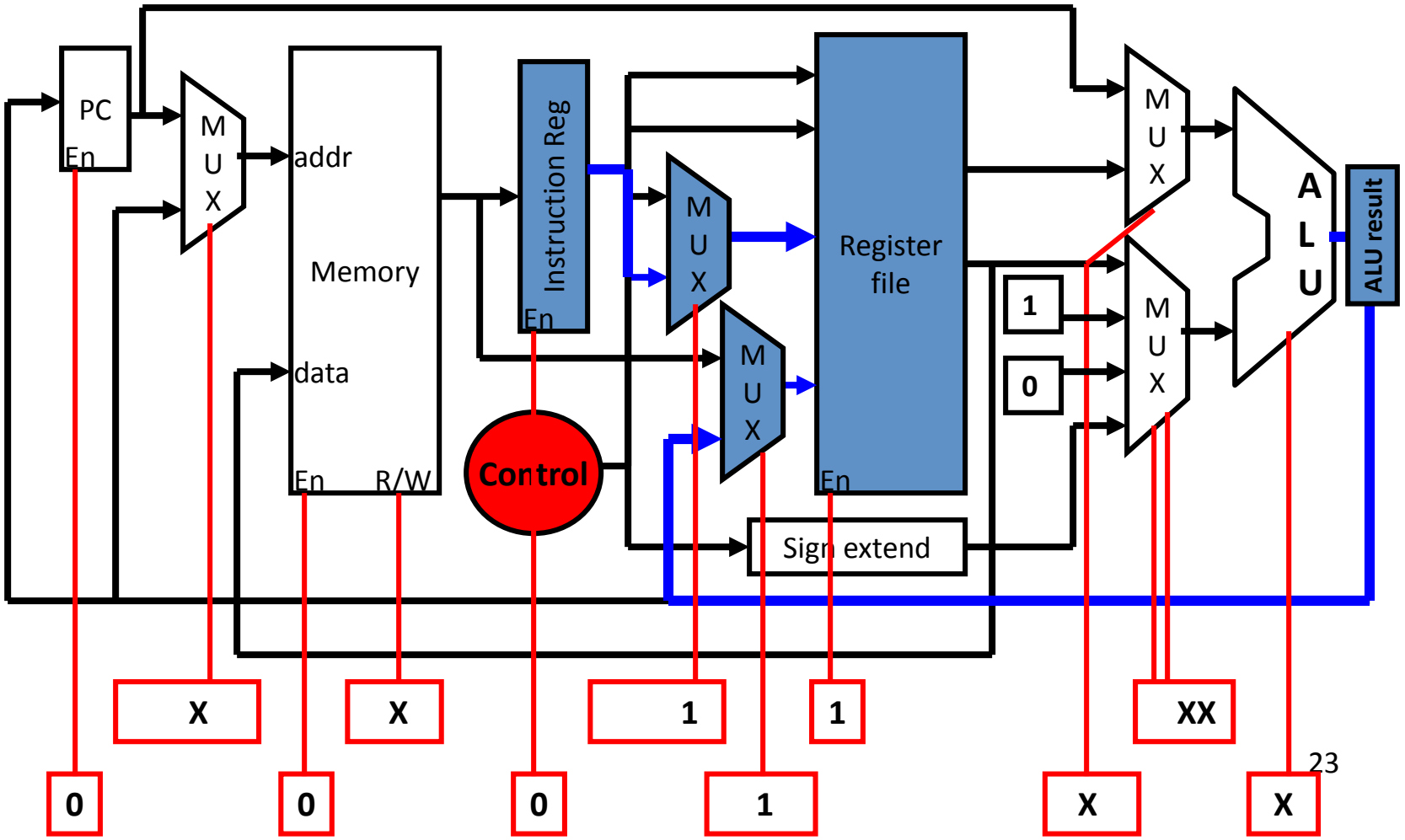


State 3: Add cycle 4

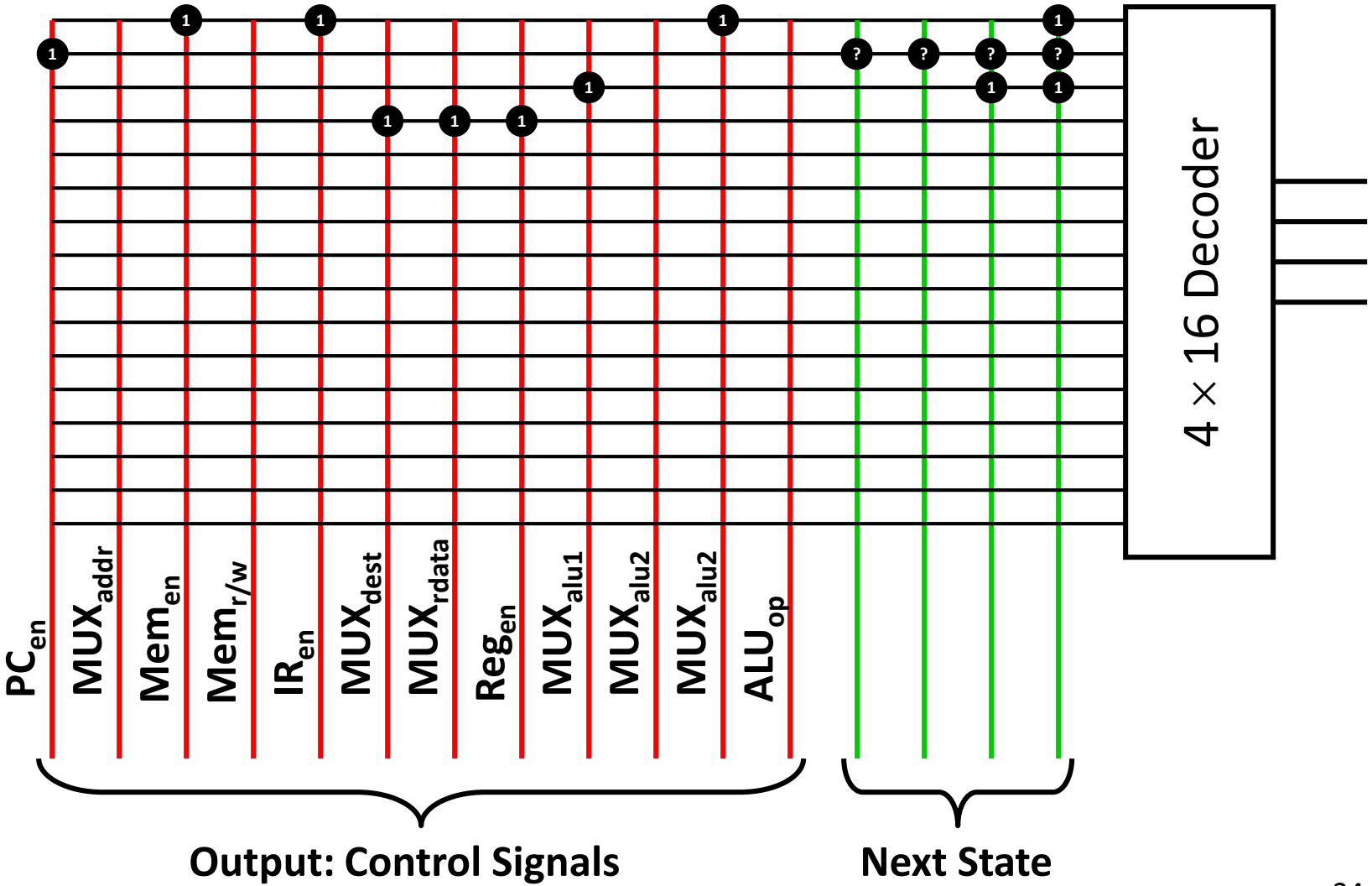


Add Cycle 4 Operation

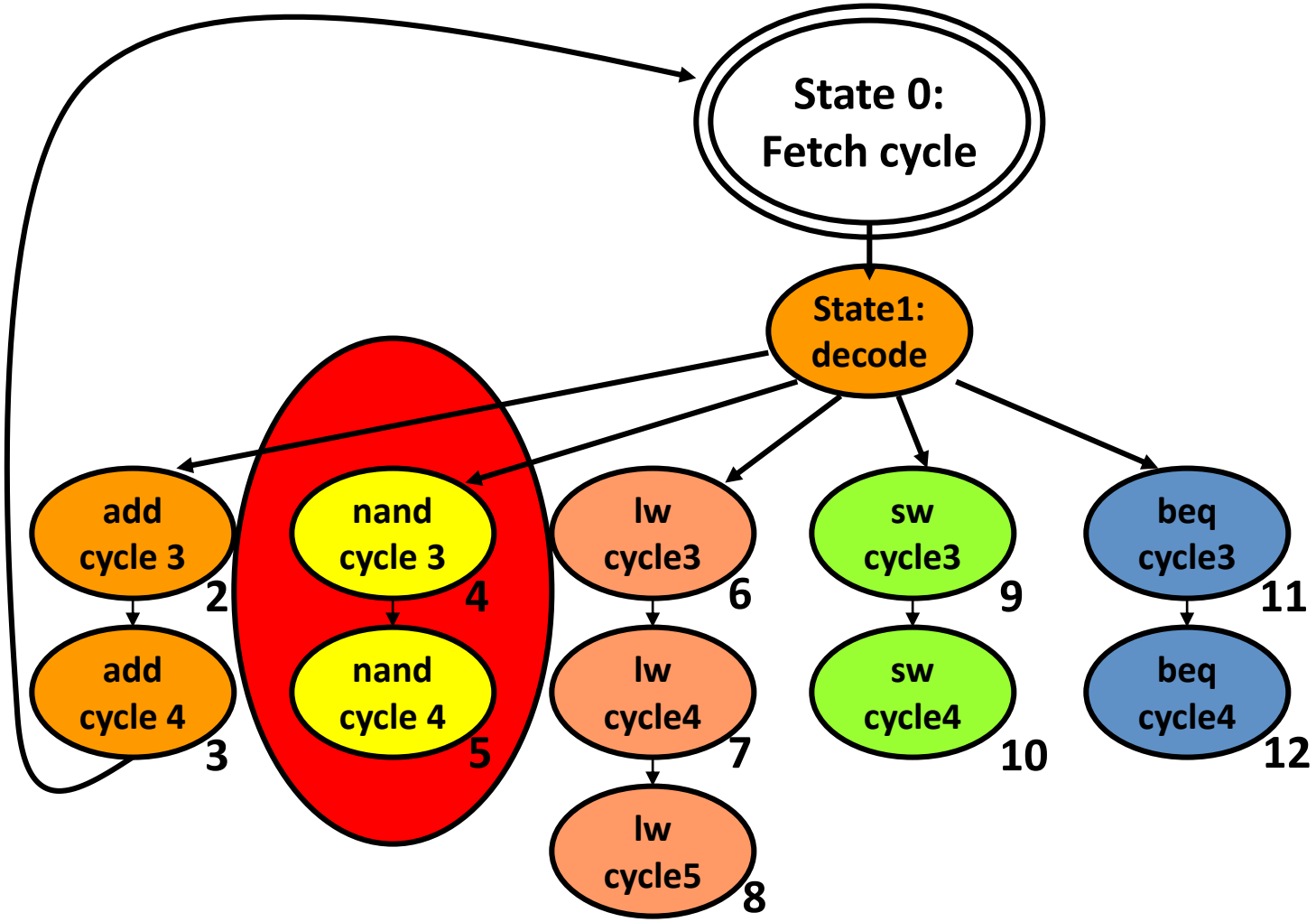
Send control signal to address MUX to select dest and to data MUX to select ALU output, then send write enable to register file.



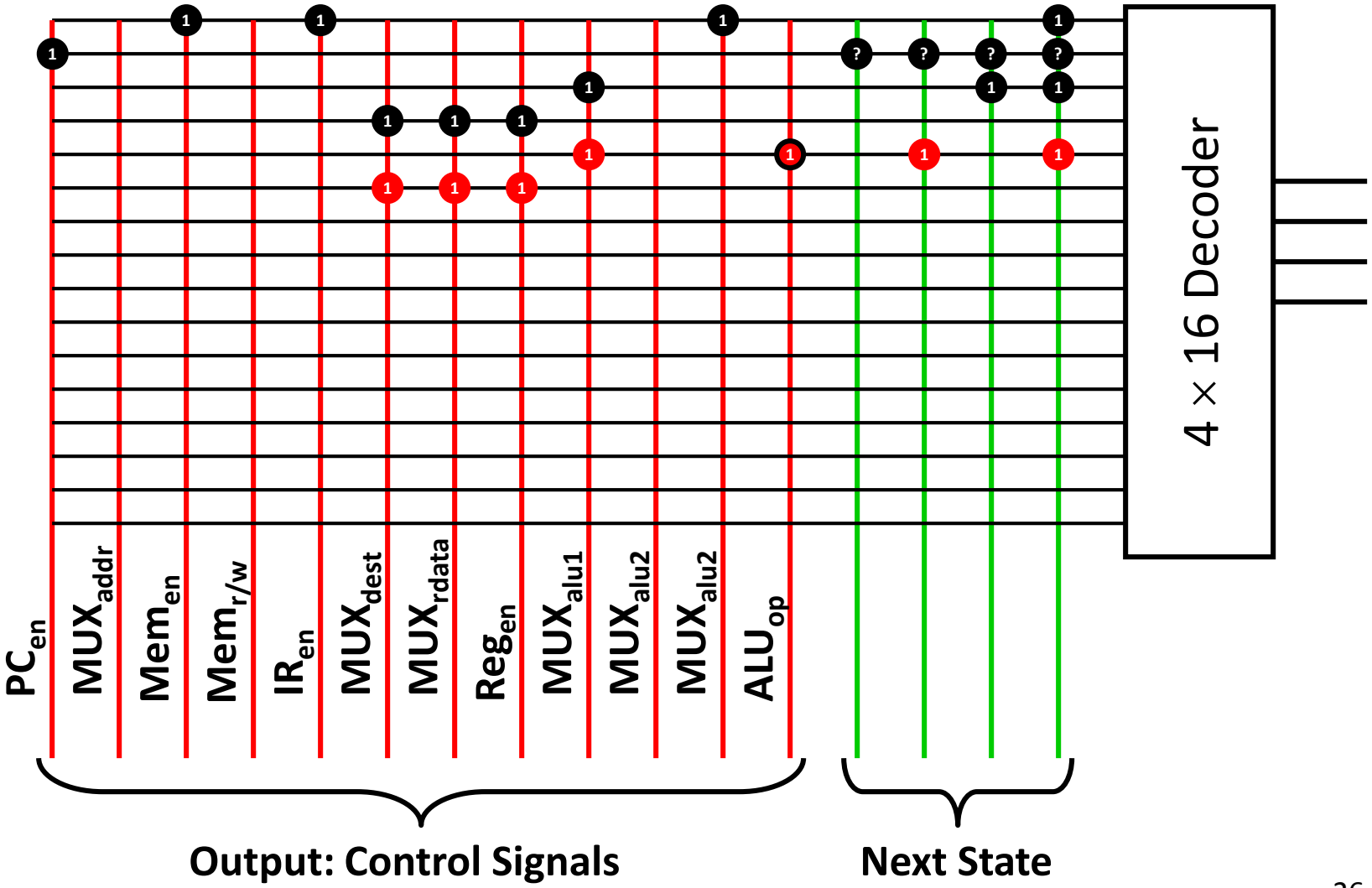
Building the Control Rom



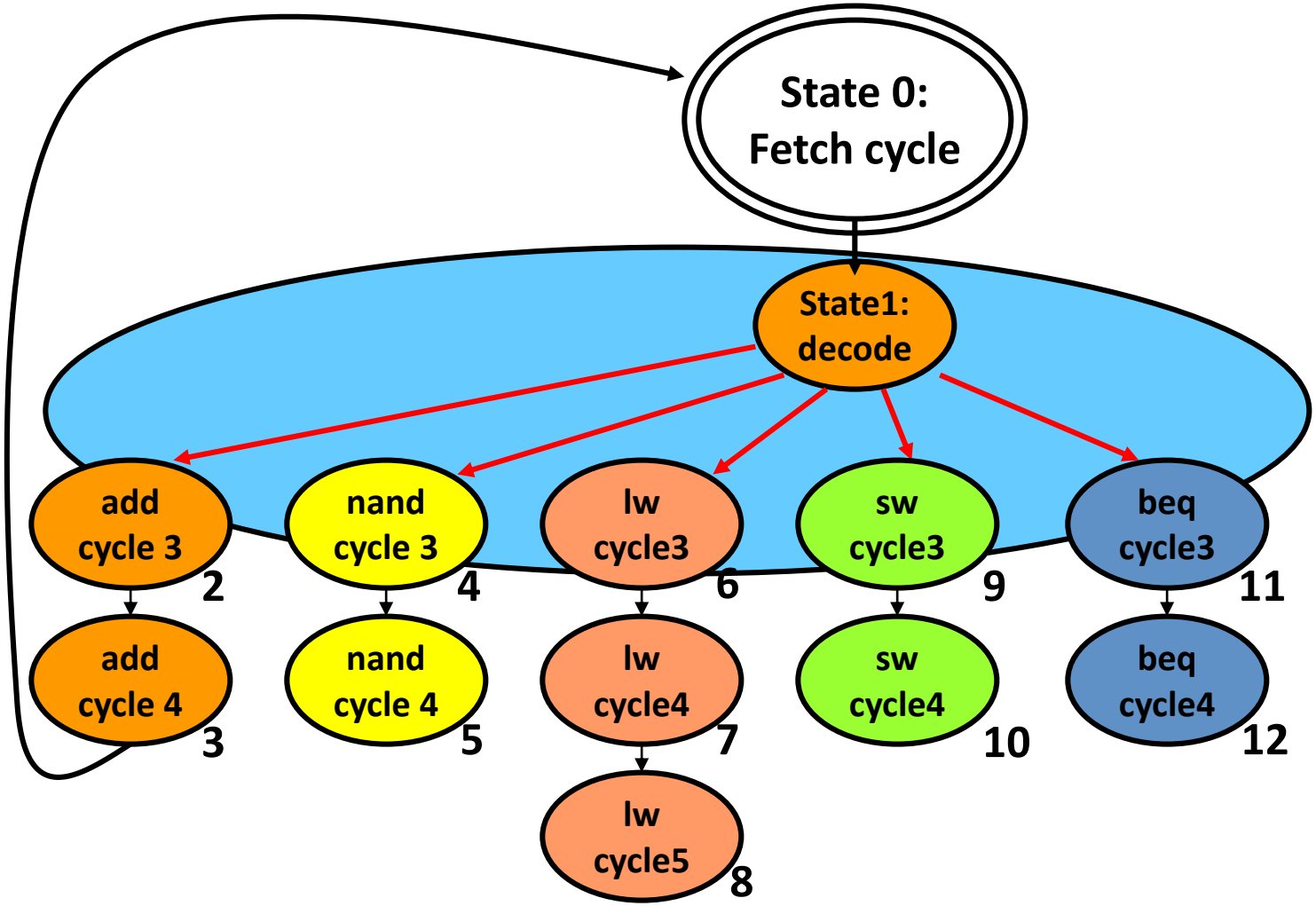
Return to State 0: Fetch cycle to execute the next instruction



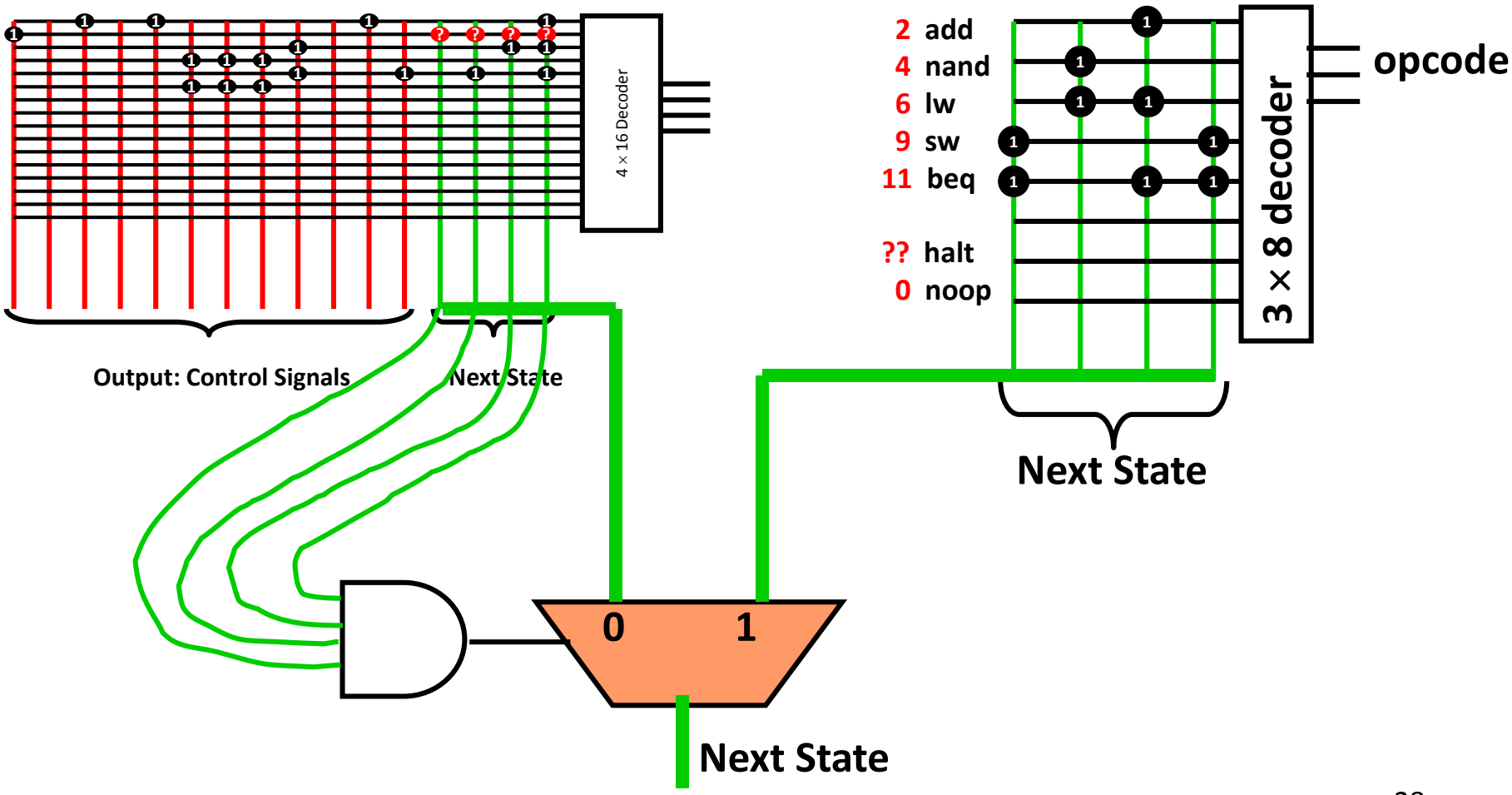
Control Rom for nand (4 and 5)



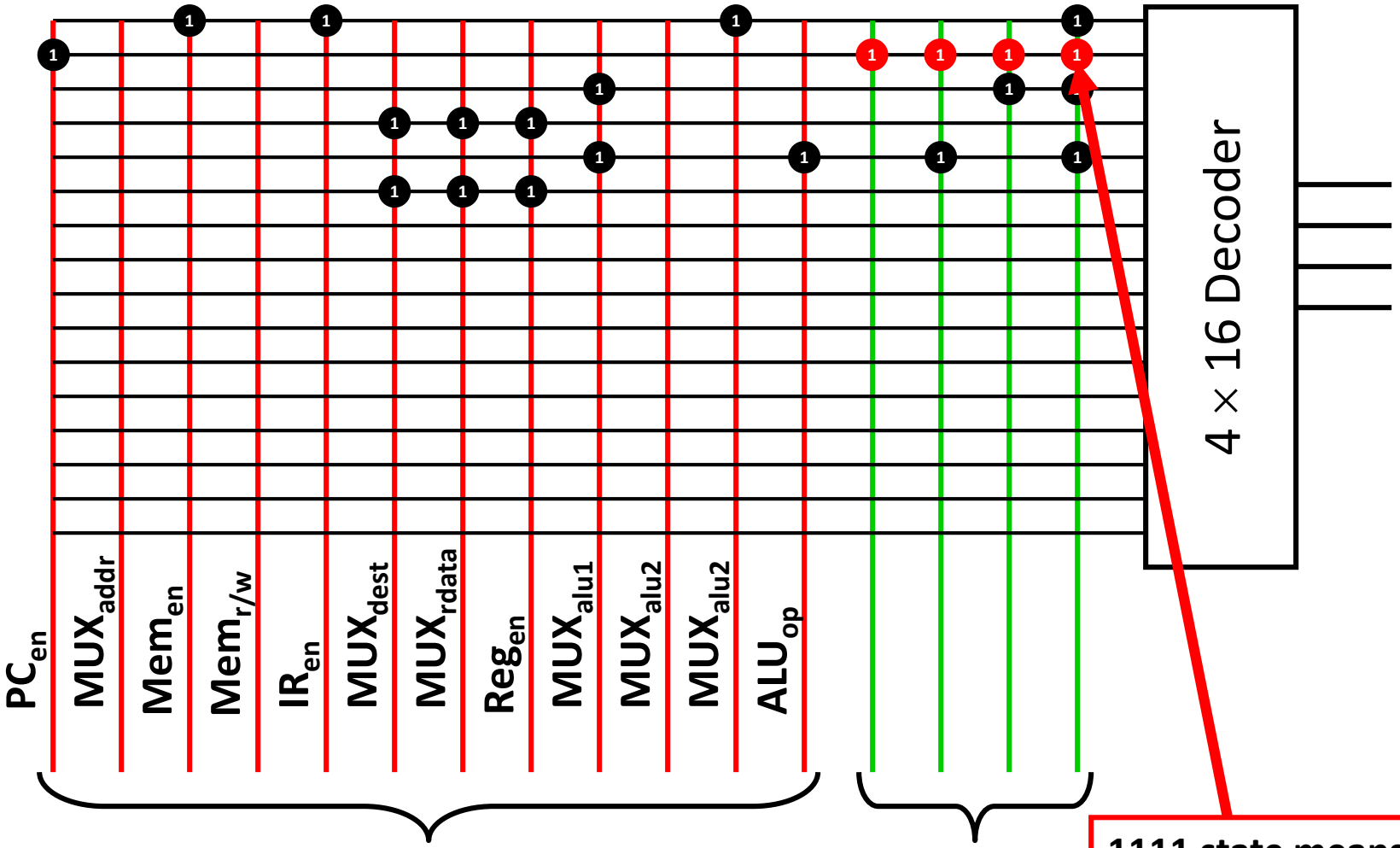
What about the transition from state 1?



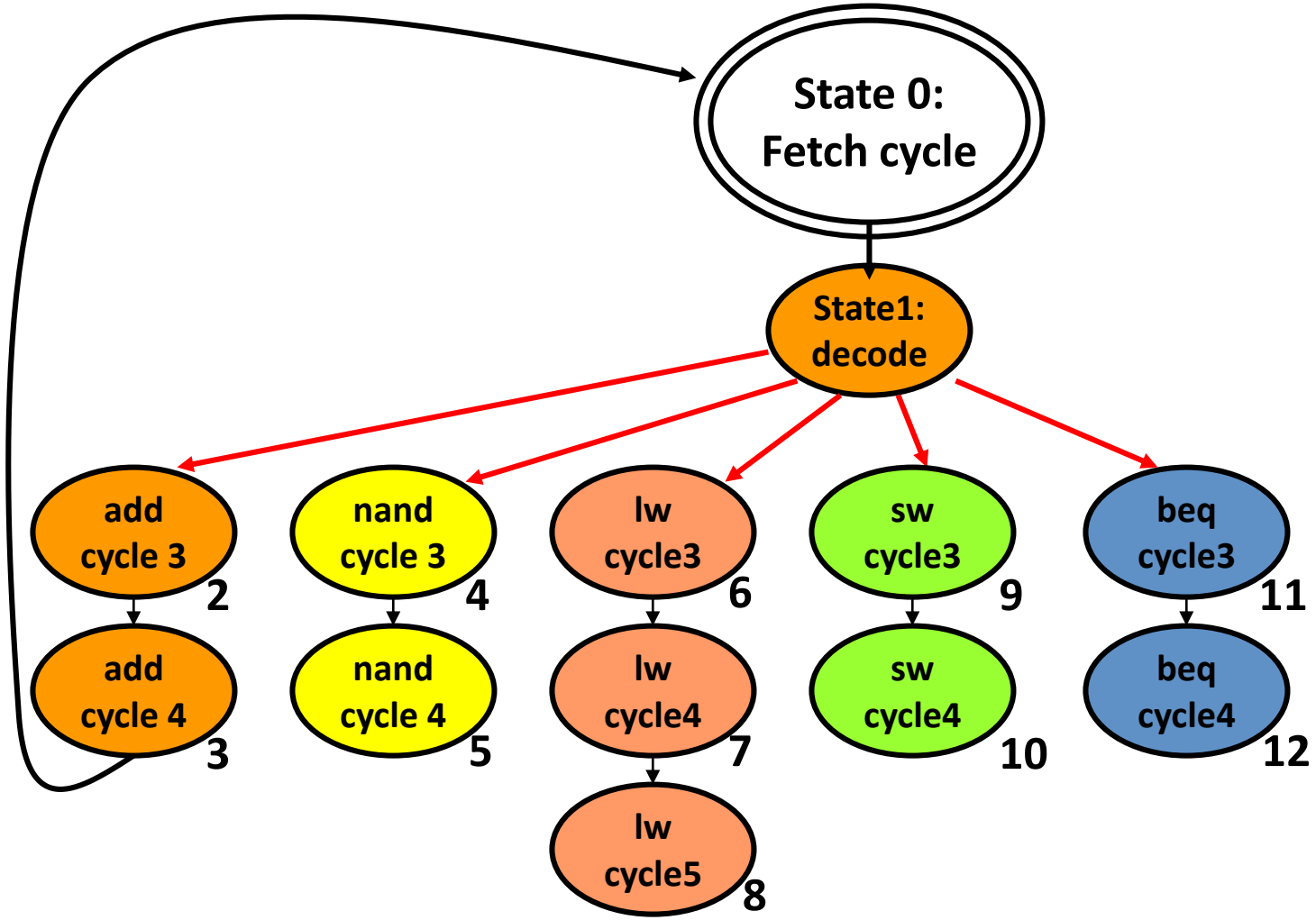
Complete transition function circuit



Control Rom (use of 1111 state)

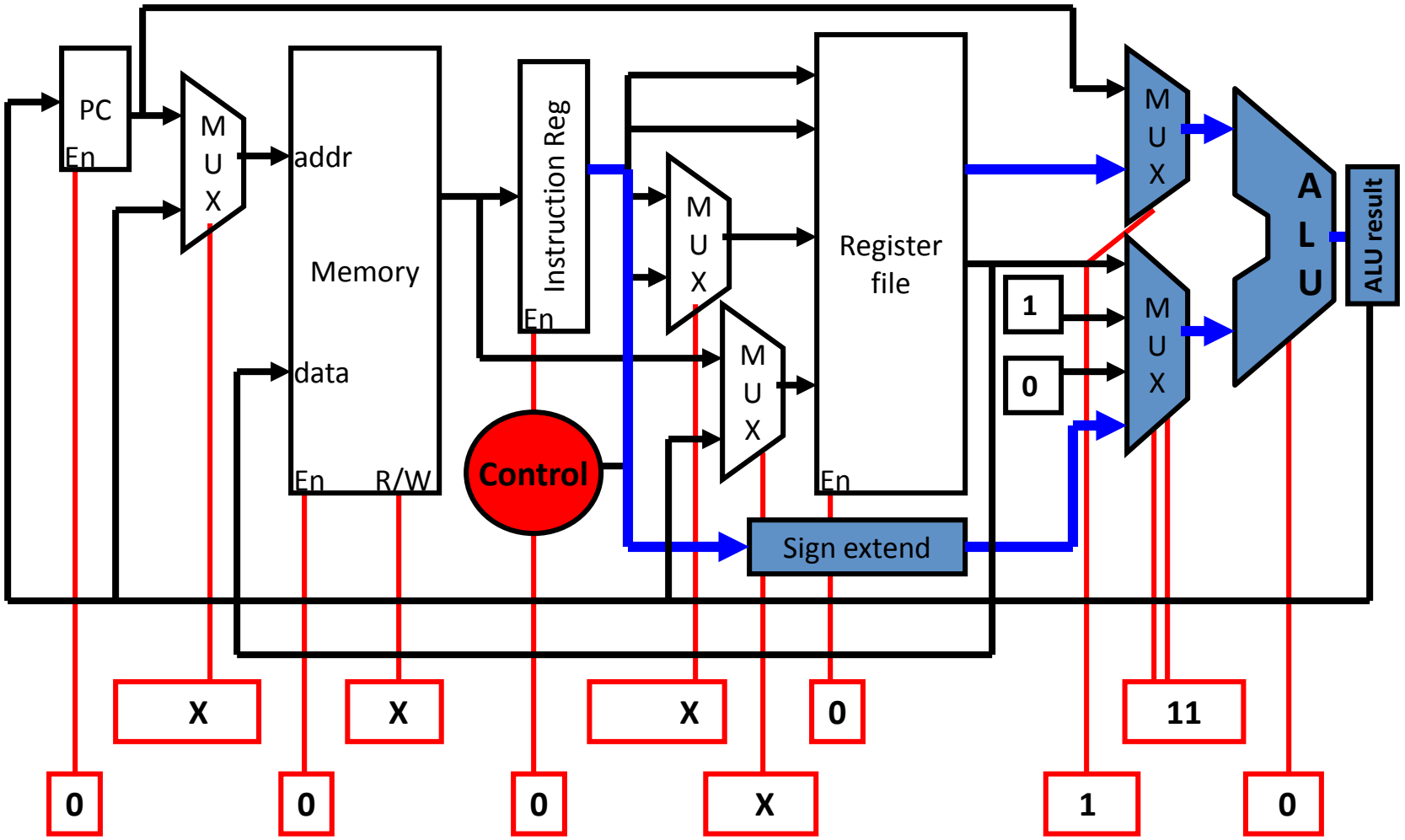


Return to State 0: Fetch cycle to execute the next instruction

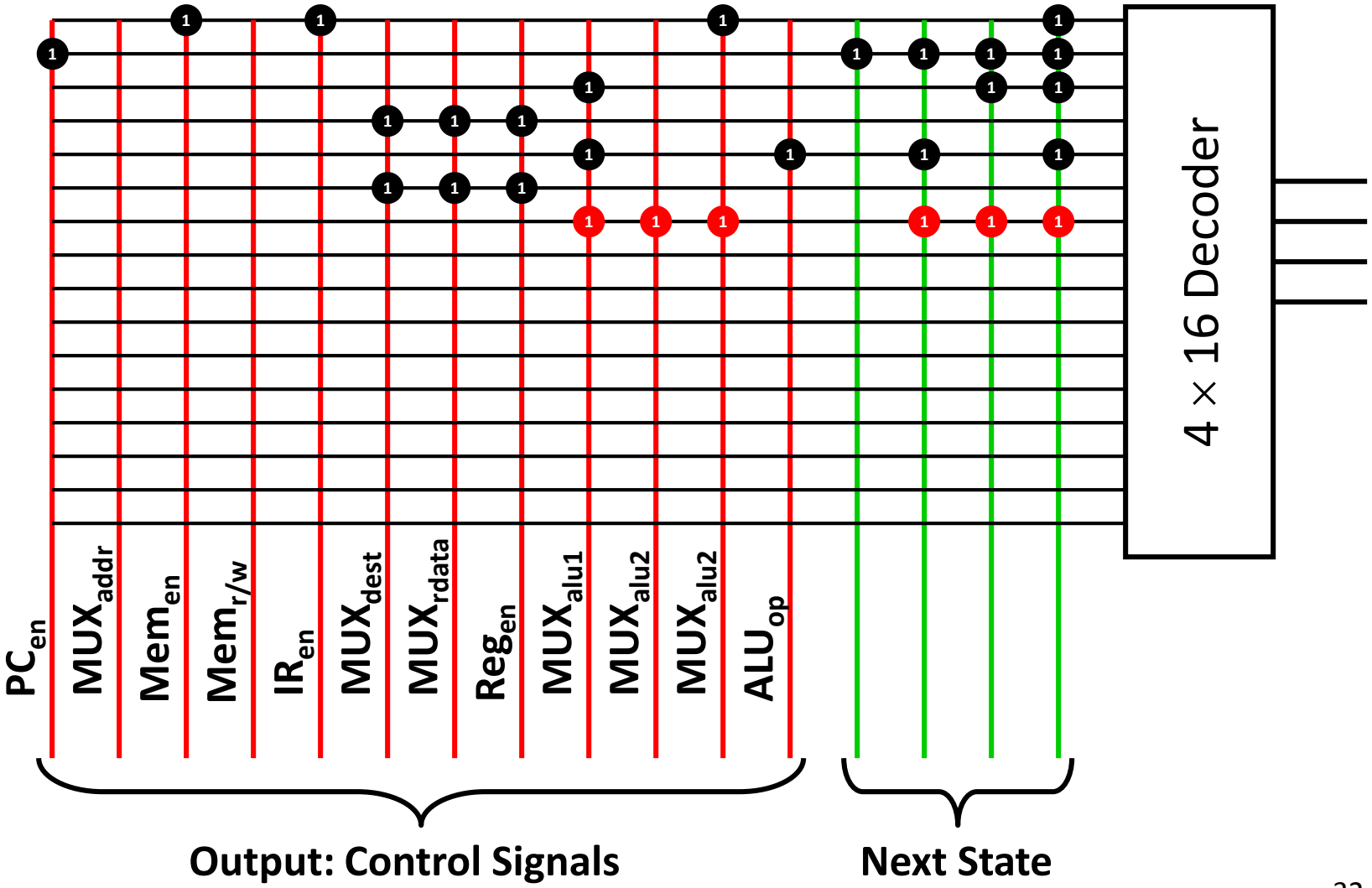


State 6: LW cycle 3

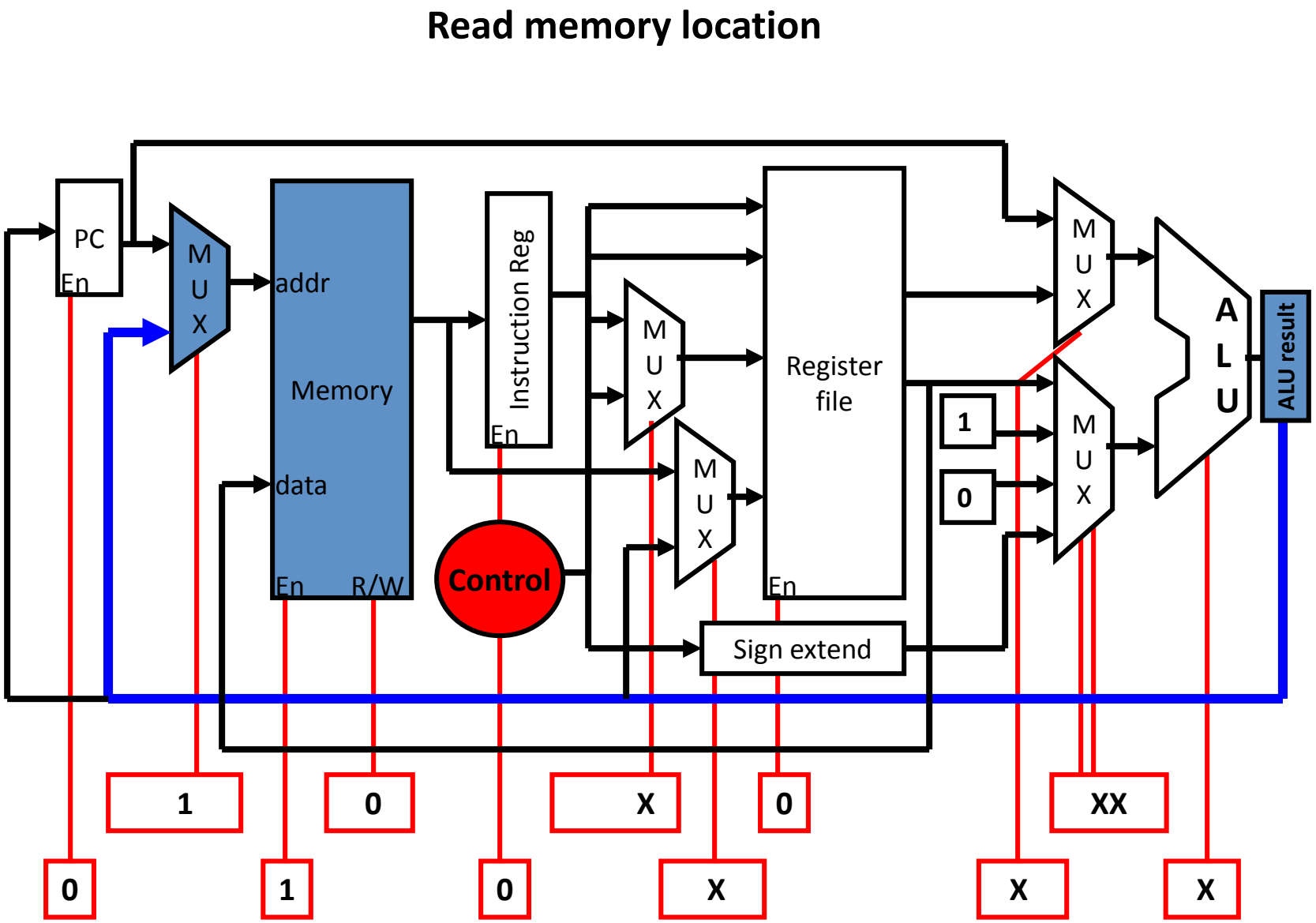
Calculate address for memory reference



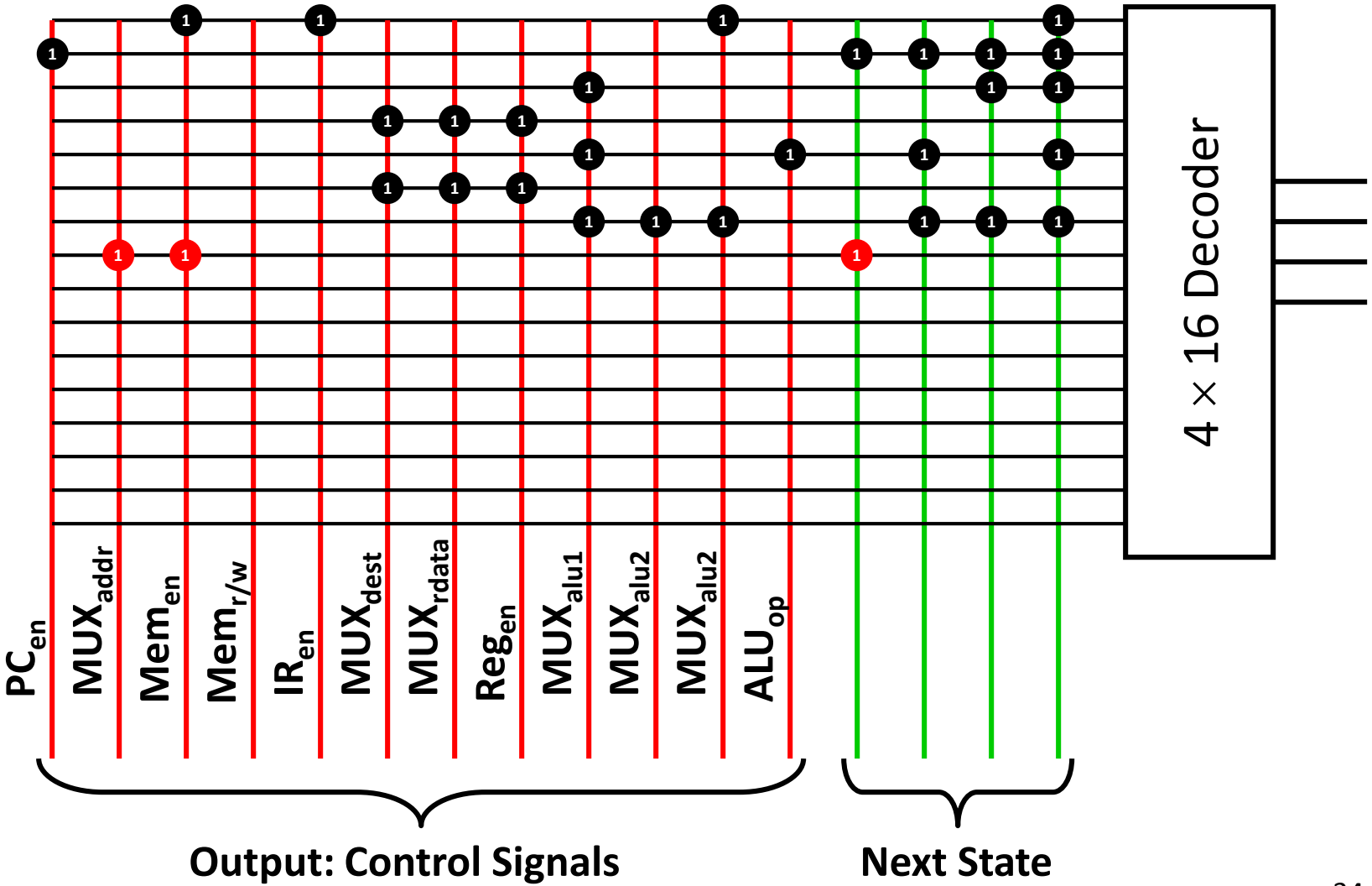
Control Rom (lw cycle 3)



State 7: LW cycle 4

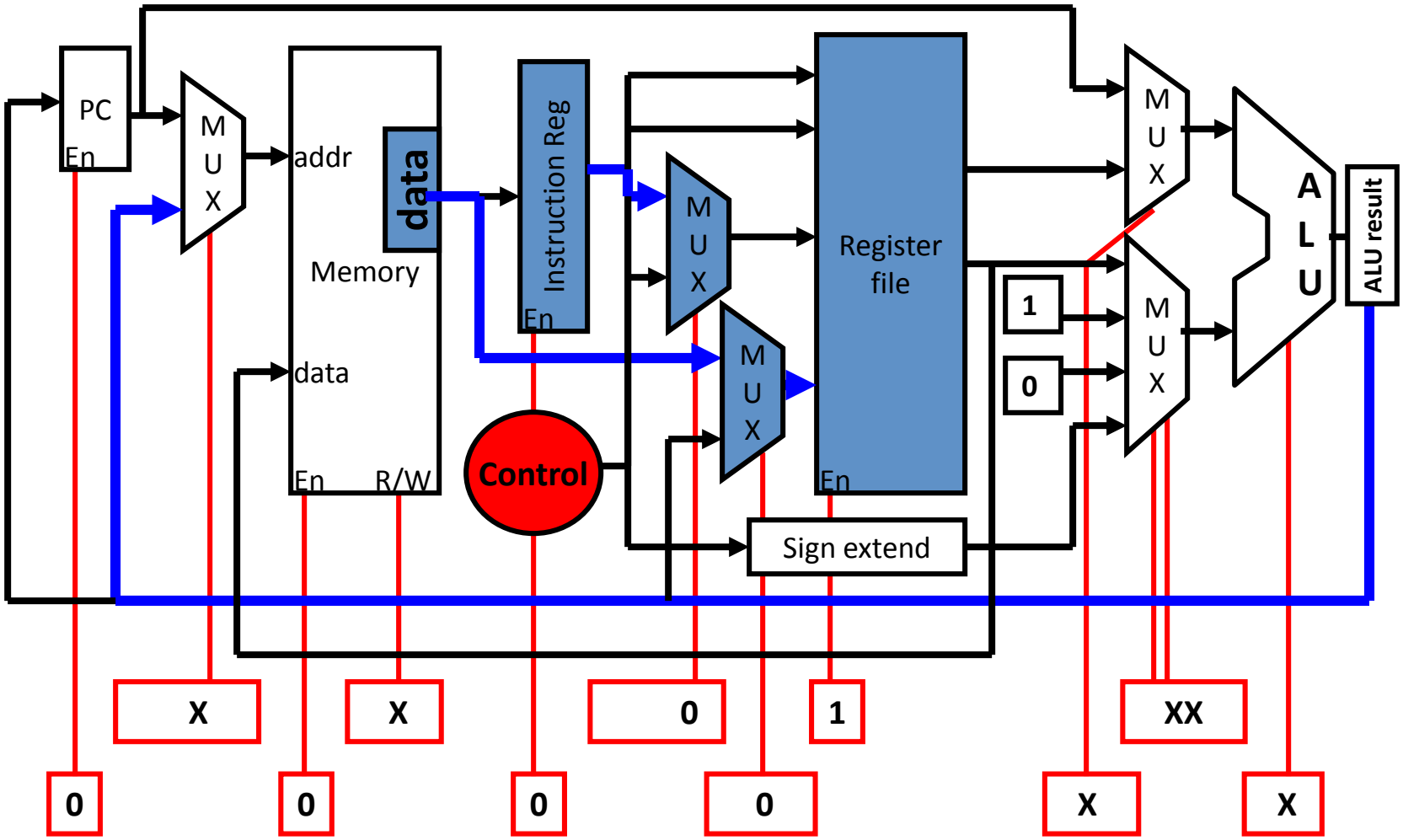


Control Rom (lw cycle 4)

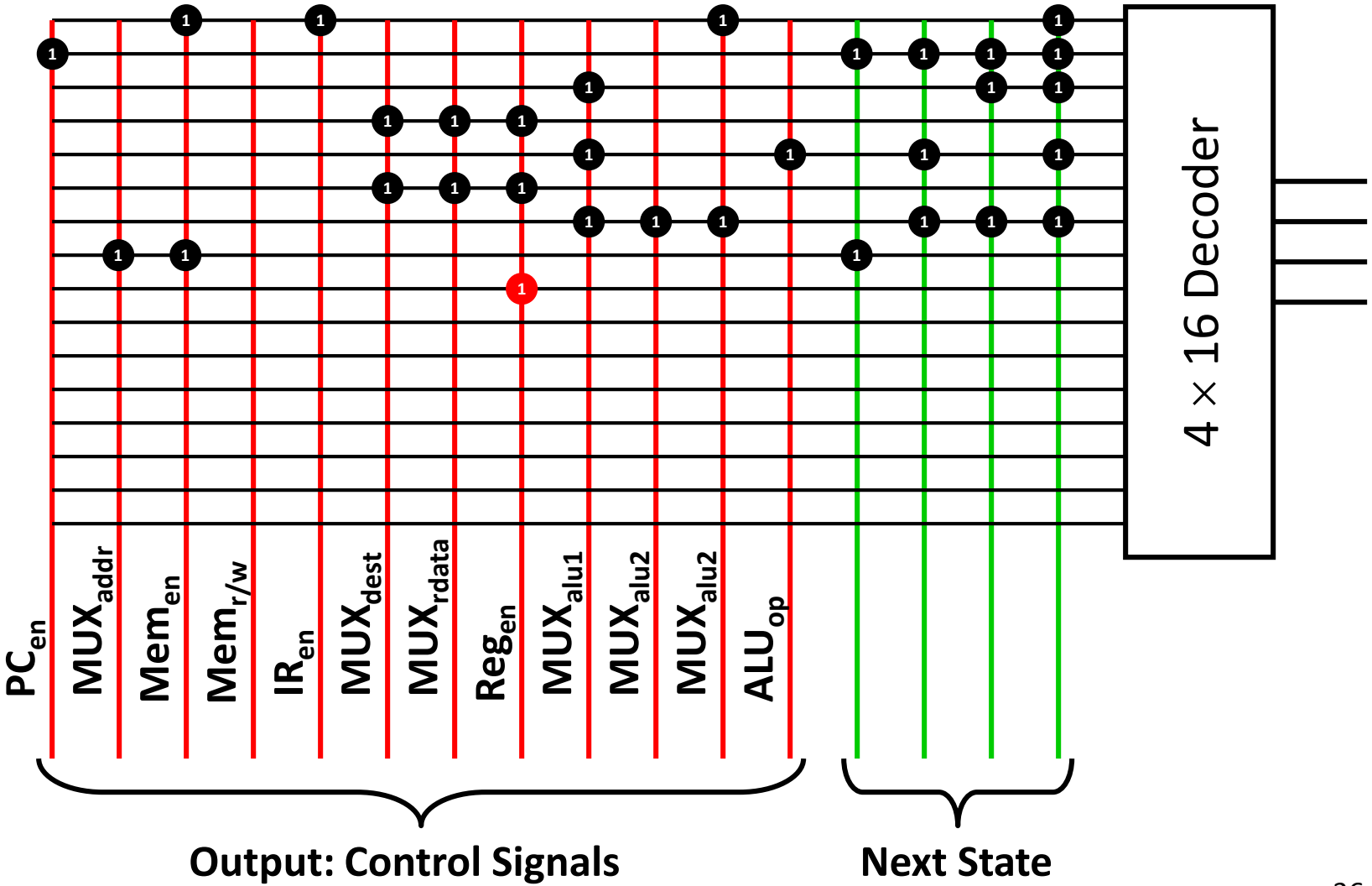


State 8: LW cycle 5

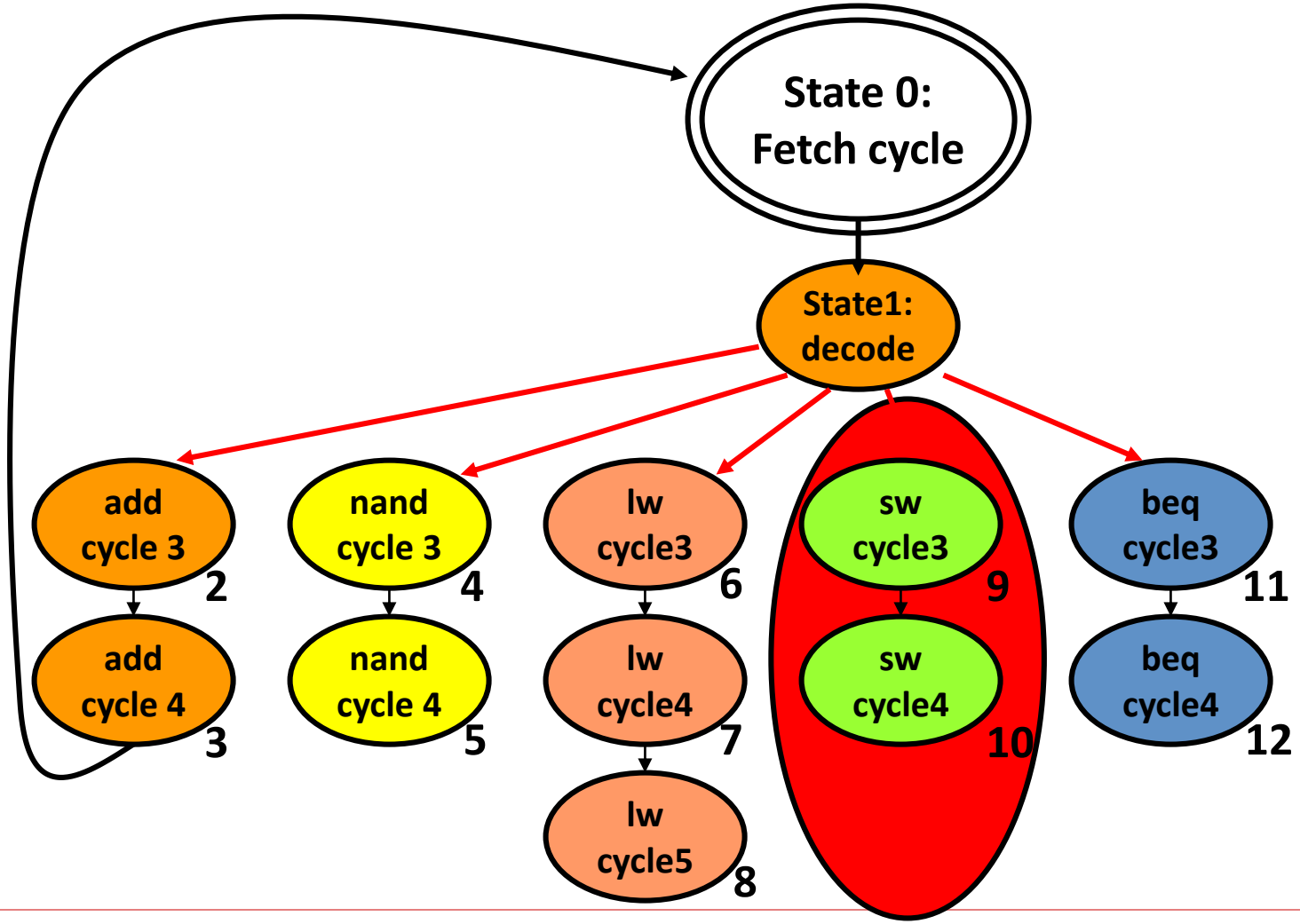
Write memory value to register file



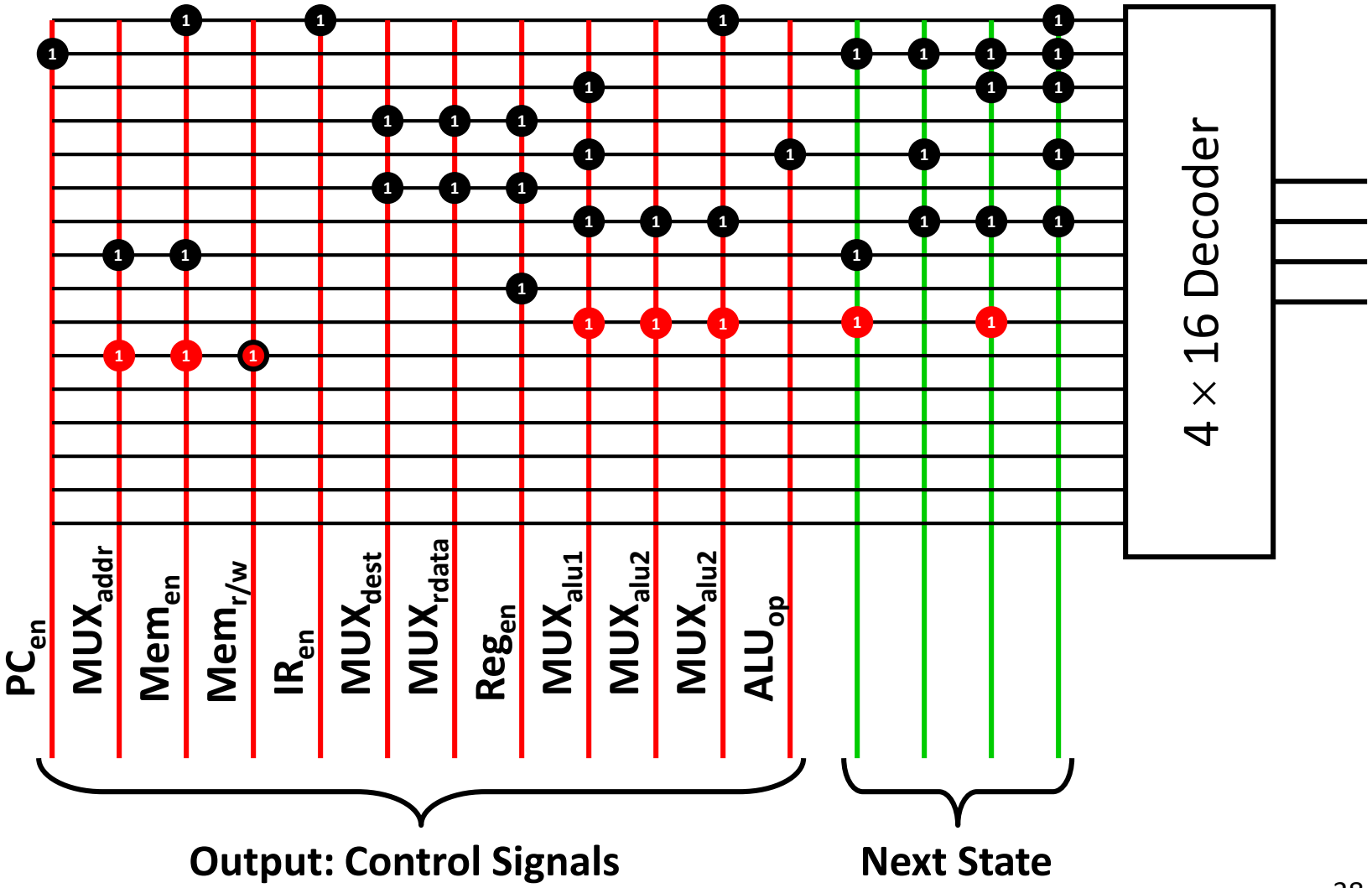
Control Rom (lw cycle 5)



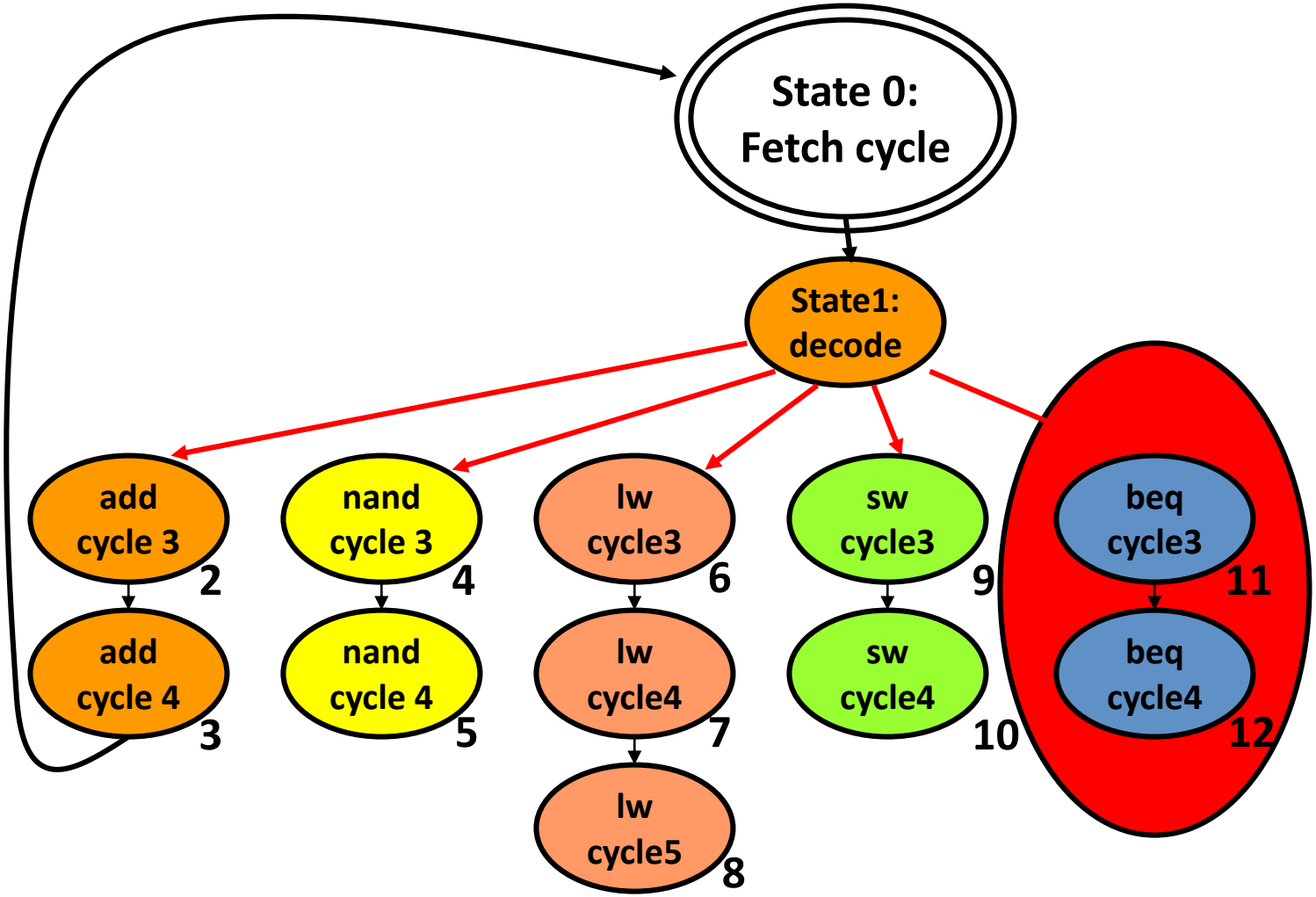
Return to State 0: Fetch cycle to execute the next instruction



Control Rom (sw cycles 3 and 4)

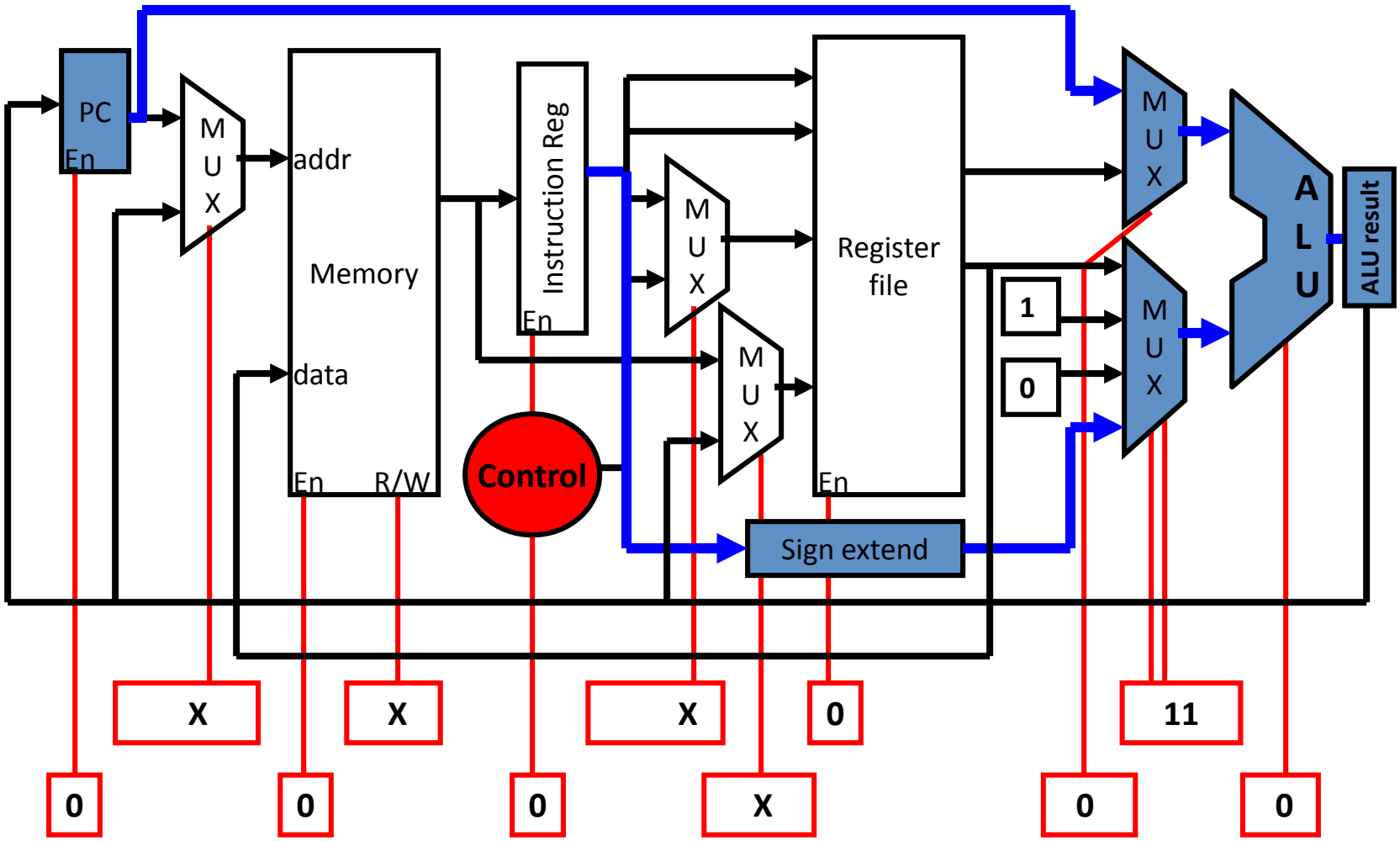


Return to State 0: Fetch cycle to execute the next instruction

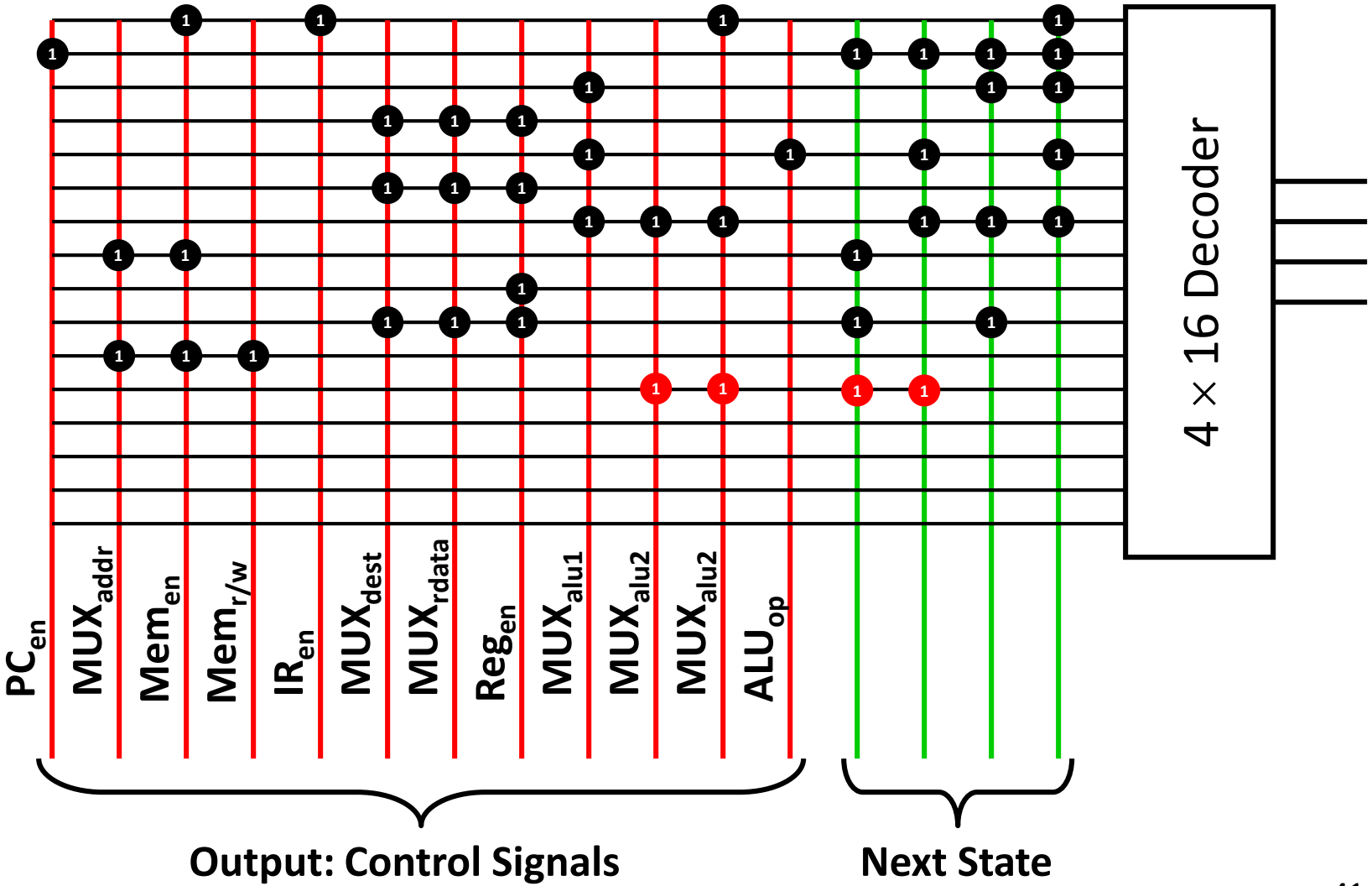


State 11: beq cycle 3

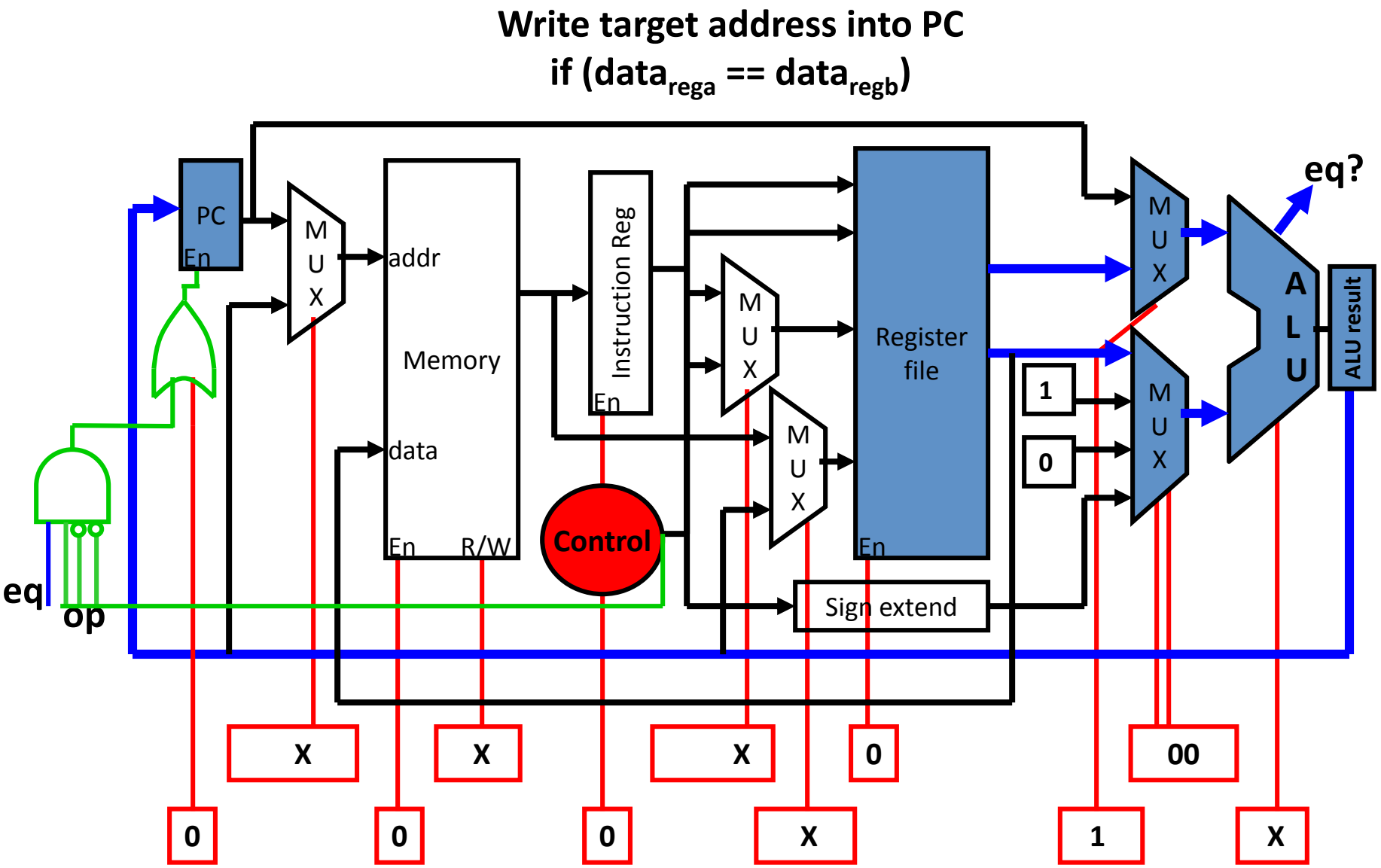
Calculate target address for branch



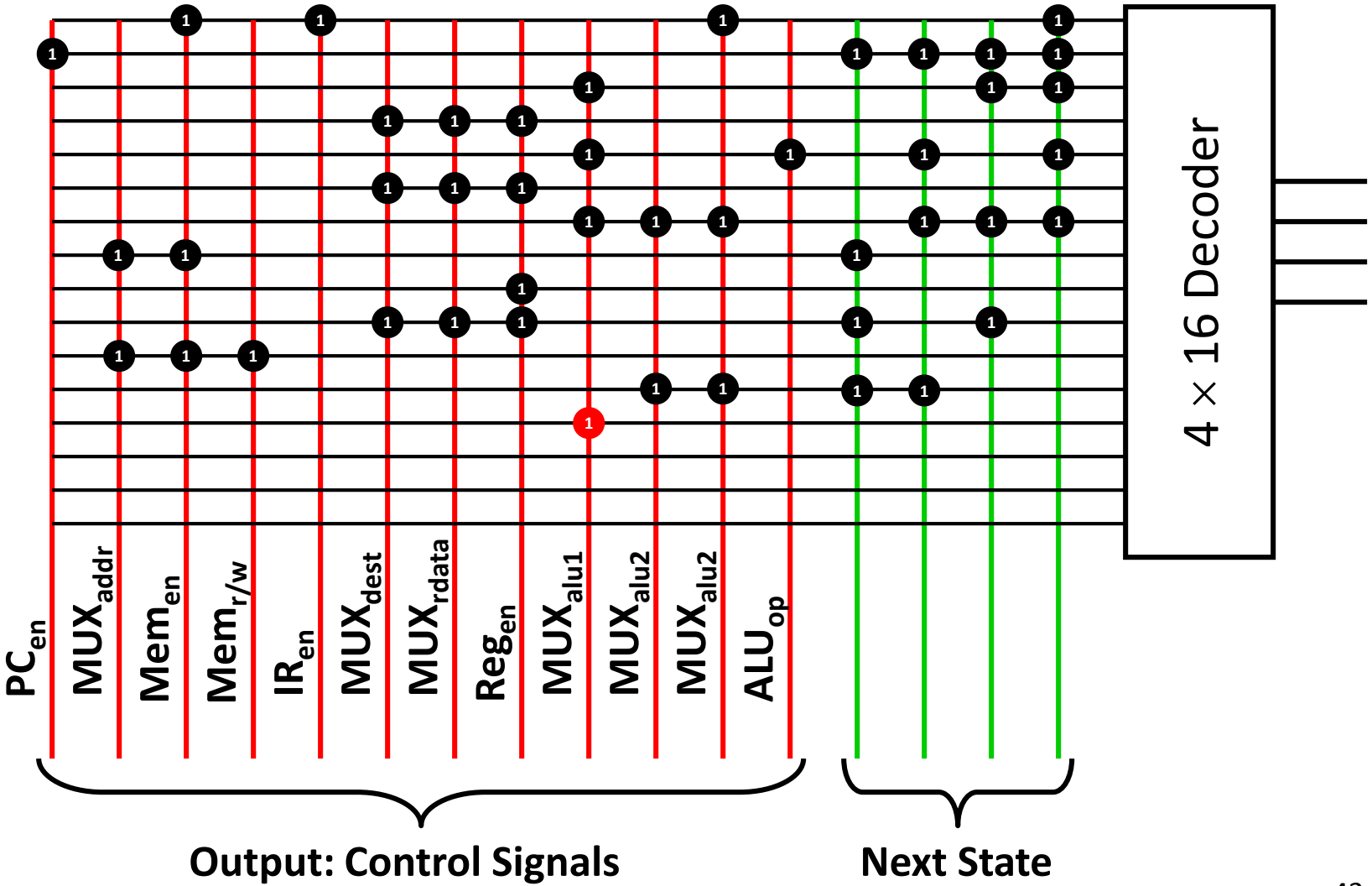
Control Rom (beq cycle 3)



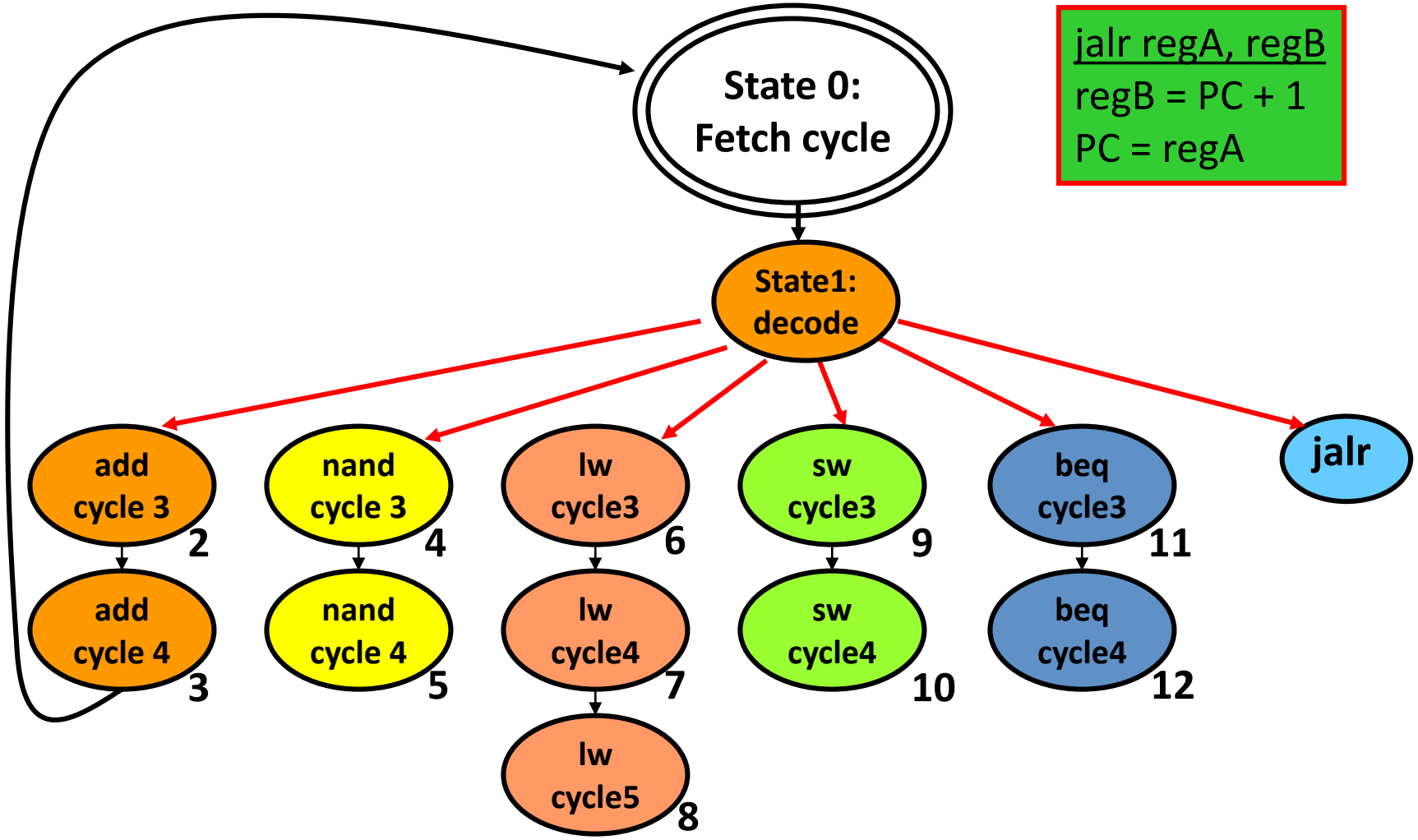
State 12: beq cycle 4



Control Rom (beq cycle 4)



OK, what about the JALR instruction?



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

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, is the example on the previous slide faster on the SC or MC design?

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