ELC4438: Embedded System Design

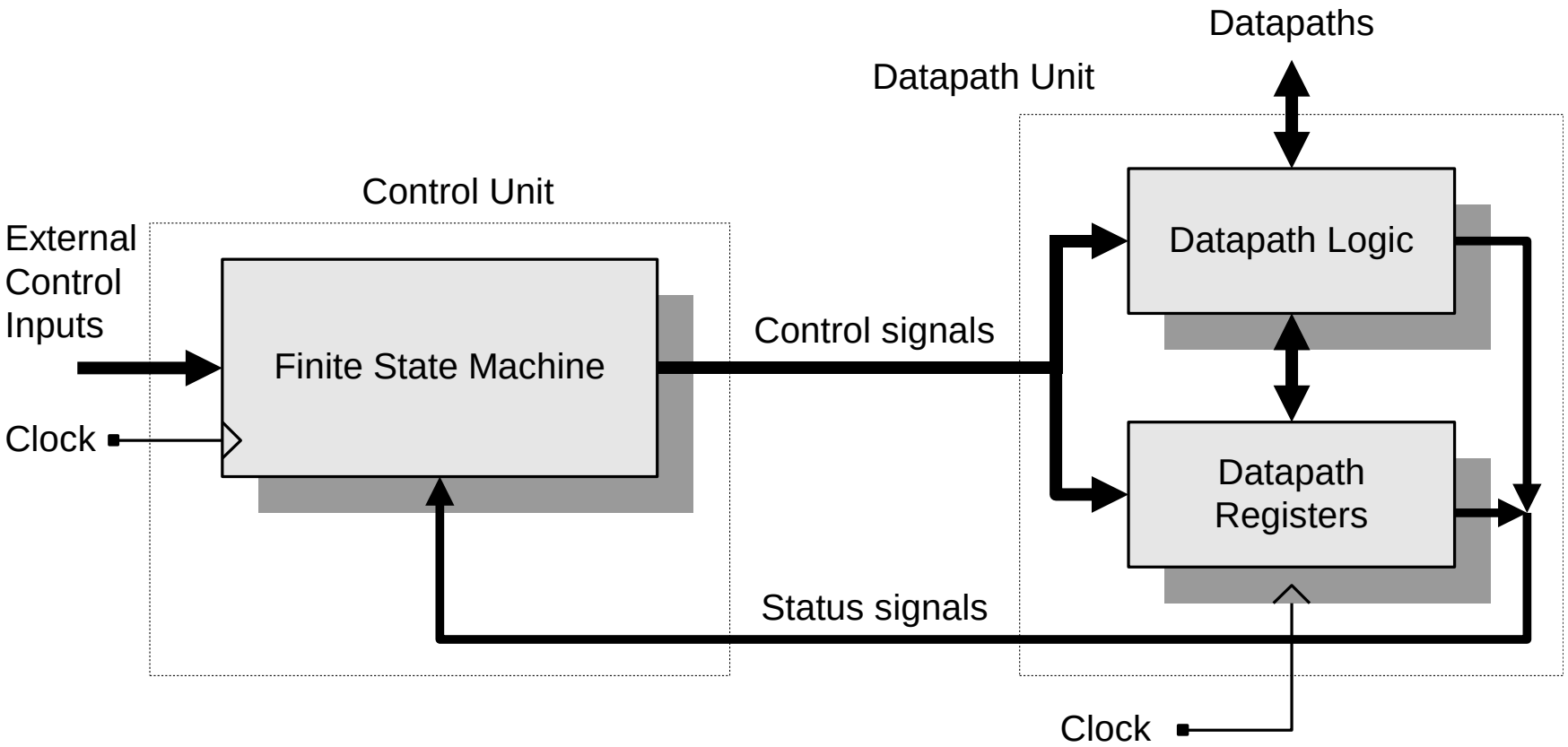
Finite State Machine for RISC SPM

Liang Dong

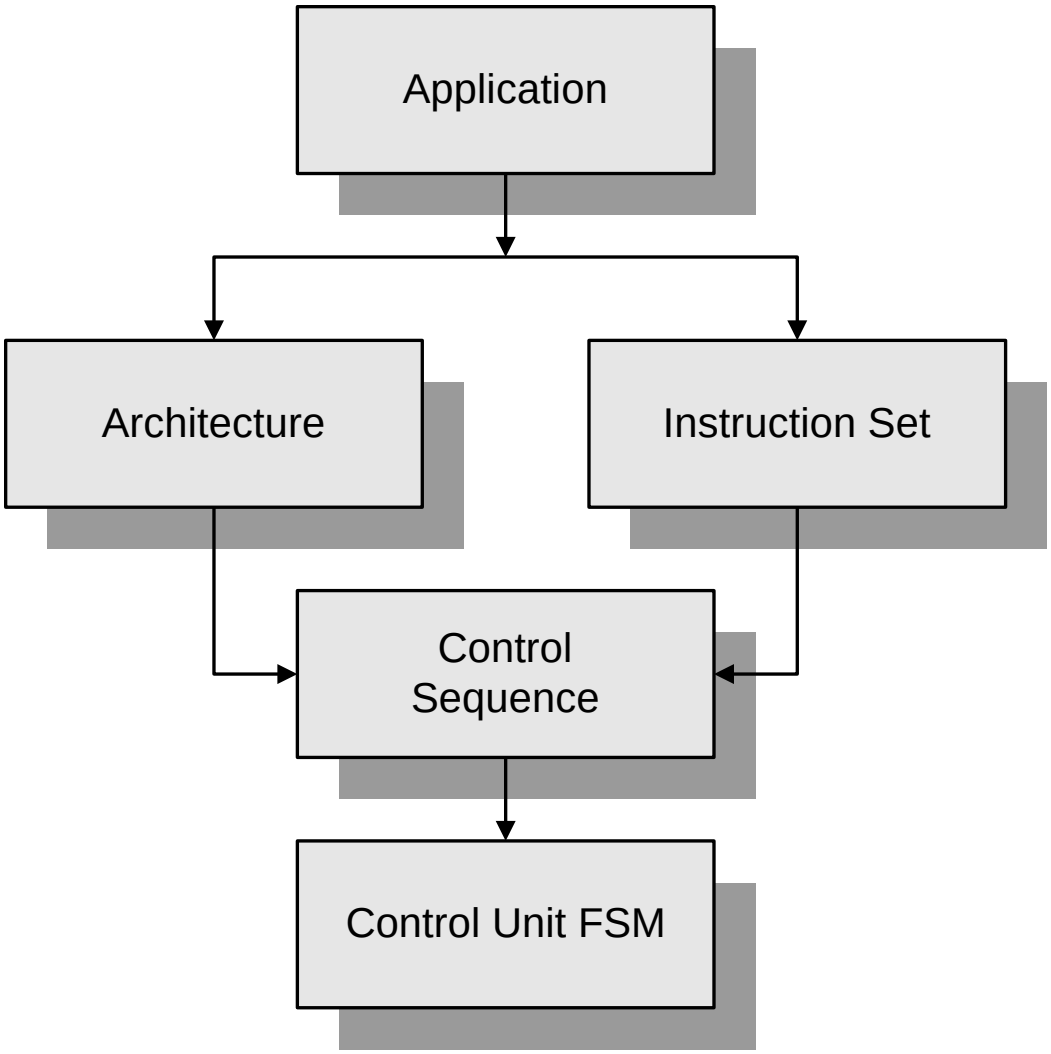
Electrical and Computer Engineering

Baylor University

Partitioned Sequential Machine



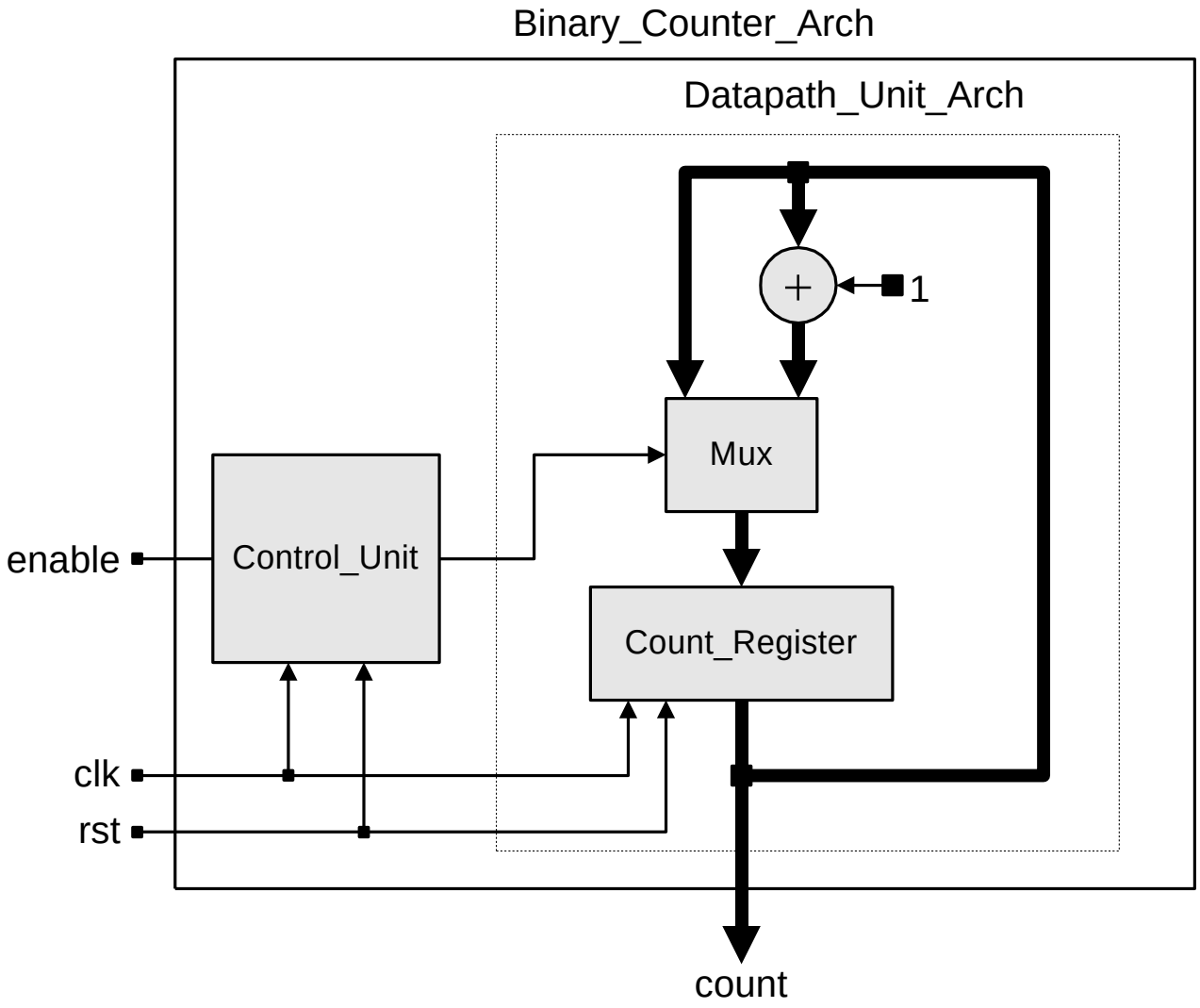
Application-Driven Architecture



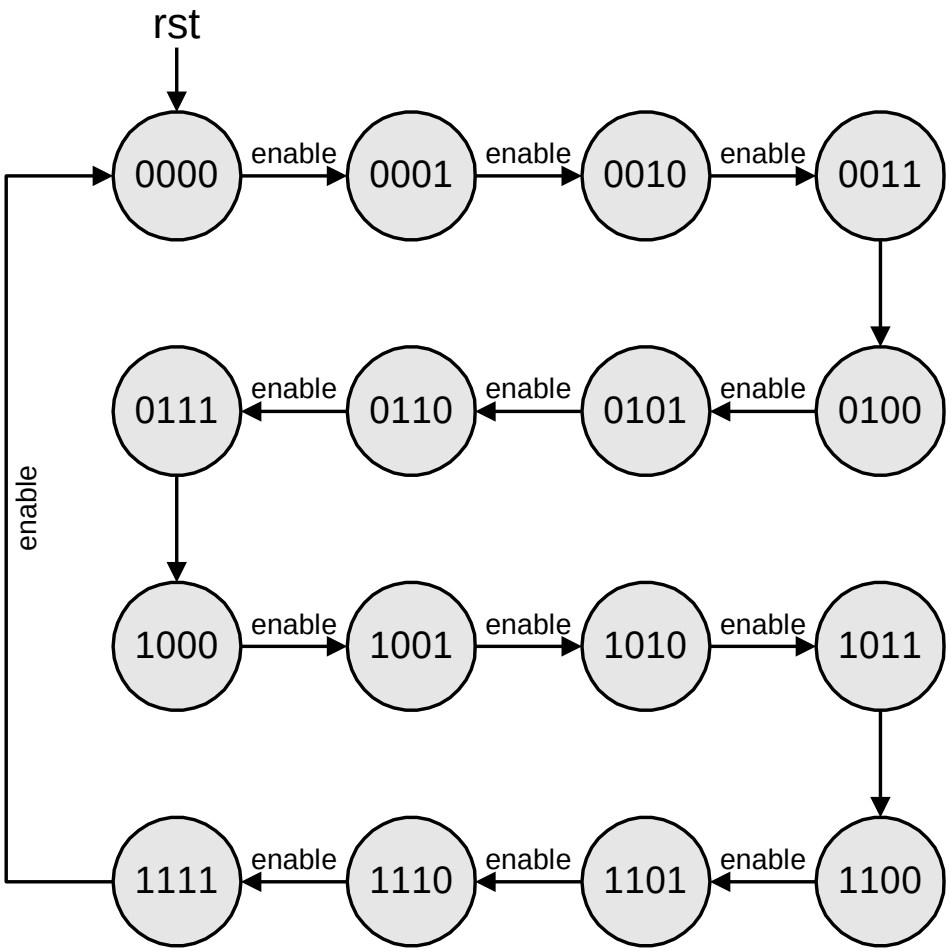
Design Example – Binary Counter

- Synchronous 4-bit binary counter
- Incremented by a count of 1 at each active edge of the clock
  - *enable* must be asserted for counting to occur
  - *rst* overrides all activity and drives the count to a value of *0000*.
- Wrap count to 0 when the count reaches 1111_2.
- Functional elements of the architecture of the datapath unit:
  - 4-bit register to hold *count*,
  - mux that steers either *count* or the sum of *count* and *1* to the input of the register
  - a 4-bit adder to increment *count*

Binary Counter Architecture



Alternative Counter Design - Explicit Finite State Machine

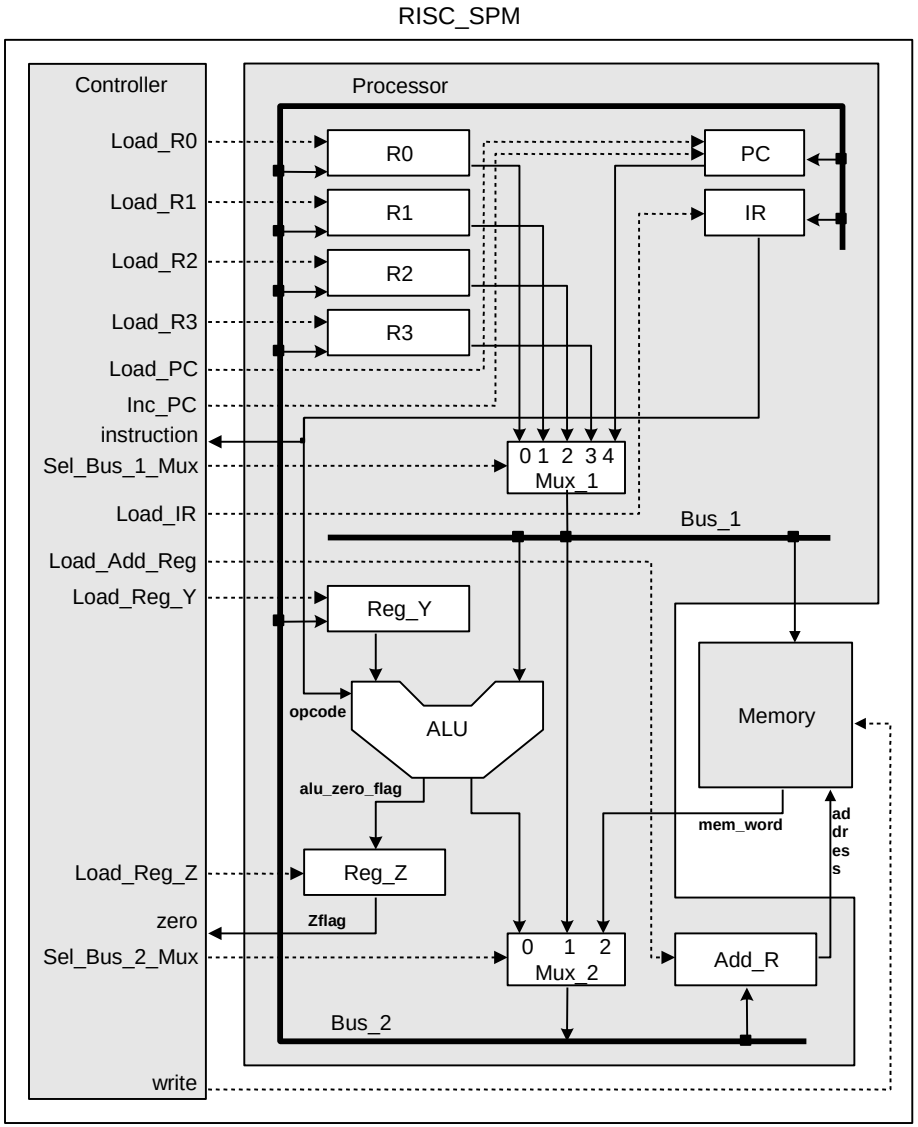


- Can this grow to larger counters?

Project Control

- Project is application oriented
  - Datapath and functional units designed to meet needs of instruction set
- Reduced complexity by splitting control / datapath.
- FSM allows for easy addition of operations
  - Allows use of single functional unit for multiple operations
- Use counters with FSM to reduce number of states
  - 1 Add state – use counter to determine when operation done

RISC Stored Program Machine (SPM)



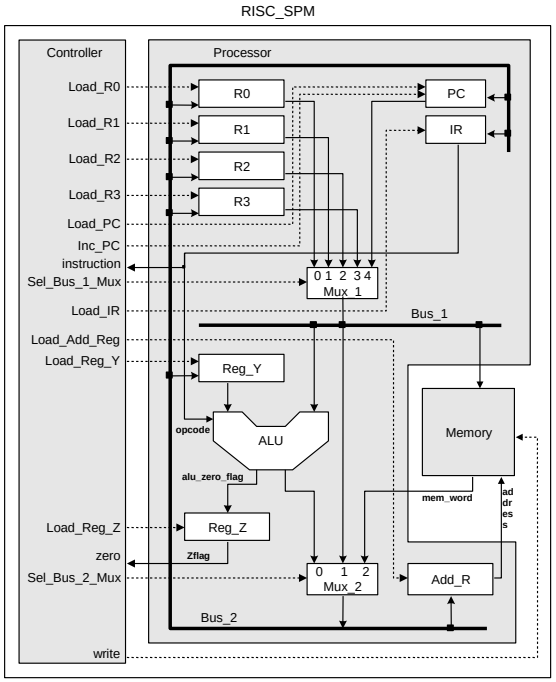
Instruction Sequence

- Fetch instruction from memory
- Decode instruction and fetch operands
- Execute instruction
  - ALU operations
  - Update storage registers
  - Update program counter (PC)
  - Update the instruction register (IR)
  - Update the address register (ADD_R)
  - Update memory
  - Control datapaths

Control Functions

- Functions of the control unit:
  - Determine when to load registers
  - Select the path of data through the multiplexers
  - Determine when data should be written to memory
  - Control the three-state busses in the architecture.

Control Signals



Control Signal	**Action**
Load_Add_Reg	Loads the address register
Load _PC	Loads *Bus_2* to the program counter
Load_IR	Loads *Bus_2* to the instruction register
Inc_PC	Increments the program counter
Sel_Bus_1_Mux	Selects among the *Program_Counter*, *R0*, *R1*, *R2*, and *R3* to drive *Bus_1*
Sel_Bus_2_Mux	Selects among *Alu_out*, *Bus_1*, and memory to drive *Bus_2*
Load_R0	Loads general purpose register *R0*
Load_R1	Loads general purpose register *R1*
Load_R2	Loads general purpose register *R2*
Load_R3	Loads general purpose register *R3*
Load_Reg_Y	Loads *Bus_2* to the register *Reg_Y*
Load Reg_Z	Stores output of *ALU* in register *Reg_Z*
write	Loads *Bus_1* into the *SRAM* memory

RISC SPM: Instruction Set [1]

- The design of the controller depends on the processor's instruction set.
- RISC SPM has two types of instructions.
- Short Instruction – 8 bits (basic arithmetic)

opcode				source		destination	
0	0	1	0	0	1	1	0

- Long Instruction – 16 bits (accessing memory)

opcode				source		destination	
0	1	1	0	1	0	don't care	don't care
address							
0	0	0	1	1	1	0	1

RISC SPM: Instruction Set [2]

Instr	Instruction Word			Action
	opcode	src	dest	
NOP	0000	??	??	none
ADD	0001	src	dest	dest <= src + dest
SUB	0010	src	dest	dest <= dest - src
AND	0011	src	dest	dest <= src && dest
NOT	0100	src	dest	dest <= ~src
RD*	0101	??	dest	dest <= memory[Add_R]
WR*	0110	src	??	memory[Add_R] <= src
BR*	0111	??	??	PC <= memory[Add_R]
BRZ*	1000	??	??	PC <= memory[Add_R]
HALT	1111	??	??	Halts execution until reset

* Requires a second word of data; ? denotes a don't care.

Controller Design

- Three phases of operation: *fetch*, *decode*, and *execute*.
  - Fetching: Retrieves an instruction from memory (2 clock cycles)
  - Decoding: Decodes the instruction, manipulates datapaths, and loads registers (1 cycle)
  - Execution: Generates the results of the instruction (0, 1, or 2 cycles)

Controller States [1]

S_idle State entered after reset is asserted. No action.

S_fet1 Load the Add_R with the contents of the PC

S_fet2 Load the IR with the word addressed by the Add_R,
Increment the PC

S_dec Decode the IR
Assert signals to control datapaths and register transfers.

S_ex1 Execute the *ALU* operation for a single-byte instruction,
Conditionally assert the zero flag, Load the destination register

Controller States [2]

S_rd1 Load Add_R with the second byte of an RD instruction
Increment the PC.

S_rd2 Load the destination register with memory[Add_R]

S_wr1 Load Add_R with the second byte of a WR instruction,
Increment the PC.

S_wr2 Write memory[Add_R] with the source register

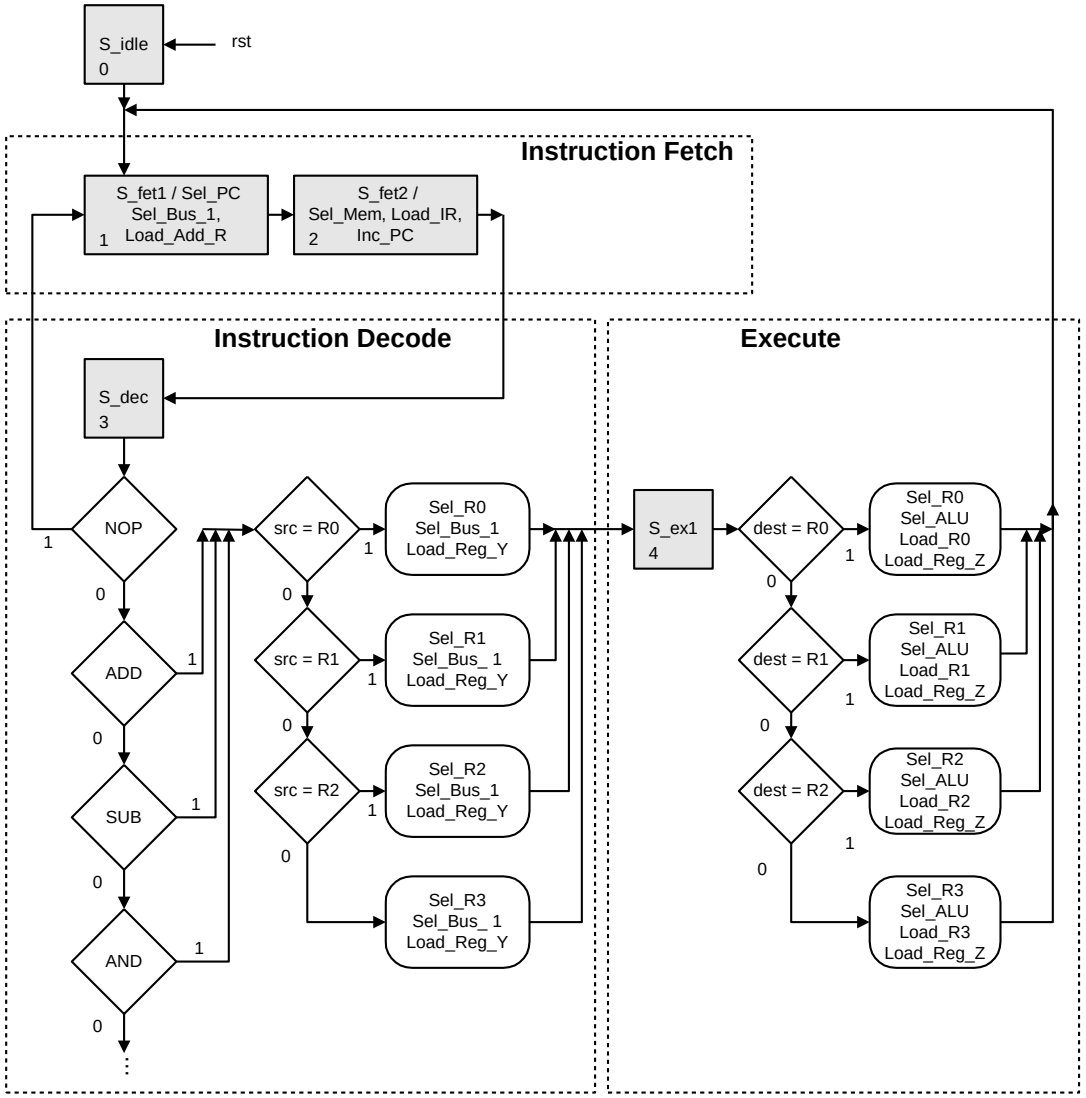
S_br1 Load Add_R with the second byte of a BR instruction
Increment the PC.

S_br2 Load the PC with the memory[Add_R]

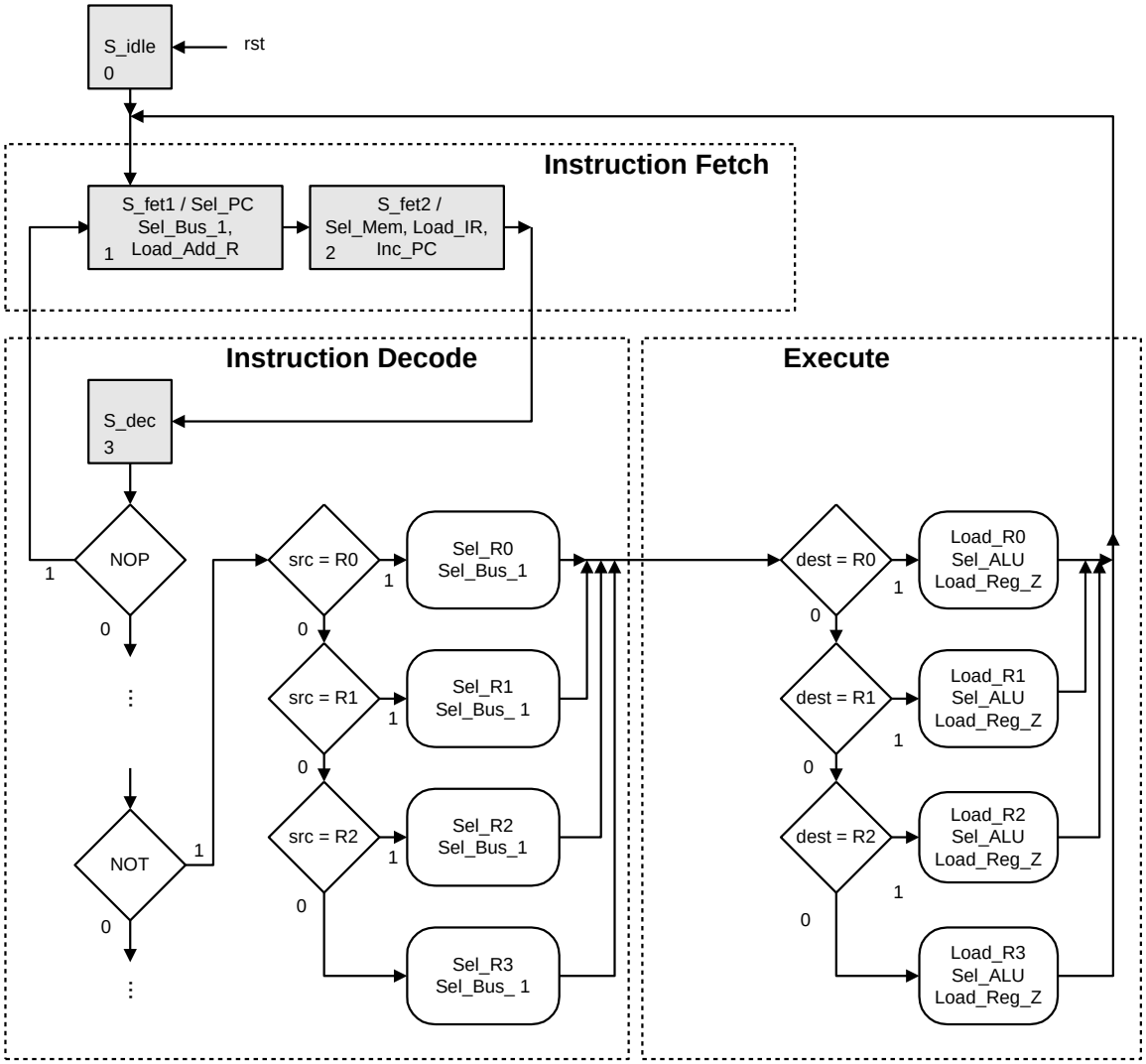
S_halt Default state to trap failure to decode a valid instruction

Which states are similar?

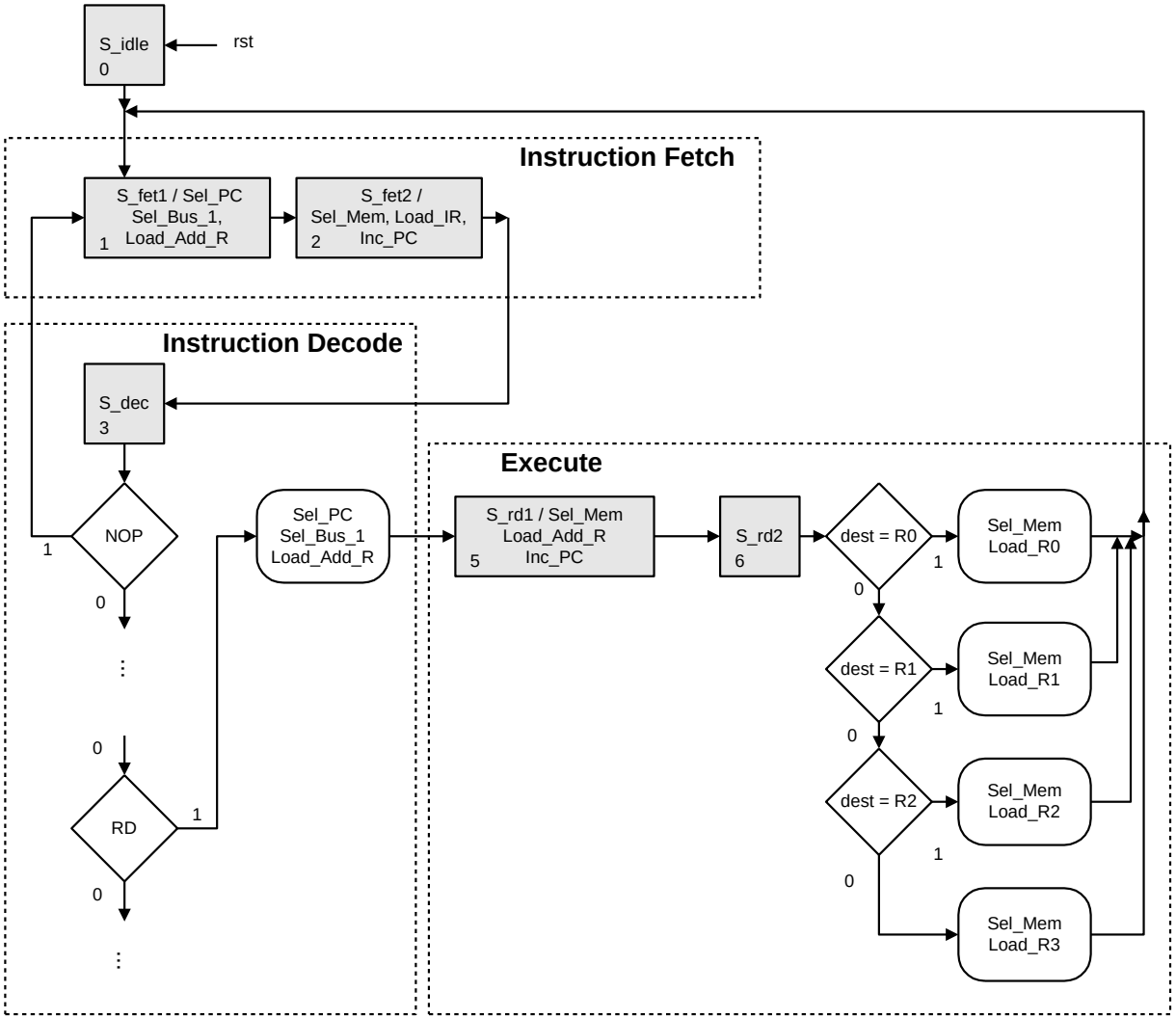
Controller ASM: NOP/ADD/SUB/AND



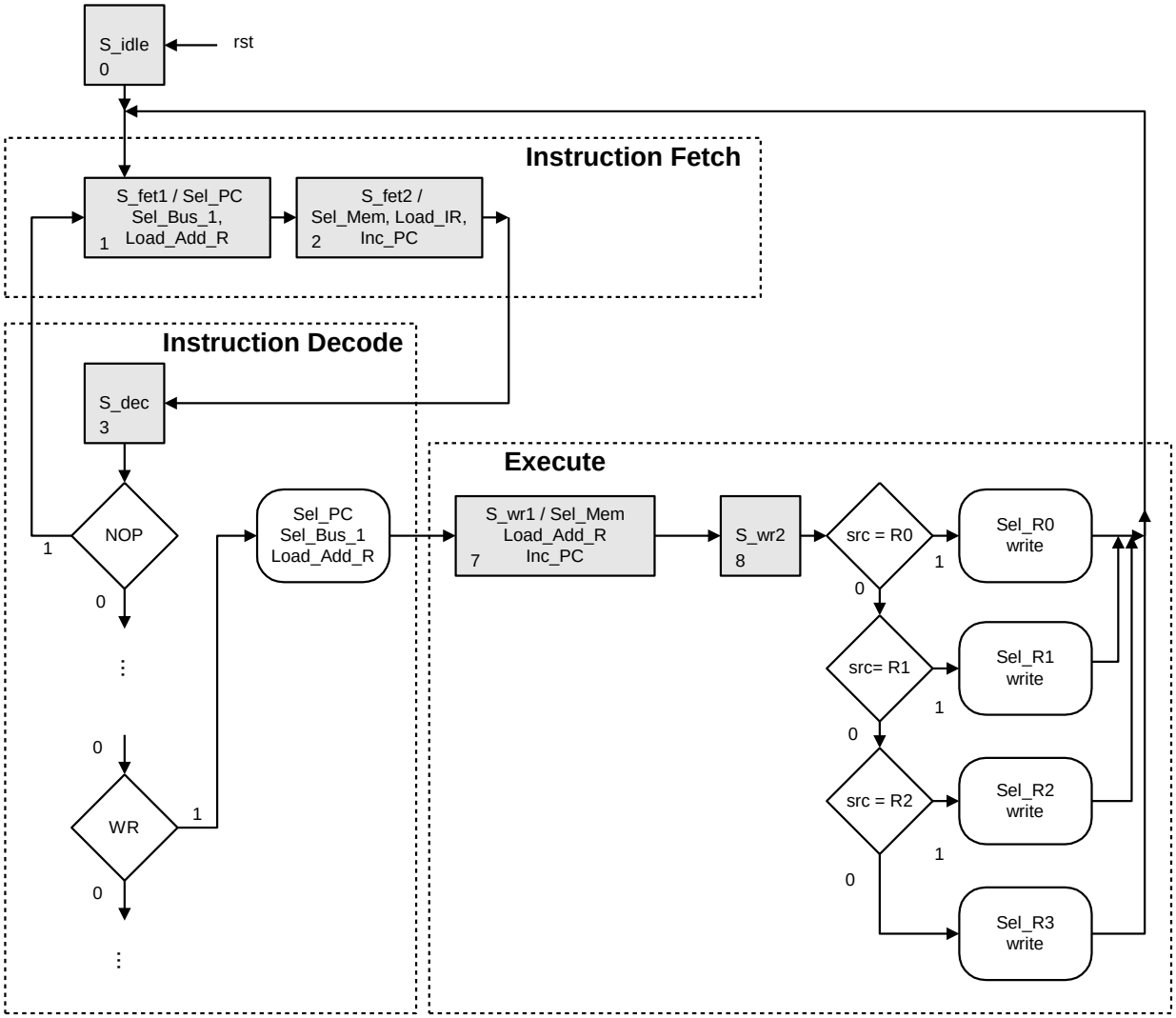
Controller ASM: NOT



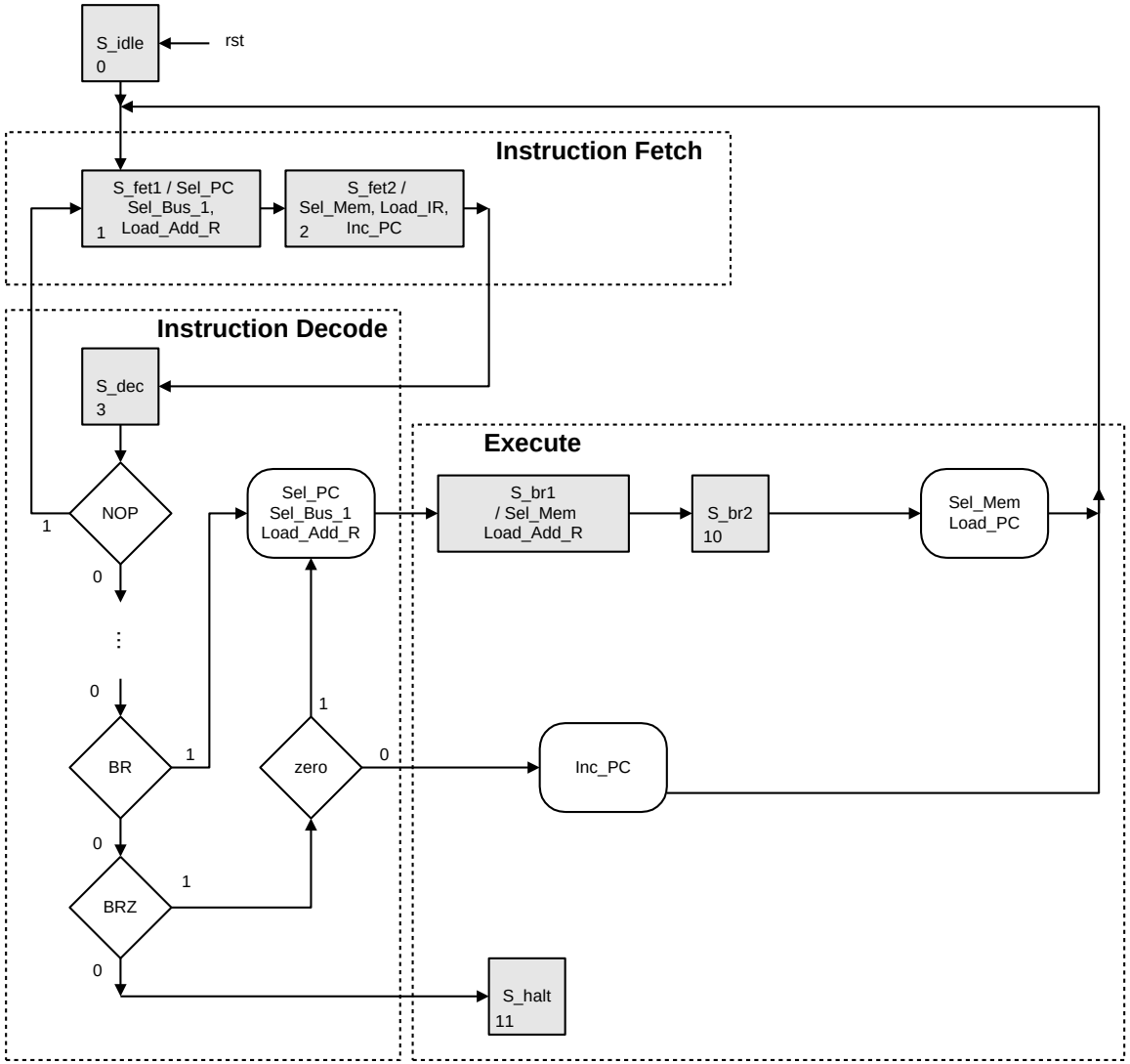
Controller ASM: RD



Controller ASM: WR



Controller ASM: BR/BRZ



Questions on RISC SPM

- Why not include memory as part of the processing unit?

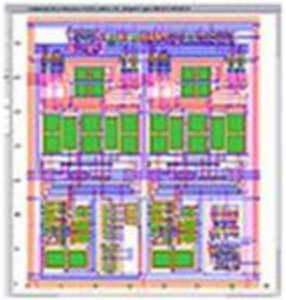
- How could the design be simplified?

Testing RISC SPM

- Clear the memory
- Load the memory with a simple program and data
- Execute simple program
  - Reads values from memory into registers
  - Perform subtraction to decrement a loop counter
  - Add register contents while executing the loop
  - Branch to a halt when the loop index is 0
- Probe memory locations and control signals to ensure correct execution

System on Chip (SoC)

From PCB to SoC



System on Chip cores