KING'S College LONDON

Hardware Design Lab

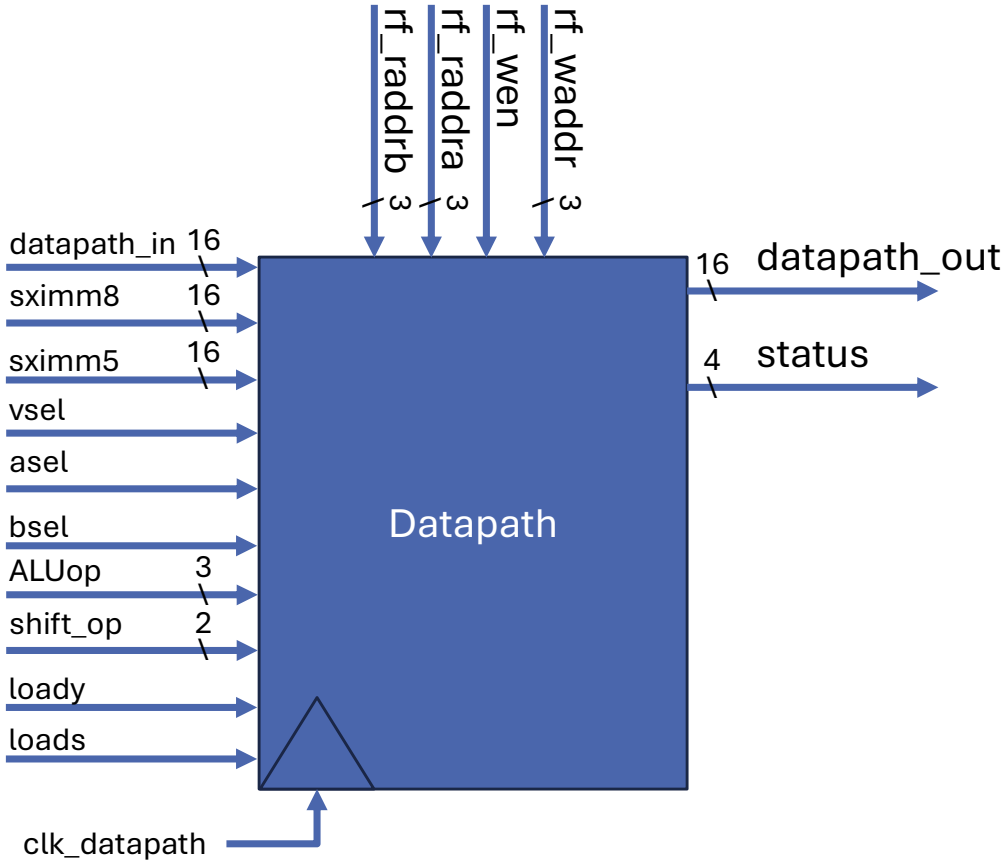
Task3: CPU FSM

Instructor: Dr. Haiyu Mao
TA: Zihao Pu
Ali Alsarraf
Stephen Johannesson

Nov.25, 2025

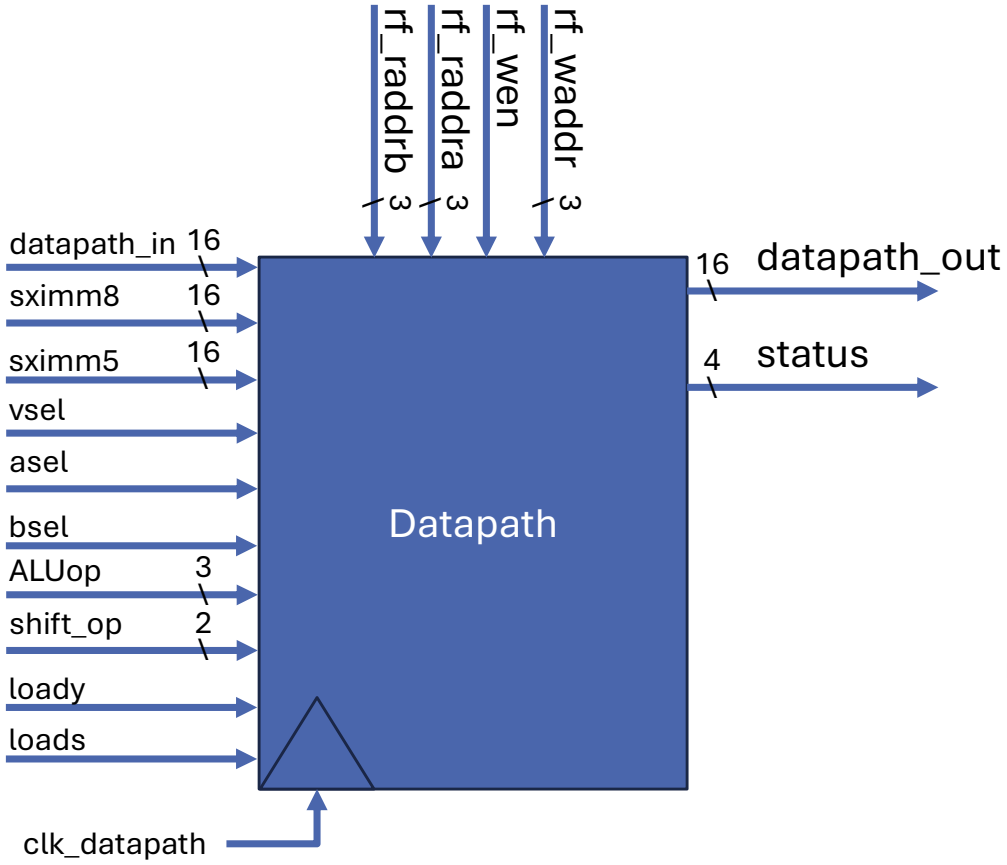
Gratefully acknowledge:
Prof. Onur Mutlu (ETH)
Dr. Yair Linn (TRIUMF Canada)
Prof. Tor Aamodt (UBC)

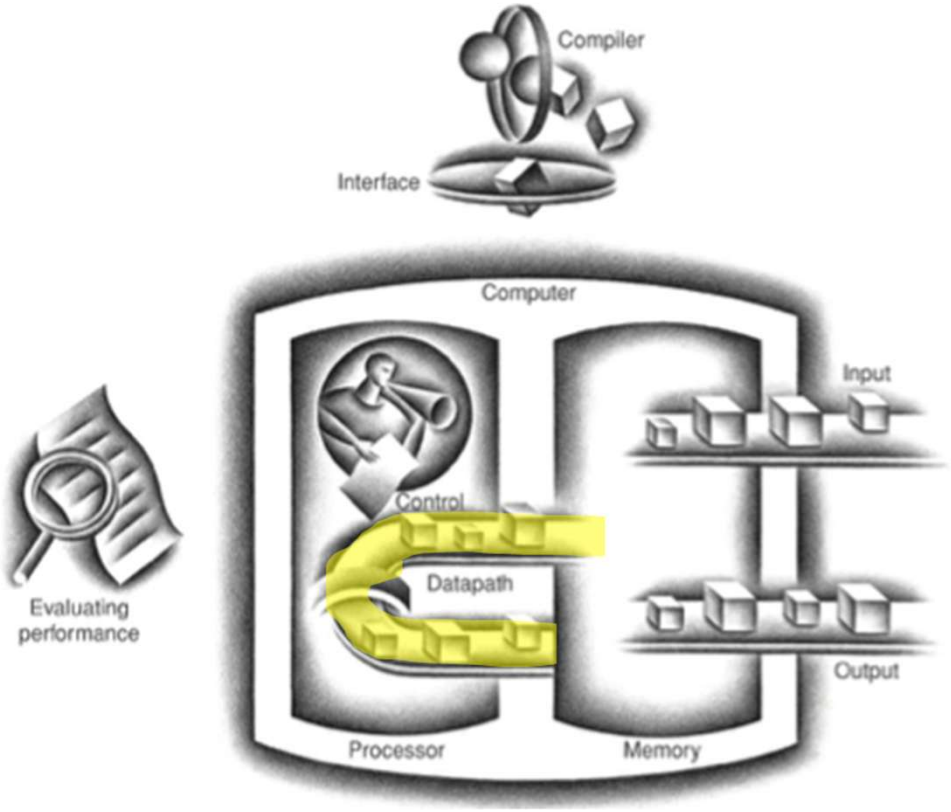
Recap: What we have now after Task 2



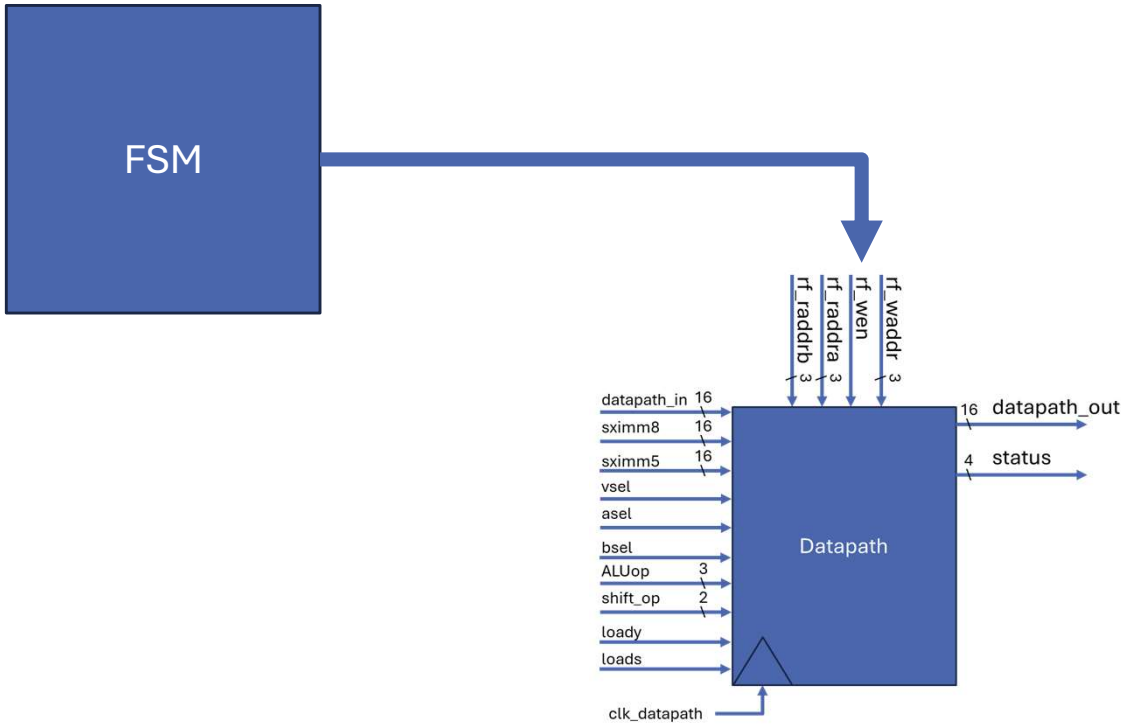
- Datapath as a block

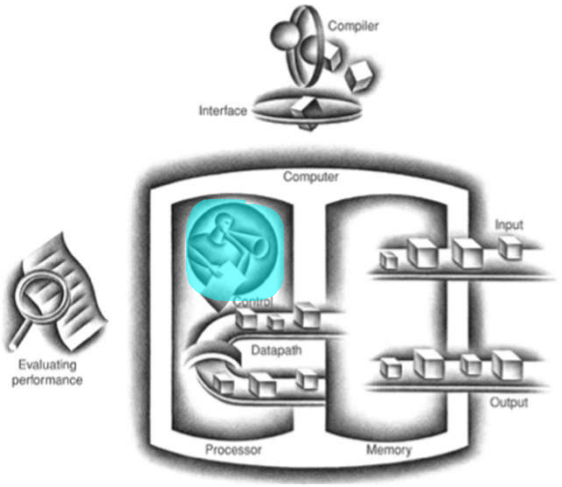
Recap on CPU: A Factory of Data



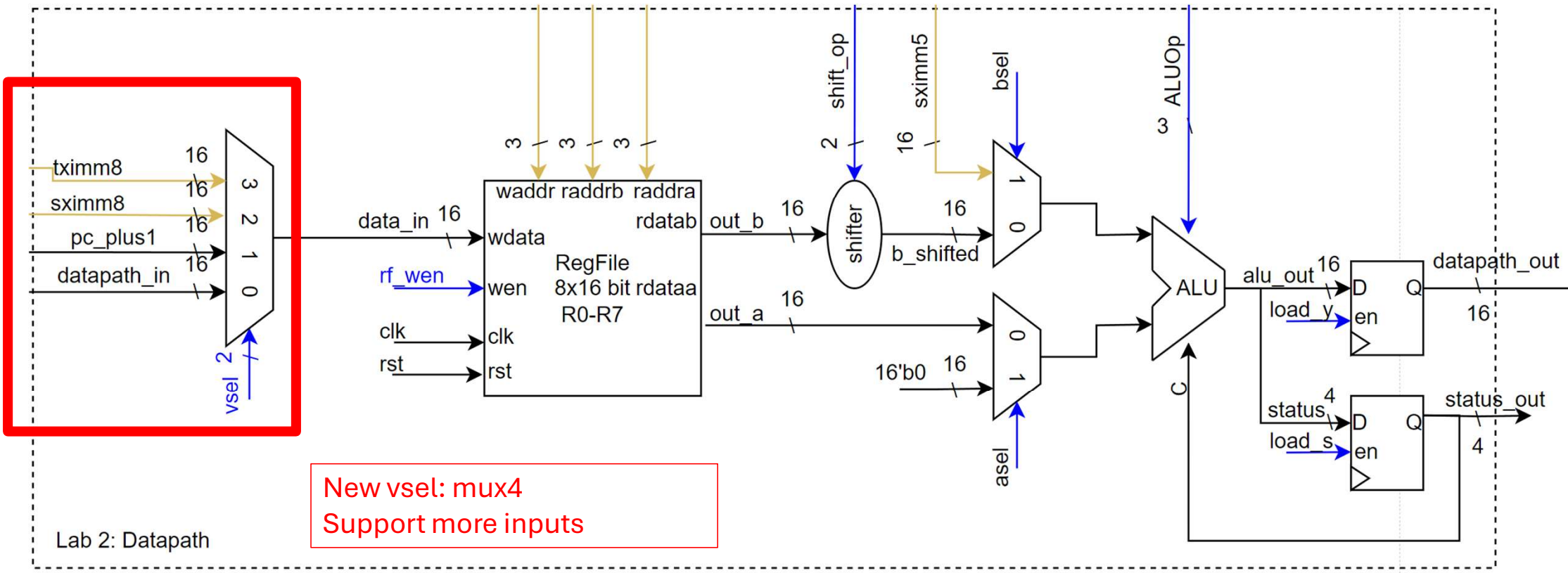


FSM: The Controller of Datapath

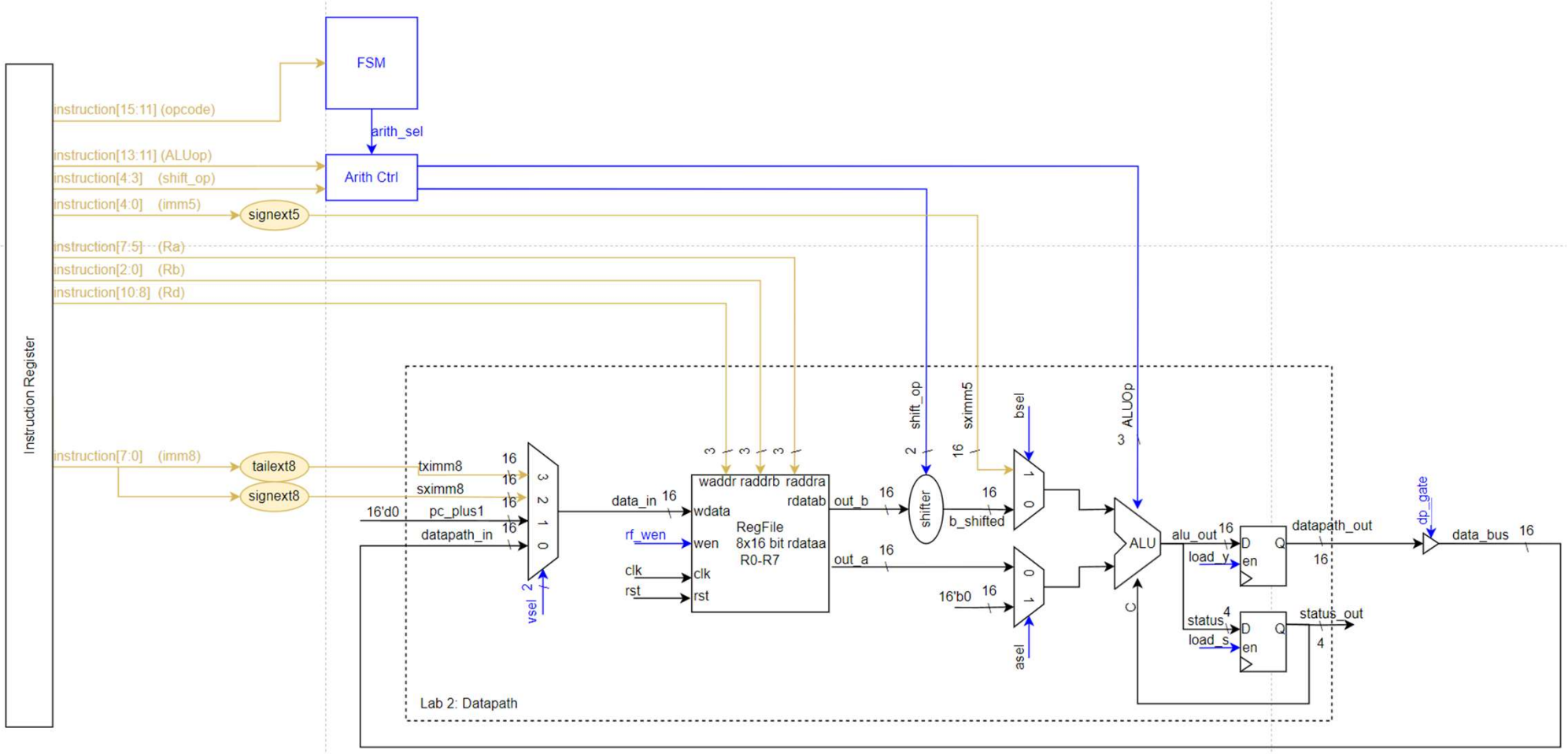




Datapath: Update

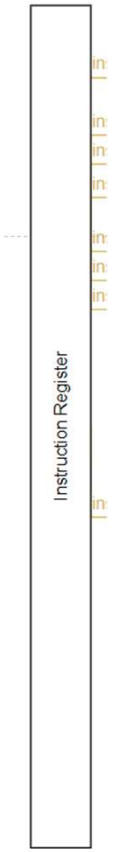


Lab 3: Overview

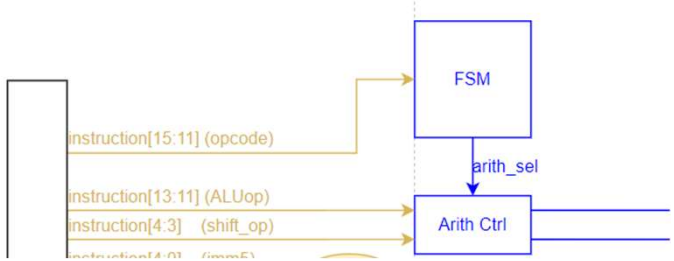


IR: Instruction Register

- Register that hold fetched instruction
- Synchronized to clk, but update/enabled with `load_ir`
- 16 bits, same length as system arch



Arith Ctrl

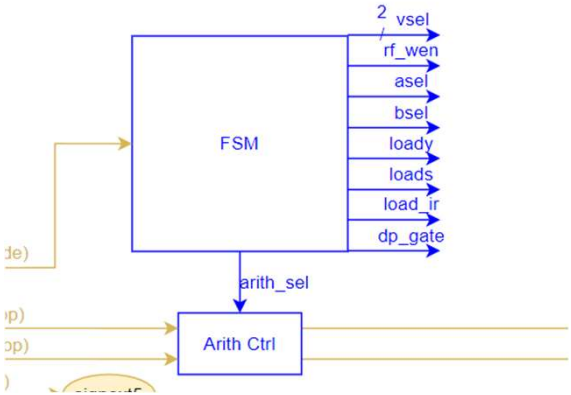


- Arith Ctrl: Similar to the ALU Decoder in the DDCA MIPS arch, but more simplified.
- One-bit control:
  - 1: Pass through mode. ALUop is 3'b000, and shift_op is 2'b00. Allow B_out pass to B_oprand. Use this mode in non-arithmetic operations.
  - 0: IR mode. ALUop and shift_op are from the instruction. Use this mode for R-type instructions and I-type instructions.

Lab 3: Supporting Instructions

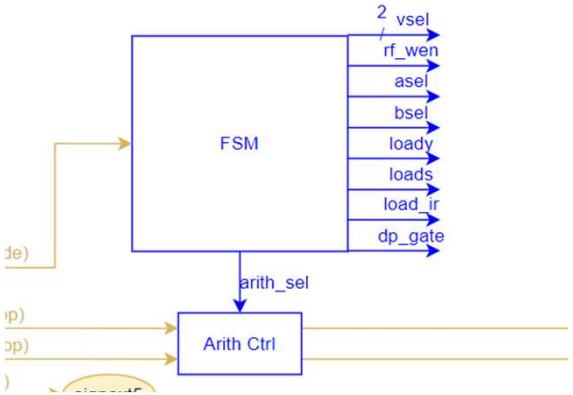
Mnemonic	Type	OpCode[15:11]		DST [10:8]	SRC1 [7:5]	SHIFT [4:3]	SRC2 [2:0]	Flags	Description & Operation
ARITHMETIC		Op[15:14]	ALUop[13:11]	Rd[10:8]	Ra[7:5]	shiftop[4:3]	Rb[2:0]		(Shifter applies to Rs2)
ADD	R	00	000	Rd	Ra	Shift	Rb	Z, N, C, V	Rd = Ra + Shift(Rb)
SUB	R	00	001	Rd	Ra	Shift	Rb	Z, N, C, V	Rd = Ra - Shift(Rb)
ADDC	R	00	010	Rd	Ra	Shift	Rb	Z, N, C, V	Add with Carry, Rd = Ra + Shift(Rb) + C
SUBC	R	00	011	Rd	Ra	Shift	Rb	Z, N, C, V	Sub with Carry, Rd = Ra - Shift(Rb) - C
AND	R	00	100	Rd	Ra	Shift	Rb	Z, N	Rd = Ra & Shift(Rb)
ANDBB	R	00	101	Rd	Ra	Shift	Rb	Z, N	Rd = Ra & Bar(Shift(Rb))
OR	R	00	110	Rd	Ra	Shift	Rb	Z, N	Rd = Ra | Shift(Rb)
ORBB	R	00	111	Rd	Ra	Shift	Rb	Z, N	Rd = Ra | Bar(Shift(Rb)
IMMEDIATE		Op[15:14]	ALUop[13:11]	Rd[10:8]	Ra[7:5]	IMM5[4:0]		Flags	(Merges Shift & Src2 into Imm5)
ADDI	I	01	000	Rd	Ra	sximm5		Z, N, C, V	Rd = Ra + SignExt(Imm5)
SUBI	I	01	001	Rd	Ra	sximm5		Z, N, C, V	Rd = Ra - SignExt(Imm5)
IMM REG LOAD		OpCode[15:11]		DST[10:8]		IMM8[7:0]			
LDI		10000		Rd	sximm8			-	Rd = SignExt(Imm8)
LUI		10001		Rd	sximm8			-	Rd = TailExt(Imm8)

LAB3: FSM

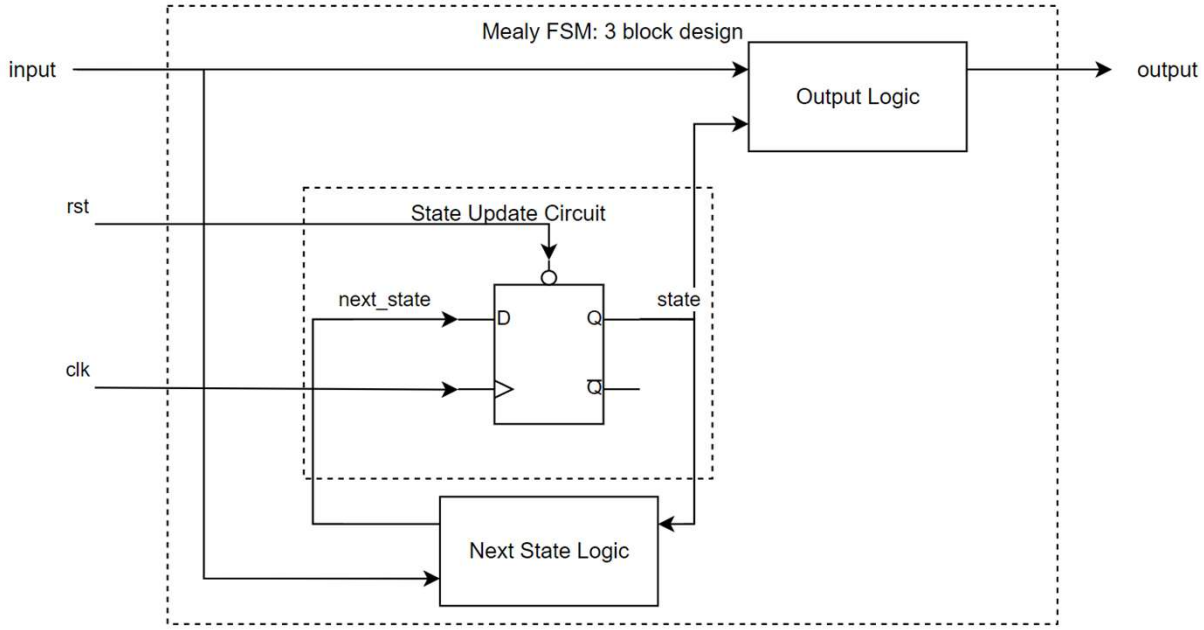


- FSM is used to control the datapath, IR, Arith_ctrl and datapath_out gate.
- Write FSM states to automatically control the CPU, as you did in Lab 2.
- You need to support the following instructions:
- R-type Instruction: ADD, SUB, ADDC, SUBC, AND, ANDBB, OR, ORBB
- I-type Instruction: ADDI, SUBI
- Load Register Immediate: LDI, LUI

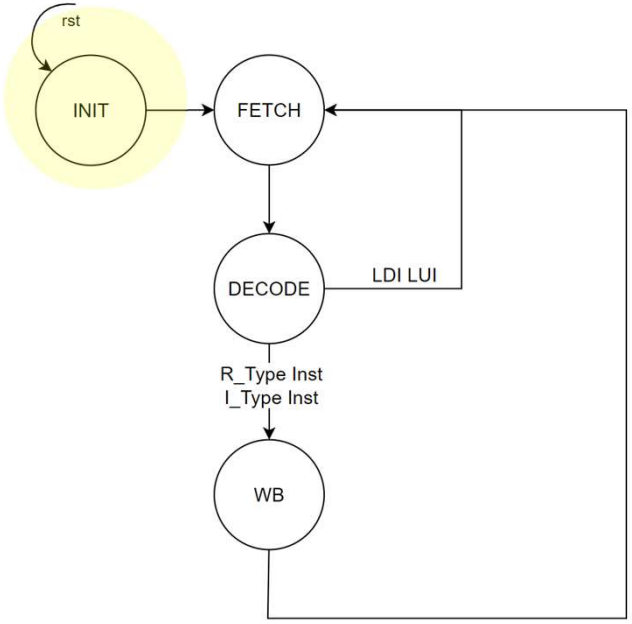
LAB3: FSM



- Suggested way of writing FSM: 3 block FSM(Mealy FSM)
  - Sequential circuit: status update
  - Combinational circuit: next state logic
  - Combinational circuit: output logic



Lab 3: FSM

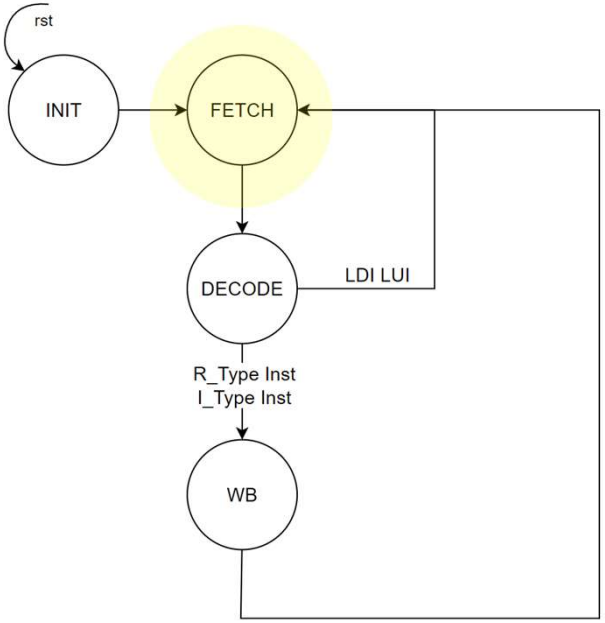


- INIT: Init state, no control signals asserted in INIT state

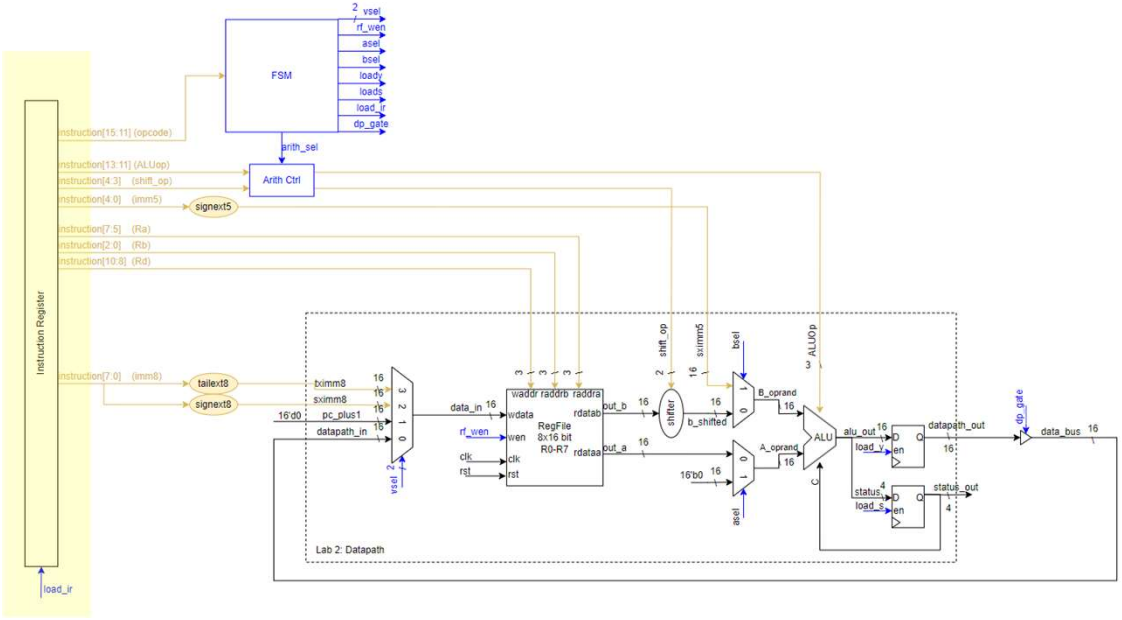
```
// Default values for control signals
vsel = VSEL_DB; // Default: 0
asel = 1'b0;
bsel = 1'b0;
loady = 1'b0;
loads = 1'b0;
rf_wen = 1'b0;
arith_sel = ARITH_SEL_IR;// Default: 0
load_ir = 1'b0;
dp_gate = 1'b0;
```

Instruction input	vsel	asel	bsel	loady	loads	rf_wen	arith_sel	load_ir	dp_gate	Next State
	0	0	0	0	0	0	0	0	0	FETCH

Lab 3: FSM

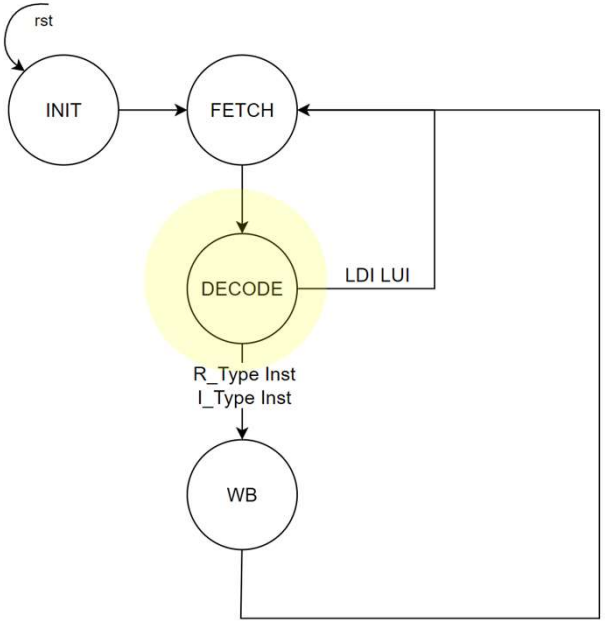


- FETCH: Fetch Instruction

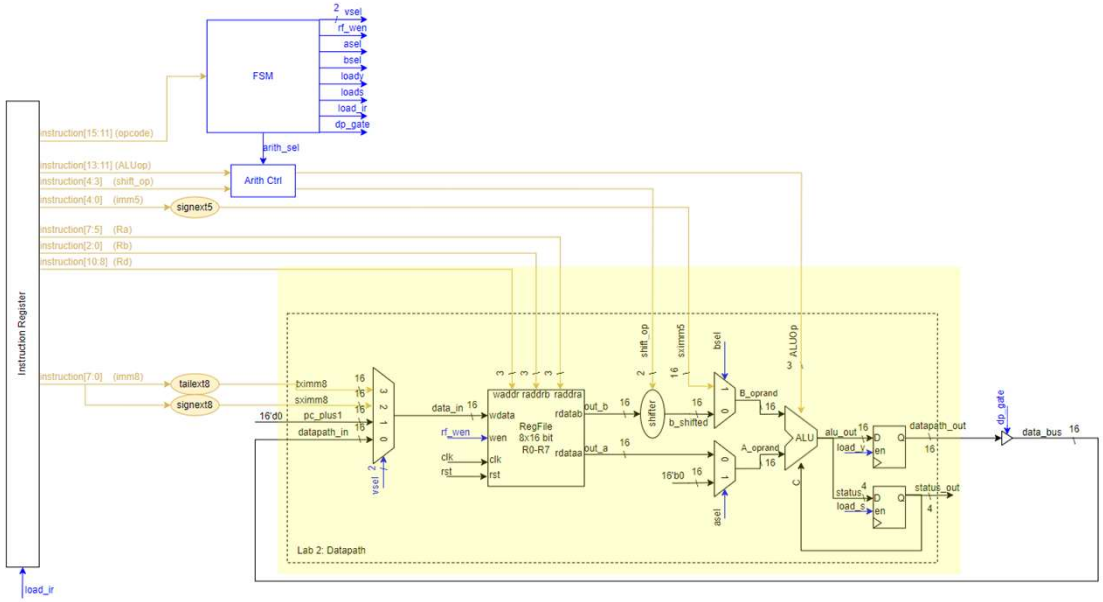


Instruction Input	vsel	asel	bsel	loady	loads	rf_wen	arith_sel	load_ir	dp_gate	Next State
	0	0	0	0	0	0	0	**1**	0	DECODE

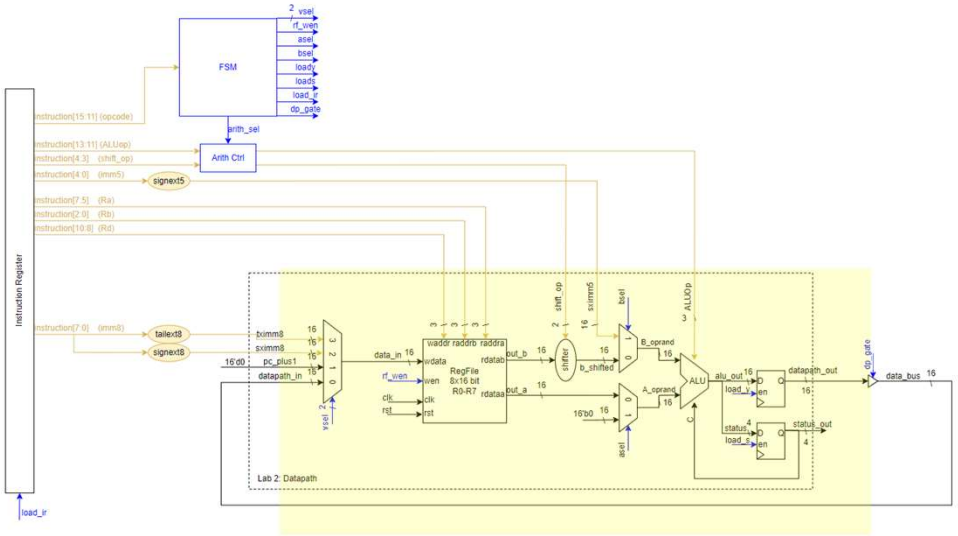
Lab 3: FSM



- DECODE: Decode instruction to datapath micro operations.



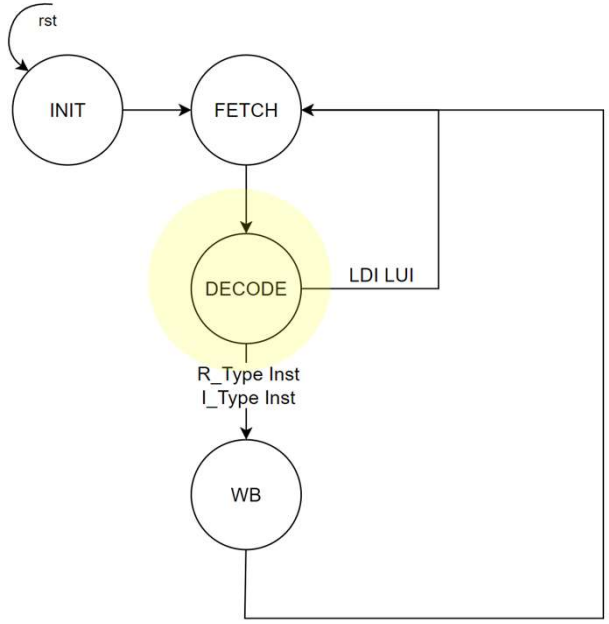
Lab 3: FSM



- DECODE: Decode instruction to datapath micro operations.

Inst:	vsel	asel	bsel	loady	loads	rf_wen	arith_sel	load_ir	dp_gate	Next State
R-type										
I-type										
LUI										
LDI										

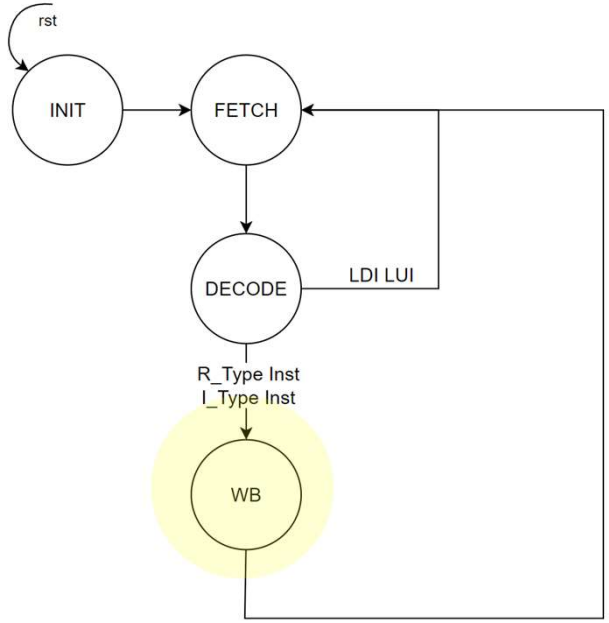
Lab 3: FSM



- DECODE Implementation: use case block

```
// Keep other default values, example control
      case(ir_opcode)
        ADD: begin
         // Your code here
        end
         // ......
         // More cases
         // ......
        LDI: begin
         // Your code here
        end
        LUI: begin
         // Your code here
        end
        default: begin
         // Your code here
        end
      endcase
```

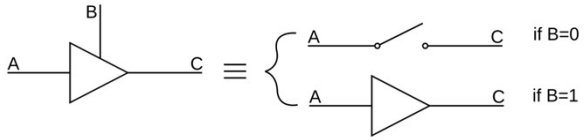
Lab 3: FSM

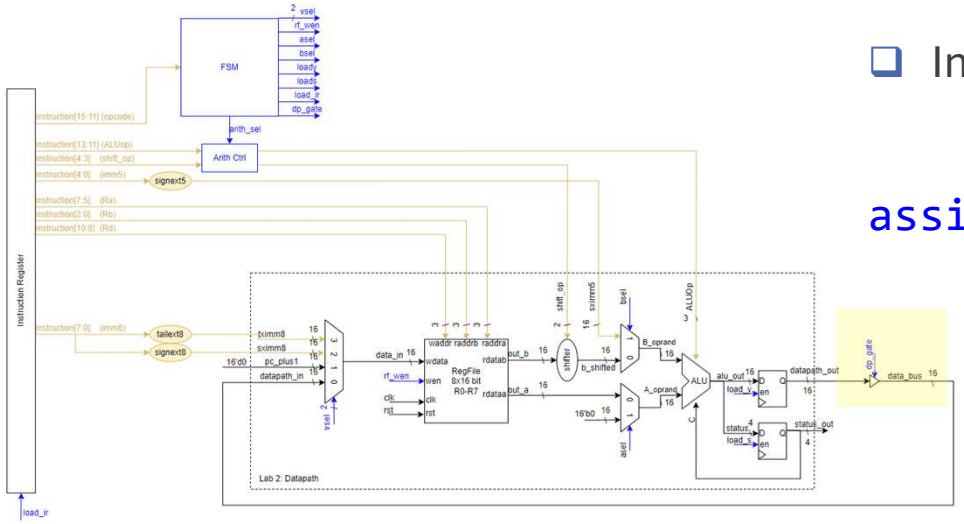


- WB: Write back output to regfile

Instruction Input	vsel	asel	bsel	loady	loads	rf_wen	arith_sel	load_ir	dp_gate	Next State
	0	0	0	0	0	1	0	**0**	1	FETCH

Tri-state buffer and data bus

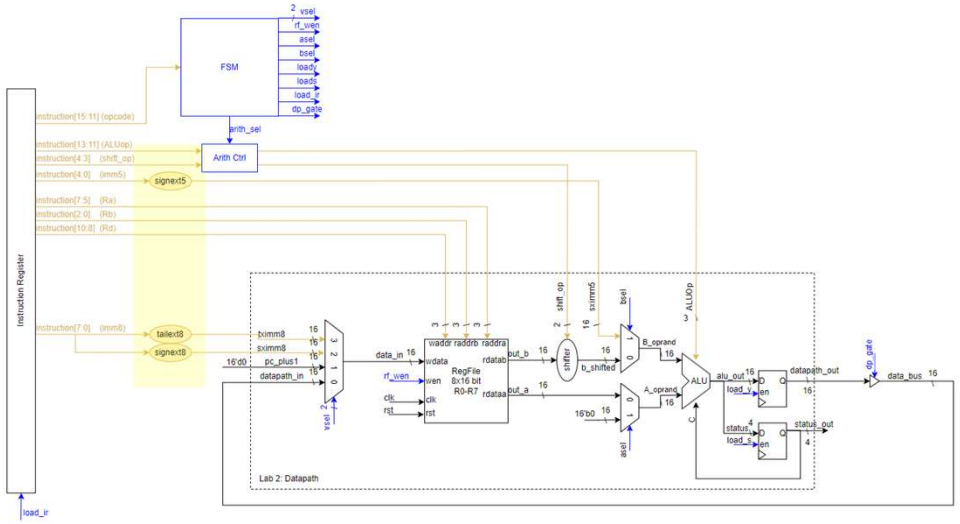




- In our RISKing CPU, the **data bus** is shared by other modules. We will discuss this in Lab 4.
- Therefore, we need a gate to control when to connect *datapath_out* to *data_bus*
- In order to do that, a tri-state gate (or tri-state buffer) is implemented to do so.
- Implement tri-state buffer in Verilog:

```
assign C = B ? A:1'bz;
```

Sign Extender and Tail Extender



Sign Extender: When converting to data type with more bits, you need to extend the sign, which is the MSB.

E.g. 8-bit number: 8’b1010_1010

Extended 16-bit number: 16’b1111_1111_1010_1010

8-bit number: 8’b0101_0101

Extended 16-bit number: 16’b0000_0000_0101_0101

Tail Extender: A shifter that shifts left N bits

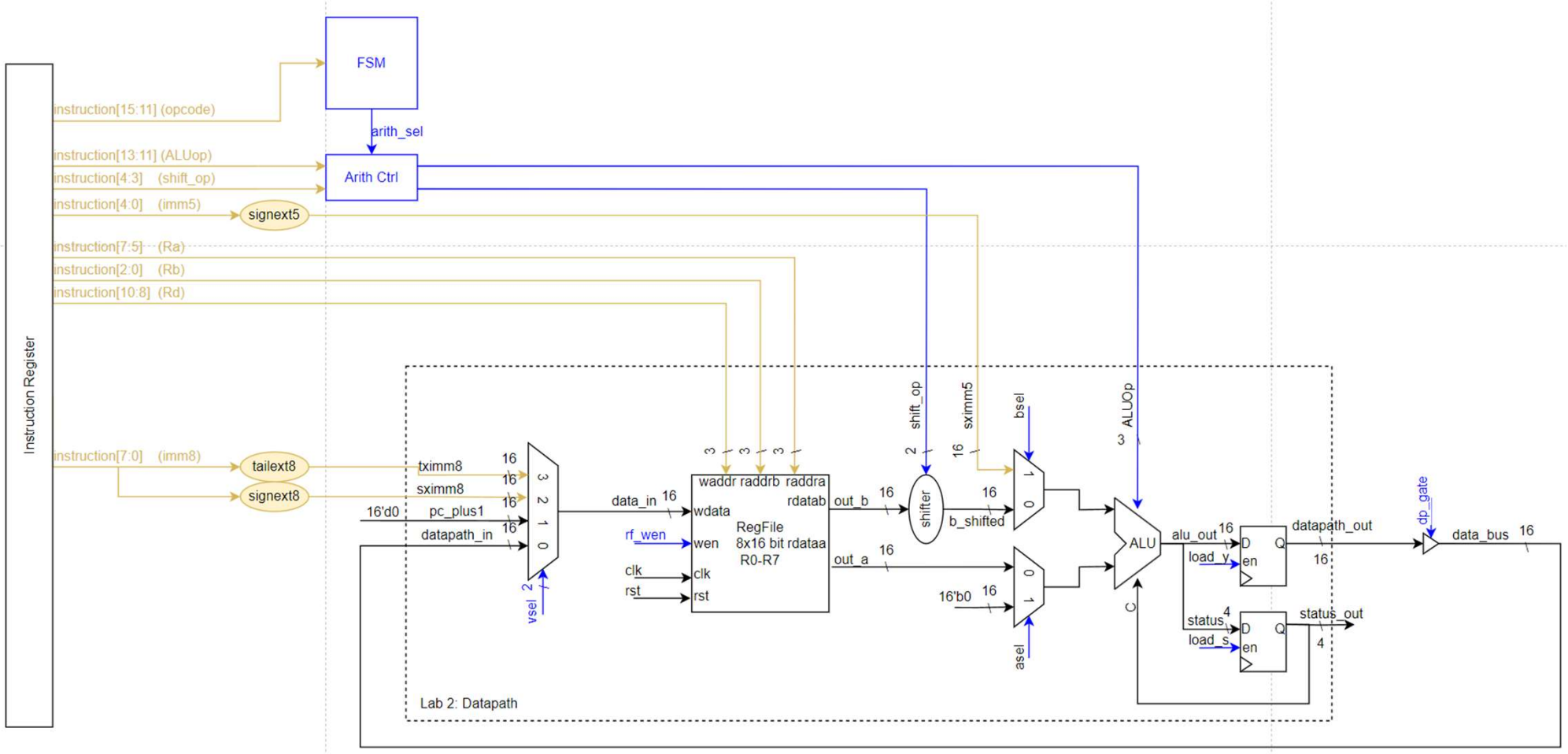
E.g. 8-bit number: 8’b1010_1010

Extended 16-bit number: 16’b1010_1010_0000_0000

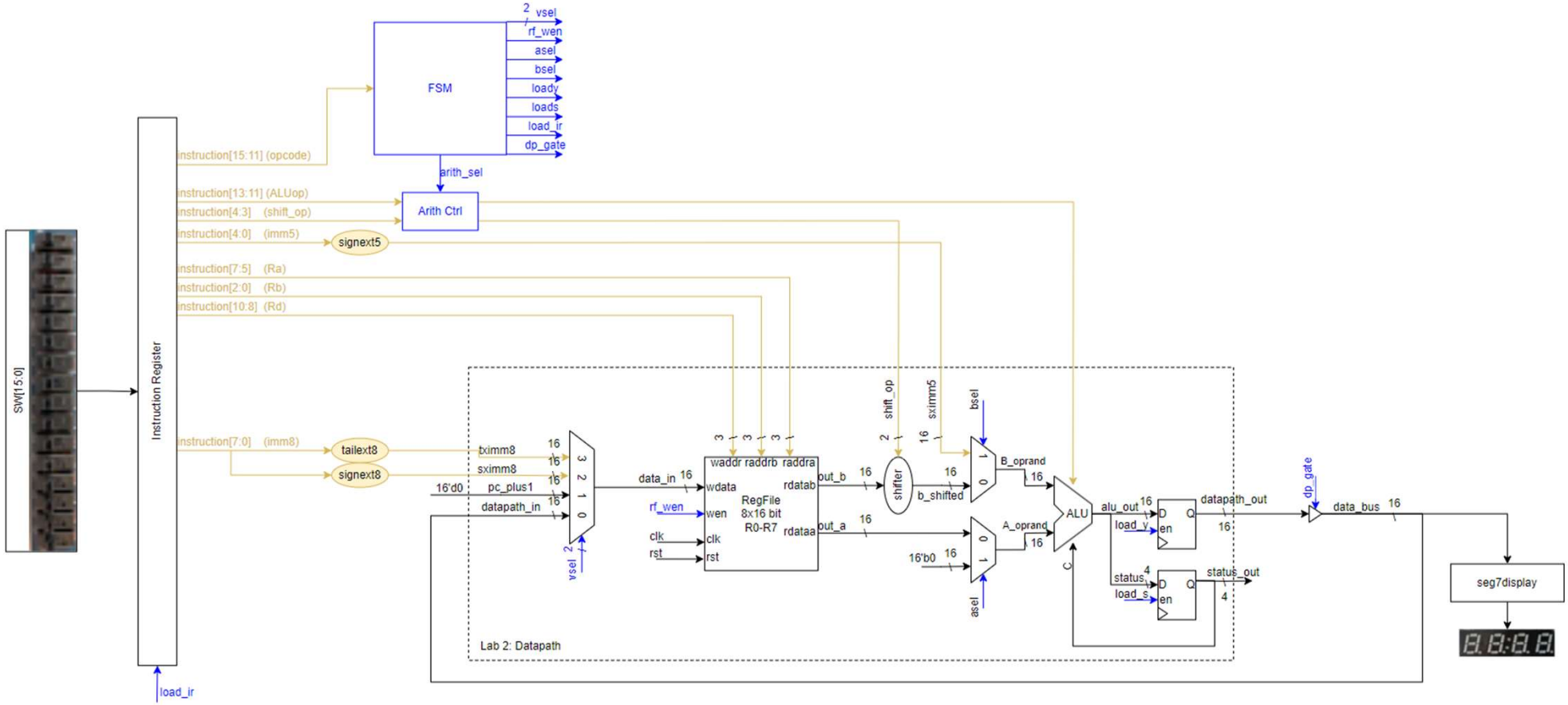
8-bit number: 8’b0101_0101

Extended 16-bit number: 16’b0101_0101_0000_0000

Lab 3: Overview



Lab 3: TOP


FSM
vsel
rf_wen
asel
bsel
loady
loads
load_ir
dp_gate
arith_sel
Arith Ctrl
signext5
tailext8
signext8
SW[15:0]
Instruction Register
load_ir
instruction[15:11] (opcode)
instruction[13:11] (ALUop)
instruction[4:3] (shift_op)
instruction[4:0] (imm5)
instruction[7:5] (Ra)
instruction[2:0] (Rb)
instruction[10:8] (Rd)
instruction[7:0] (imm8)
bximm8
sximm8
16'd0
pc_plus1
datapath_in
data_in
rf_wen
clk
rst
waddr raddrb raddra
wdata
wen
RegFile
8x16 bit
R0-R7
rdatab
rdataa
out_b
out_a
shift_op
sximm5
shifter
b_shifted
bsel
B_oprand
A_oprand
16'b0
asel
ALU
alu_out
load_v
status
load_s
datapath_out
status_out
dp_gate
data_bus
seg7display
Lab 2: Datapath