Hardware Design Lab

Task2: Datapath Design

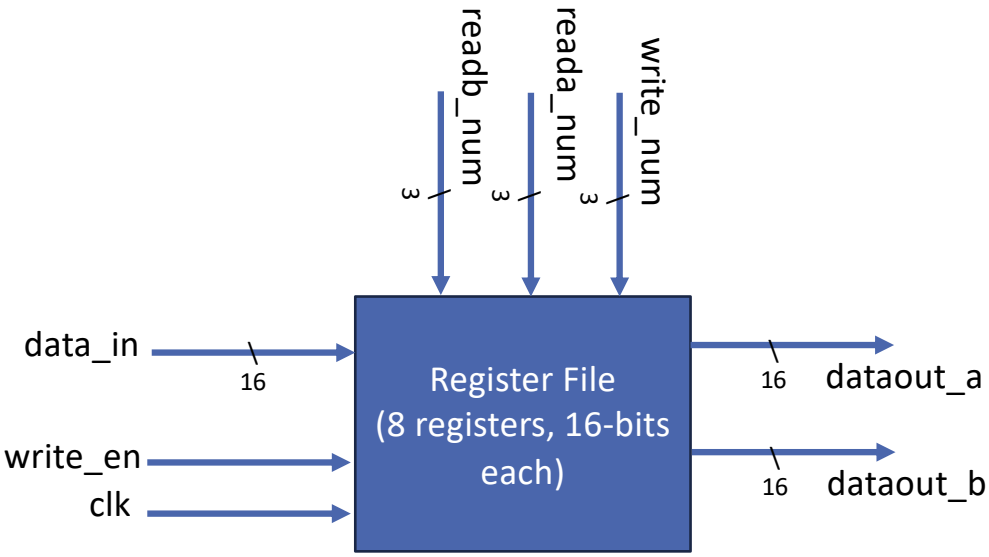
Instructor: Dr. Haiyu Mao
TA: Zihao Pu
Ali Alsarraf
Stephen Johannesson

Nov.25, 2025

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

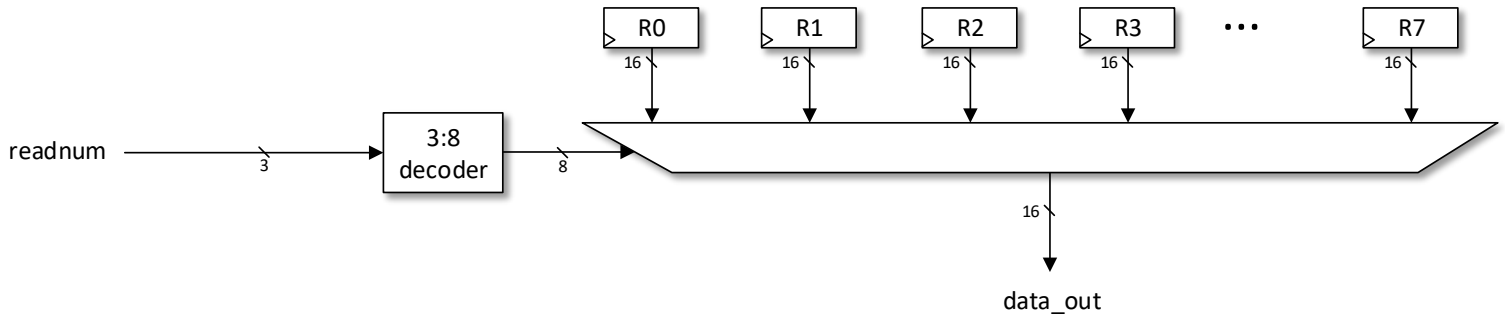
Register File

- A **Register File** is a small, fast memory structure inside the CPU that stores temporary data and operands for instructions.
- Role in Datapath:
  - Supplies operands to the ALU for arithmetic/logic operations.
  - Stores intermediate results and frequently used variables.
  - Interfaces with instruction decode stage for register addressing.



Register File: Design

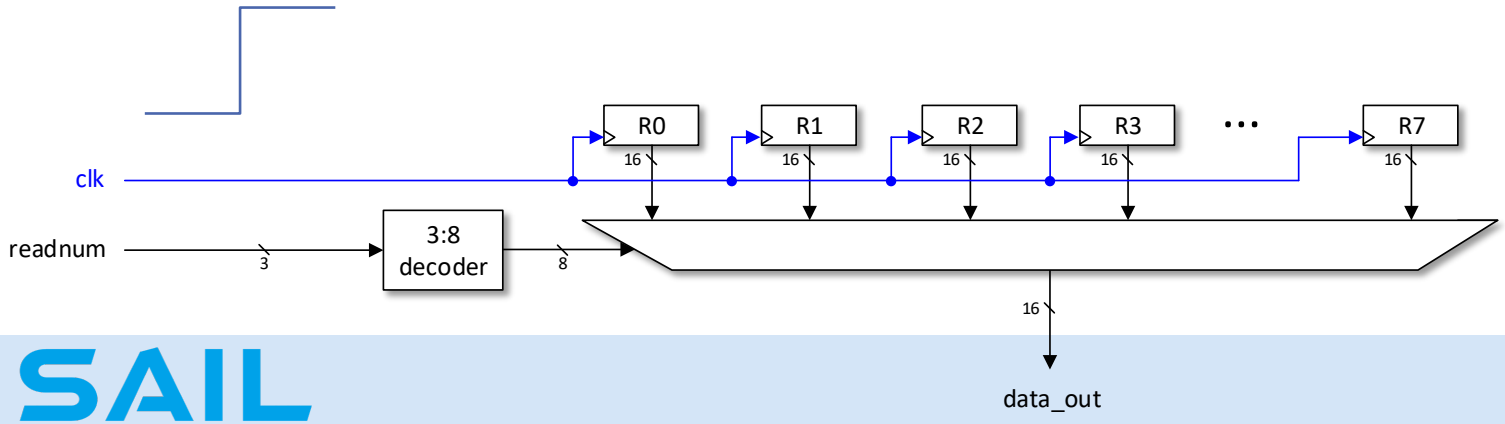
- Writing to the register file?



Register File: Design

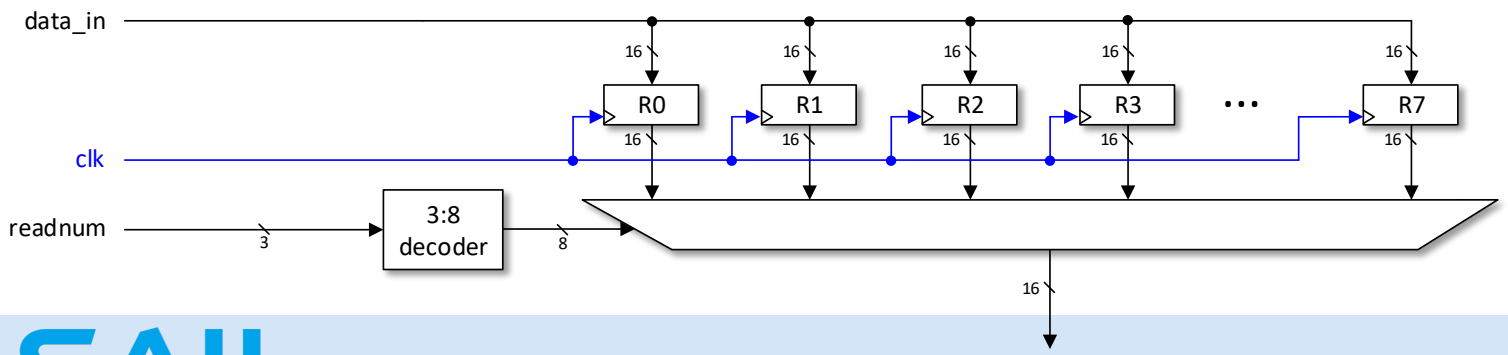
Writing to the register file…

When? Write data into the register file on rising edge of clk.



Register File: Design

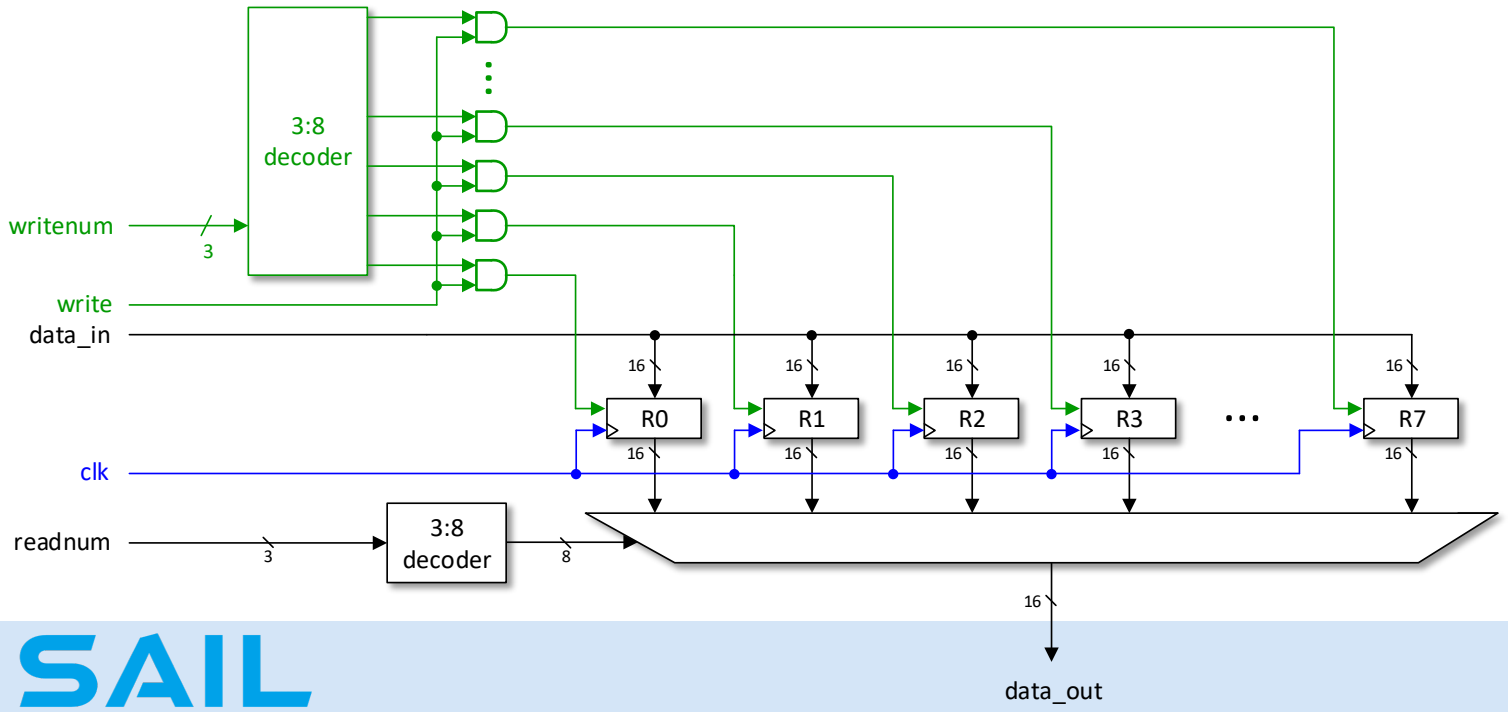
- Writing to the register file…
- Where? Data to be written into the register supplied on “data_in”



Register File

Writing to the register file…

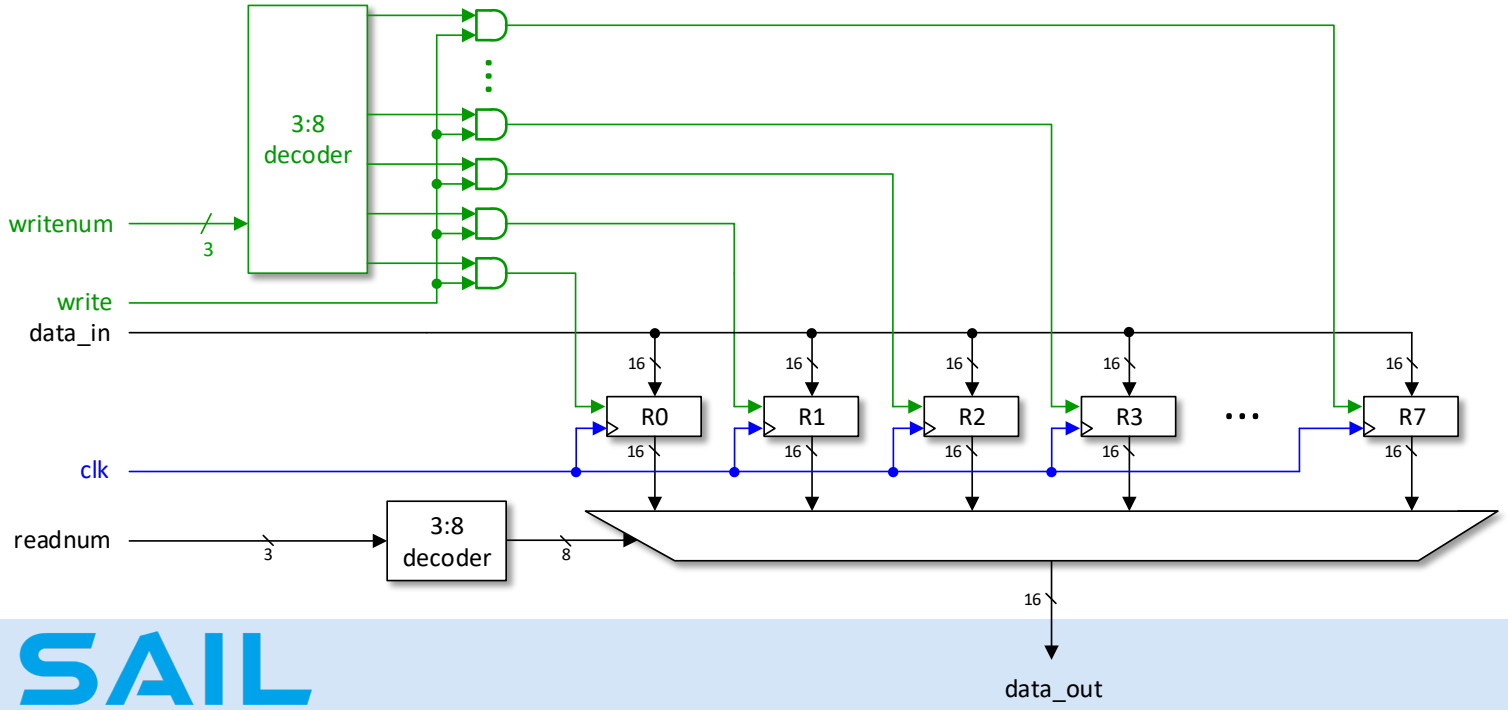
How? Enable only one register using decoder and AND gates:



Register File

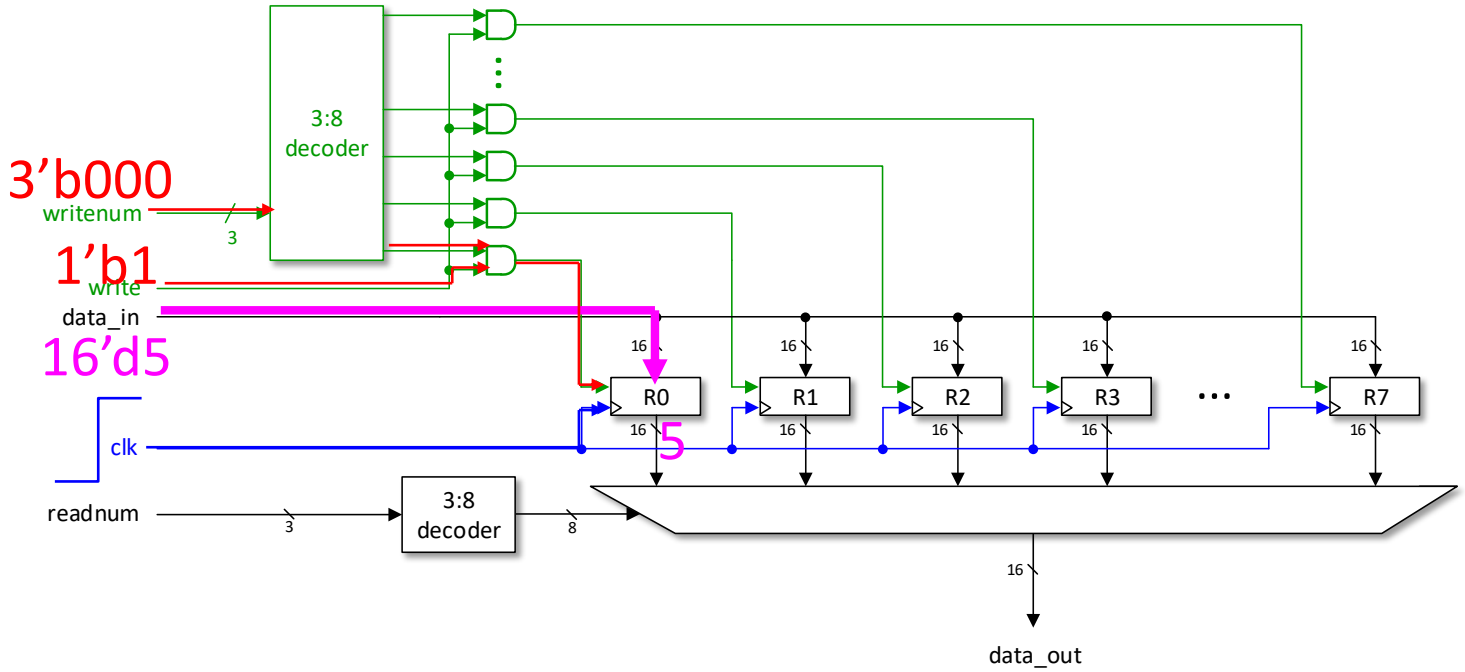
Writing to the register file:

1. Put the data you want to write on the input bus (`data_in`)
2. Put the register number on `writenum` (NOTE: only 3 bits!)
3. When `clk` goes high, the data is written

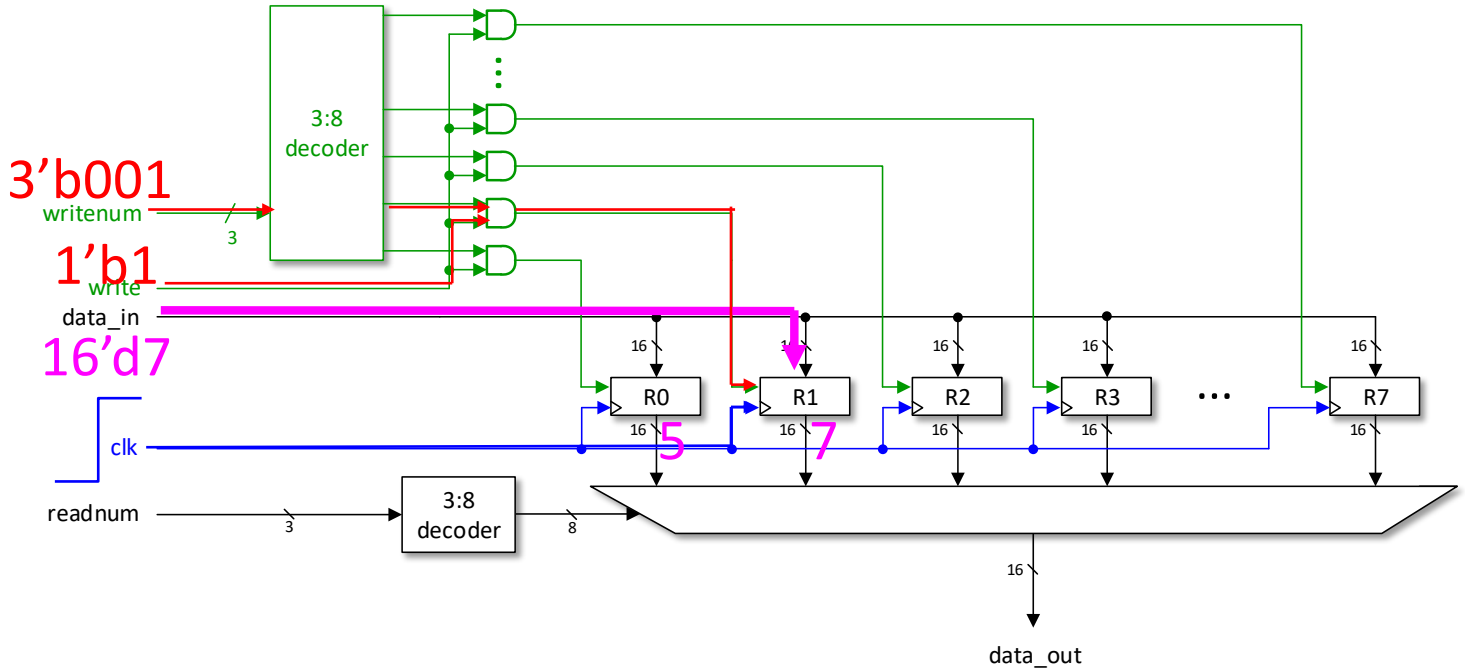


Example: Writing 5 into R0

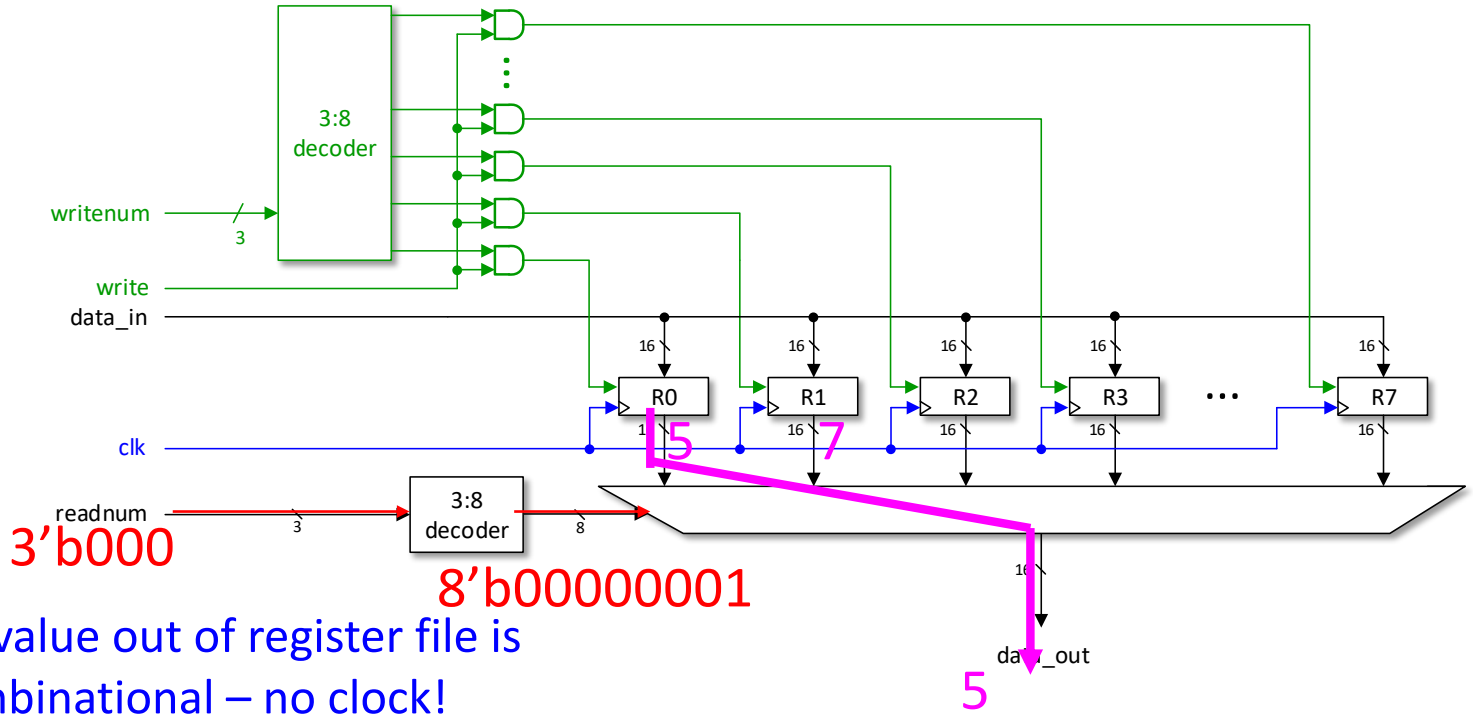
- Writing a value into the register file is clock edge sensitive!



Example: Writing 7 into R1



Register File: Reading R0



Reading a value out of register file is purely combinational – no clock!

Why only 8 Registers?

- Smaller is faster
- Reason: delay increases with length of wires
- (Generally, all others things being equal, prefer computers to be faster.)

- For arithmetic instructions in the Simple RISC Machine ISA all “operands” must be in the register file.
- Other ISAs do not make same choice (x86, MIPS, ARM32, ARM64, etc.).

Datapath Example w/ Registers

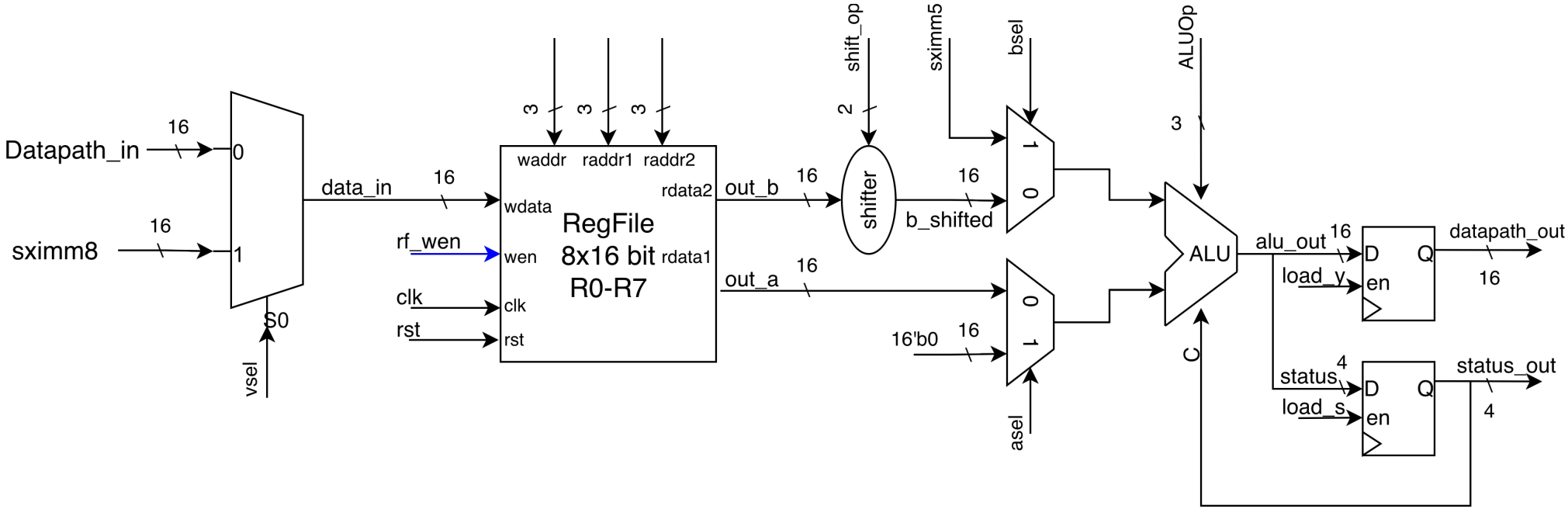
The variables f, g, h, i, and j are assigned to the registers r0, r1, r2, r3, and r4, respectively. What is the compiled ARM code for the following C?

```
f = (g + h) – (i + j);
```

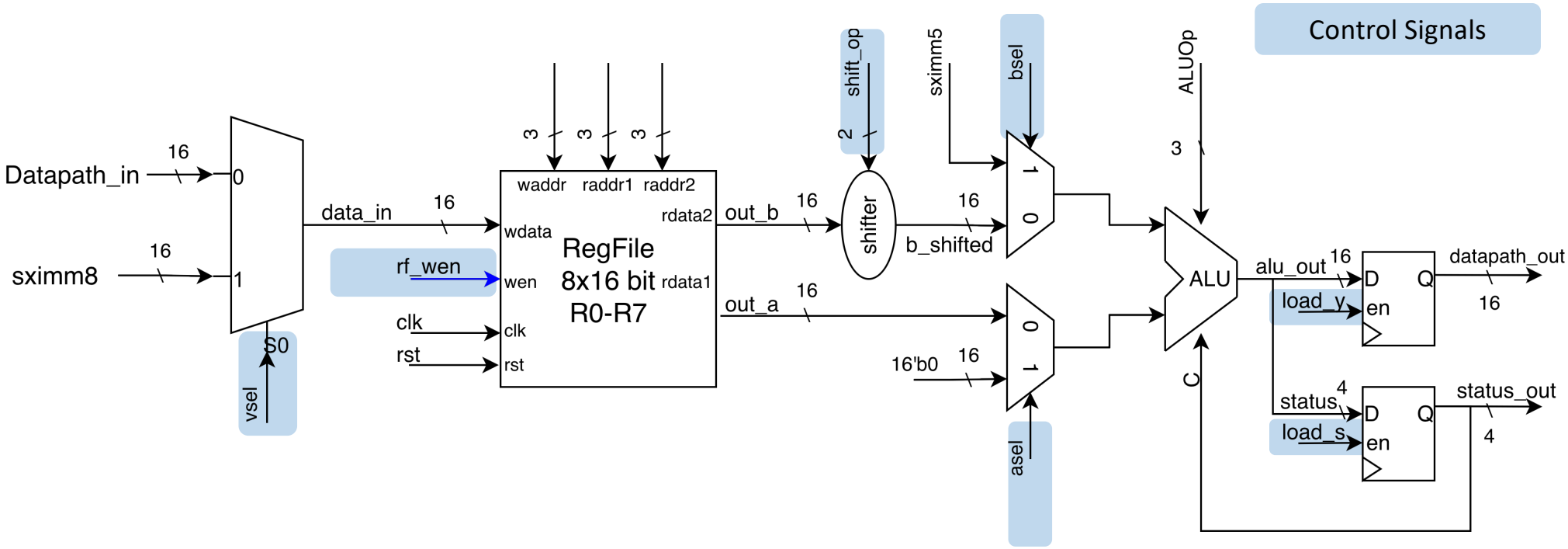
Solution:

```
ADD R5,R1,R2 // register R5 contains g + h
ADD R6,R3,R4 // register R6 contains i + j
SUB R0,R5,R6 // R4 gets R5 – R6,
             // which is (g + h) – (i + j)
```

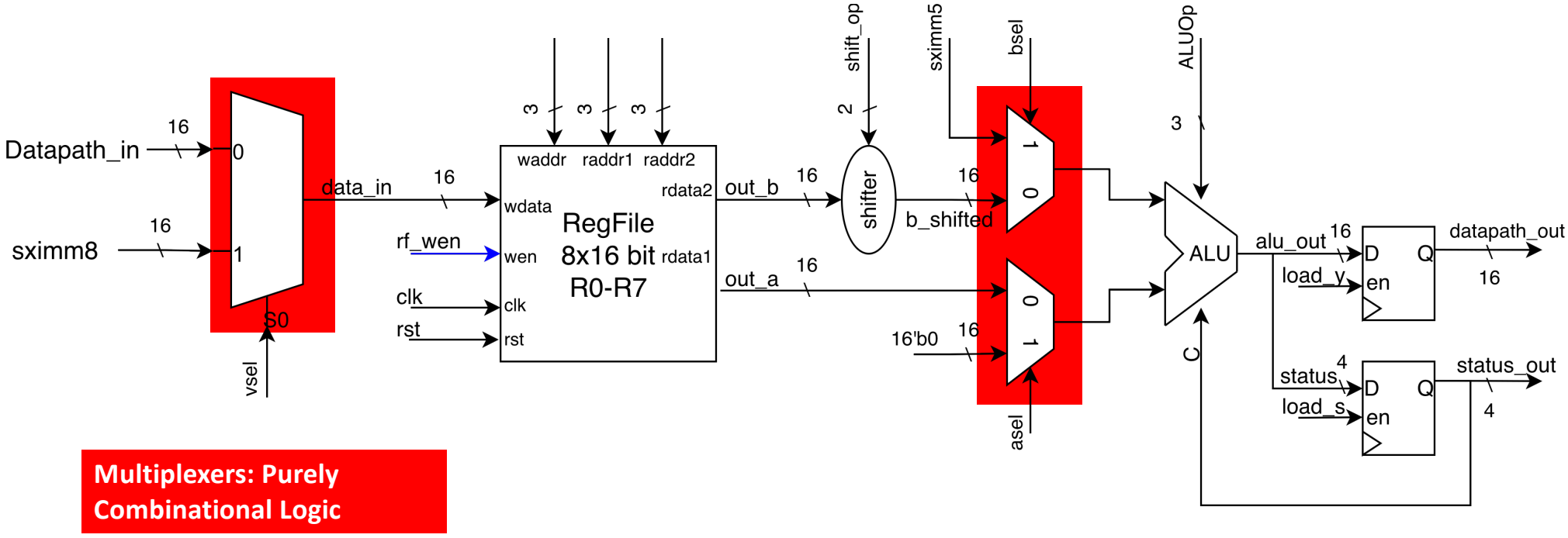
RISKing Datapath: Overview



RISKing Datapath: Overview

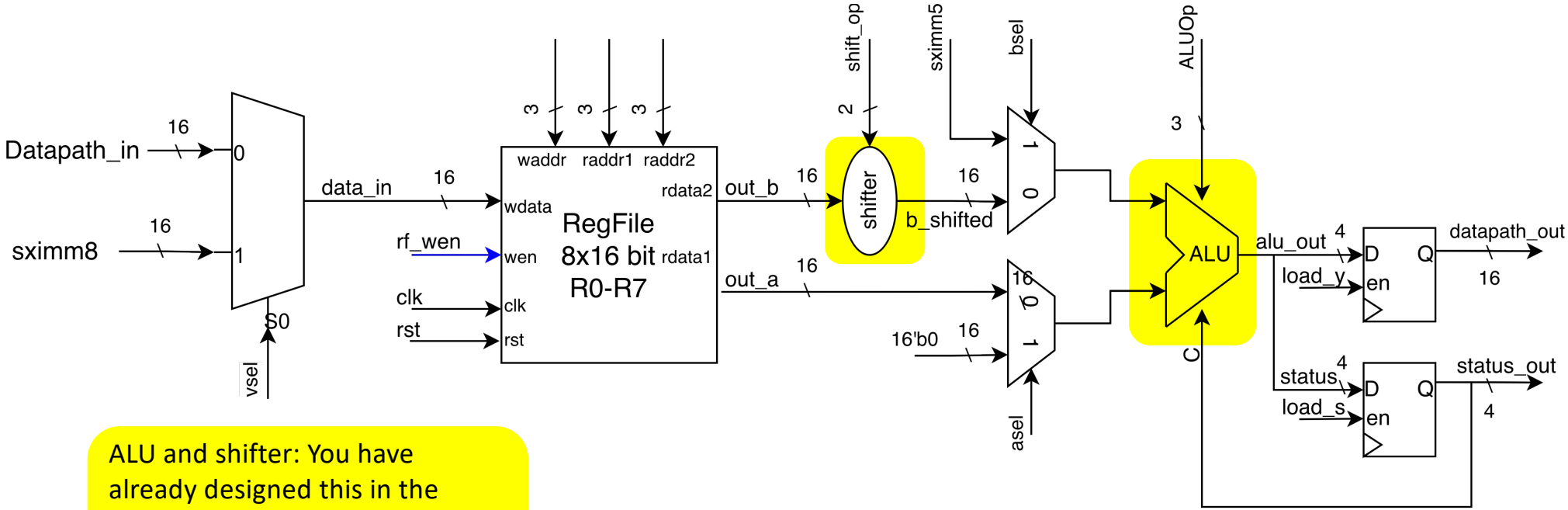


RISKing Datapath: Overview



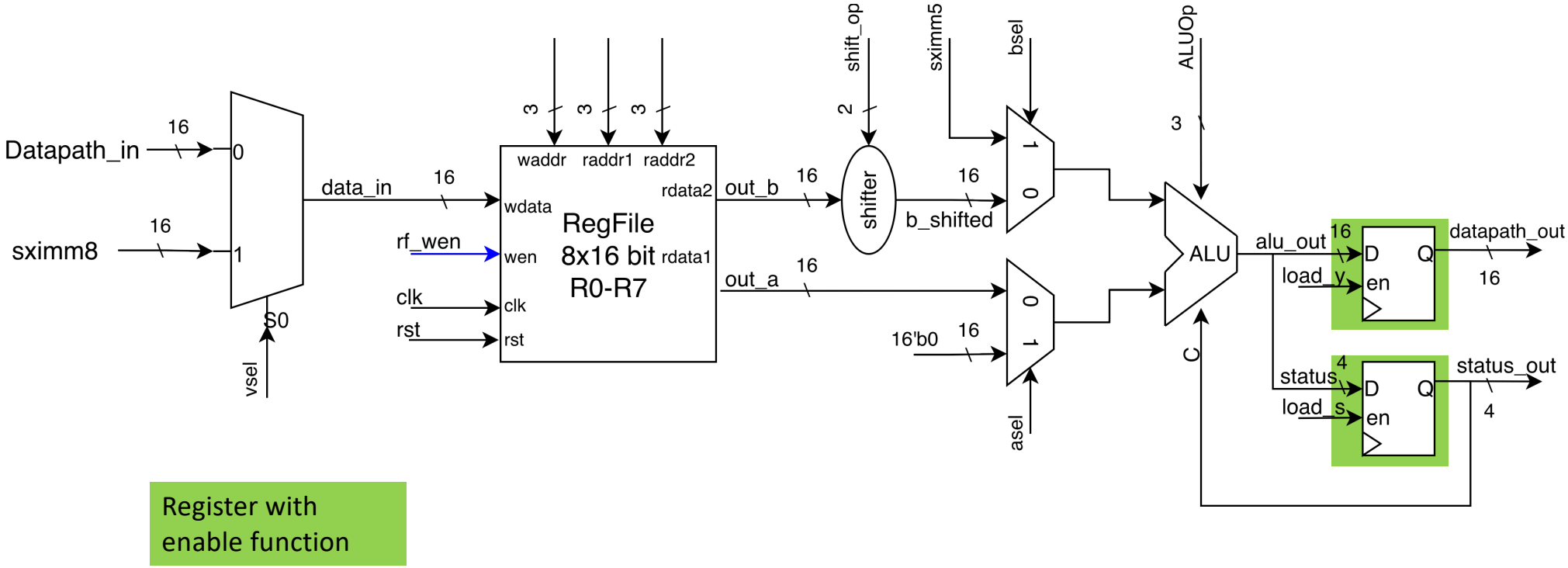
Multiplexers: Purely Combinational Logic

RISKing Datapath: Overview



ALU and shifter: You have already designed this in the earlier task. You should directly make an instantiation in this task.

RISKing Datapath: Overview



Register with enable function

From C to Assembly

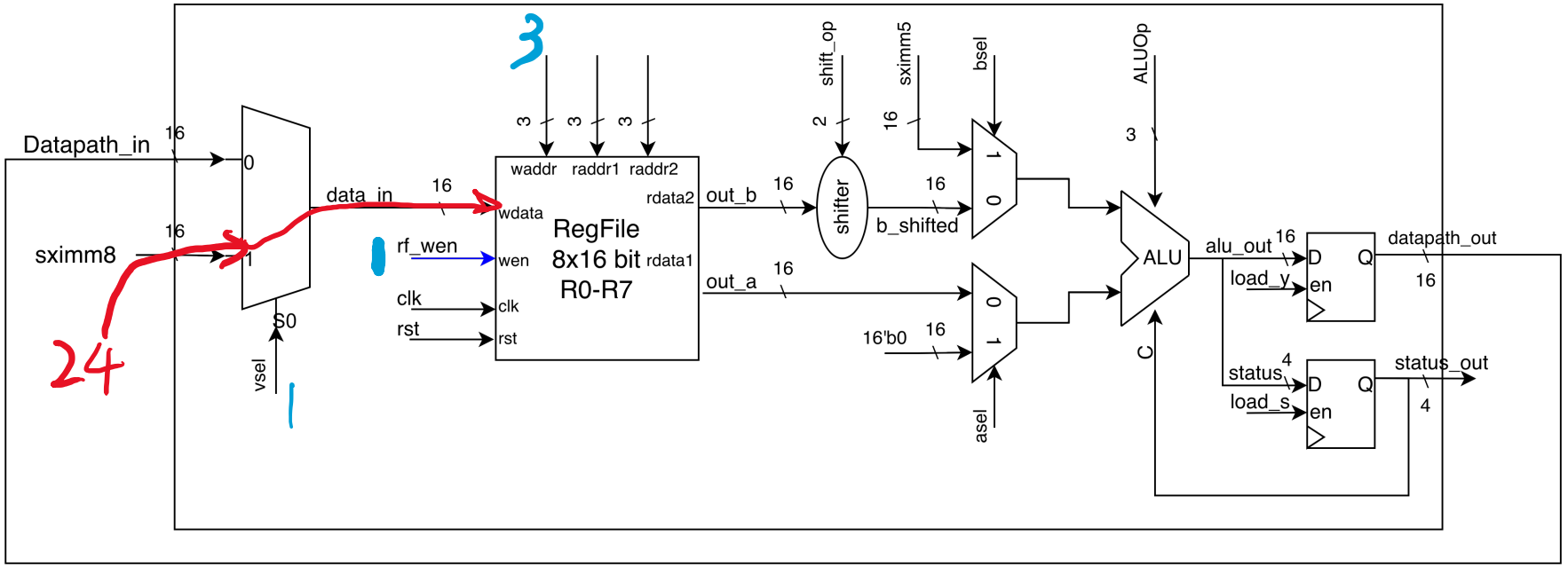
- Machine cannot read C code directly, obviously.
- Therefore, C code must compile to Assembly language first.
- Each instruction takes between 1 and 4 cycles to execute: Each cycle, certain control signals are asserted to perform the required operations
- Which control signals are asserted depends on which instruction is being executed. A decoder and state machine drives the signals at the right time. You'll design that part in Lab Task 3.

```
a = 0x24;
b = 0x13;
c = a + b; // c should be 0x37
d = c
```

Compile

```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

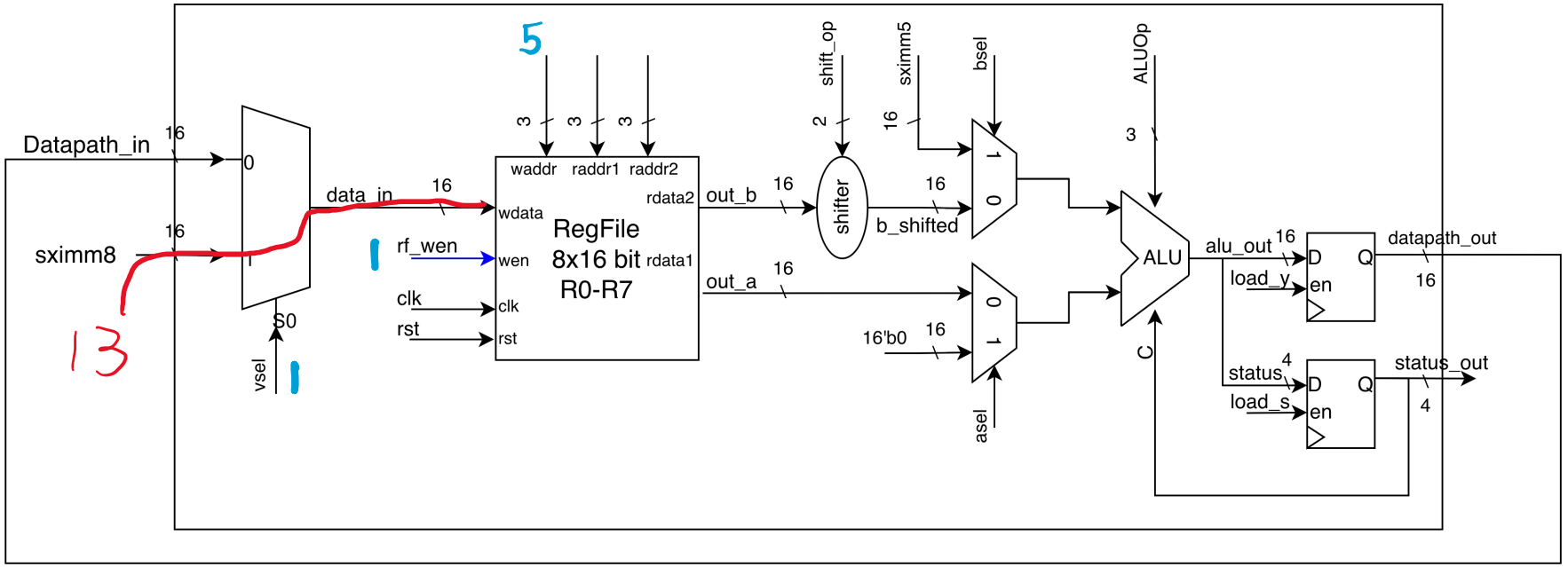
RISKing Machine Datapath




```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Suppose we want to load the actual value “24” into Register 3. We can do this in one cycle!

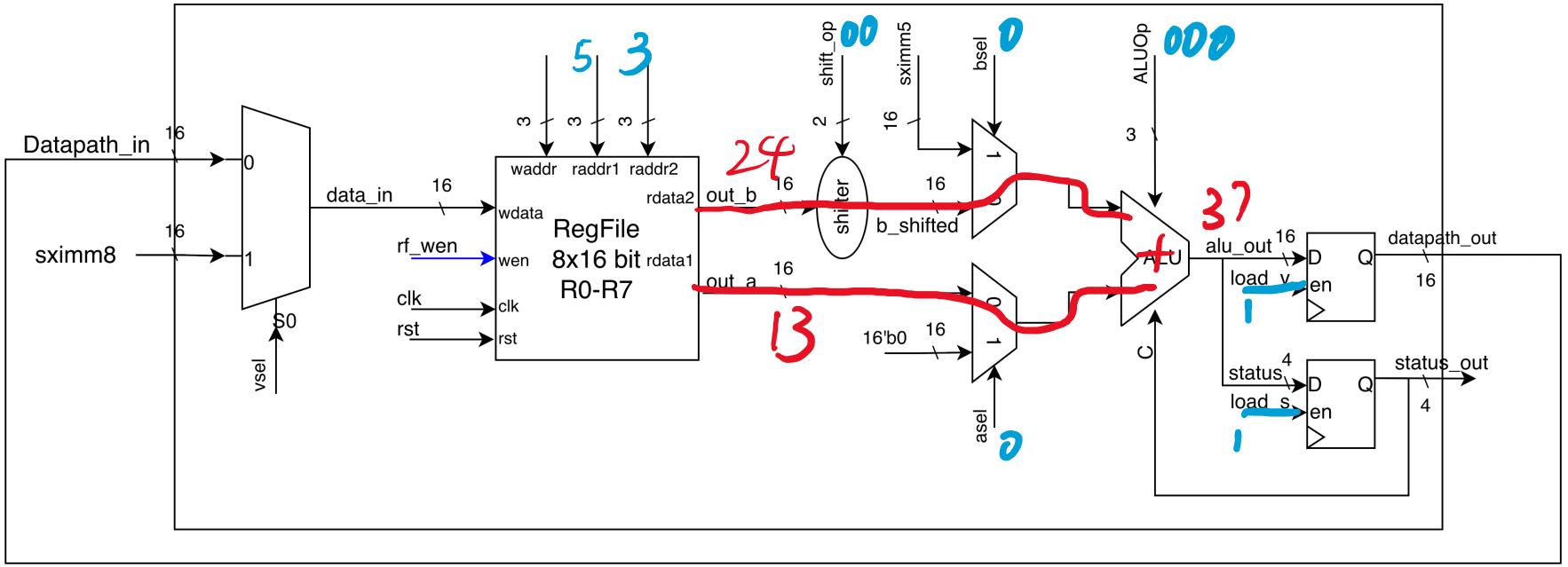
RISKing Machine Datapath




```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Suppose we want to load the actual value “13” into Register 5.
We can do this in one cycle!

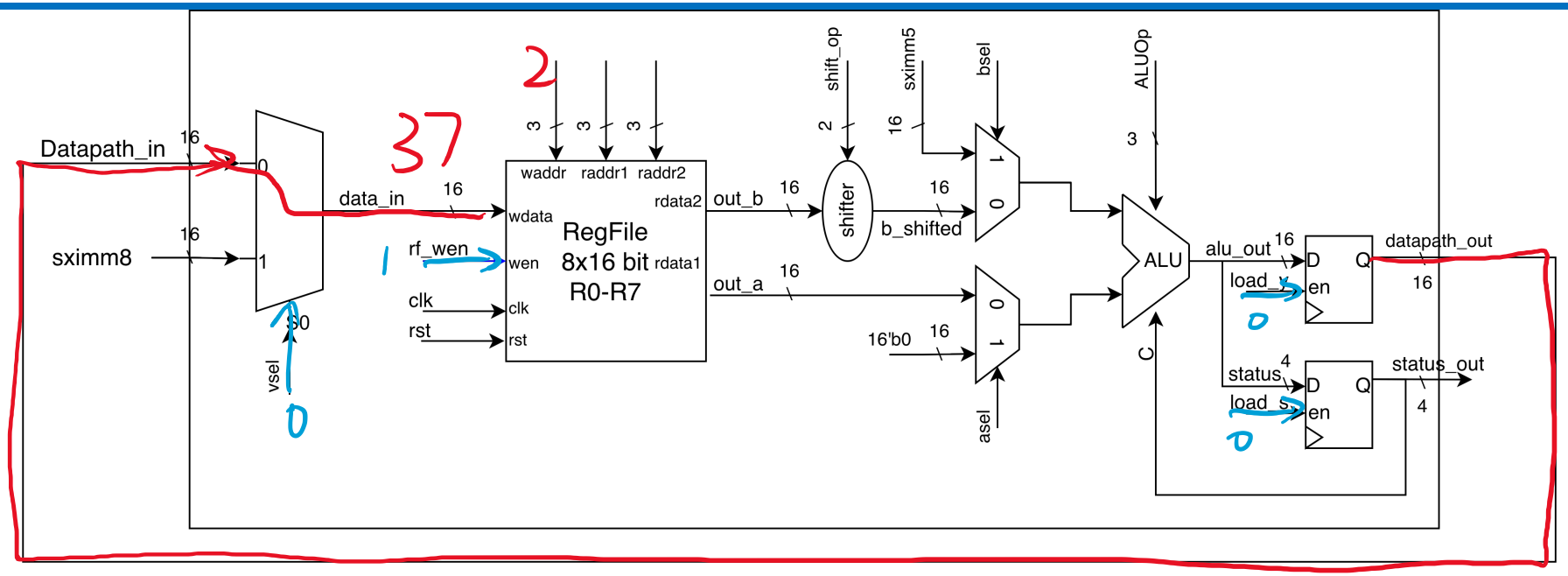
RISKing Machine Datapath




```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Cycle 1: raddr1=5, raddr2=3,
loady = 1, loads=1.

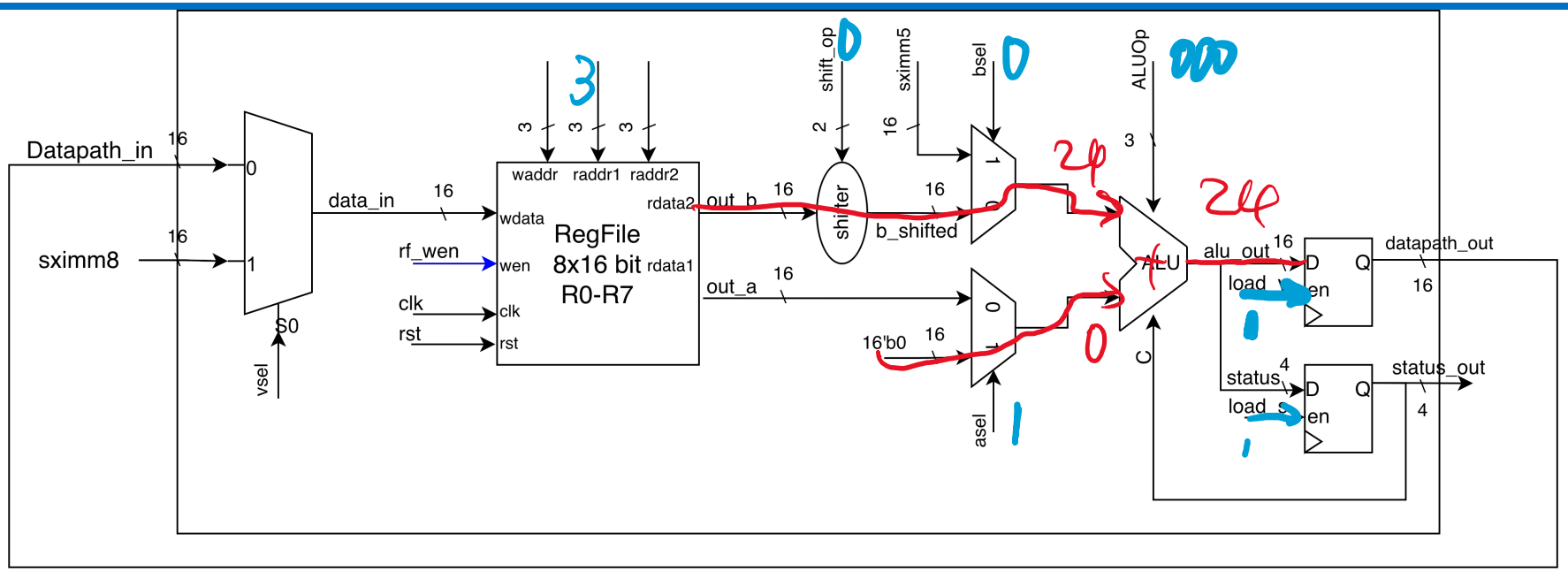
RISKing Machine Datapath




```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Cycle 1: raddr1=5, raddr2=3,
loady = 1, loads=1.
Cycle 2: waddr=2, rf_wen=1
loady = 0, loads=0, vsel=0. Result write into R2.

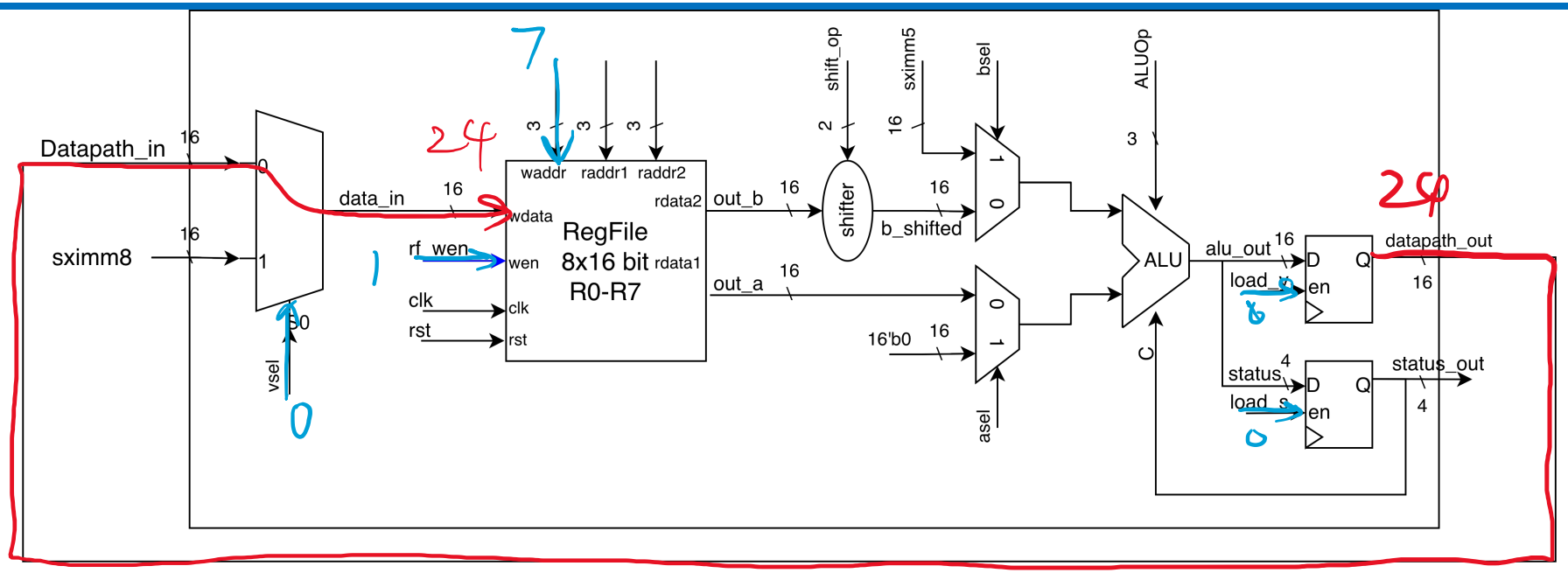
RISKing Machine Datapath




```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Cycle 1: raddr1=3, asel = 1, bsel=0, ALUop=000
Result is R3+0 = R3. loady=1, loads=1

RISKing Machine Datapath

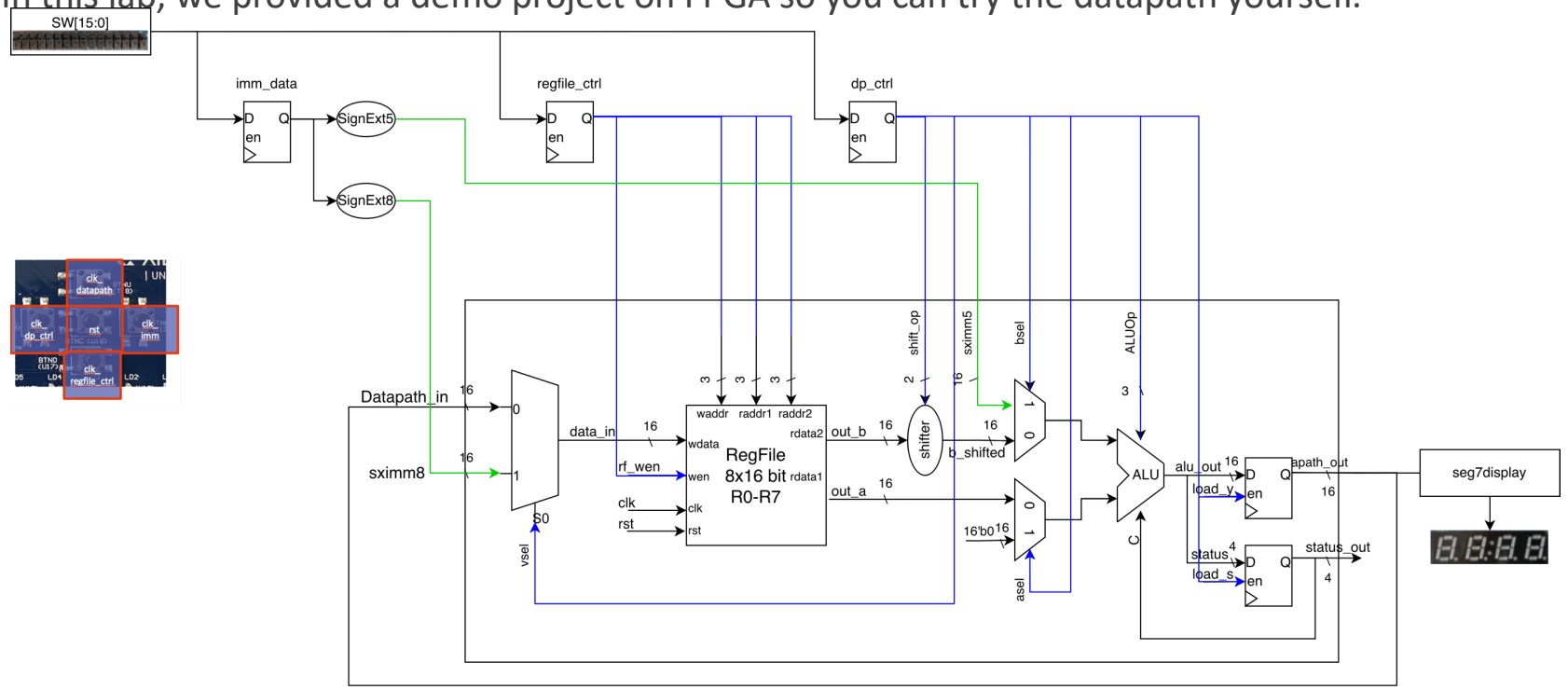



```
MOVI R3,#24  // register R3 contains 24
MOVI R5,#13  // register R5 contains 13
ADD R2,R5,R3 // R2 gets 24 + 13 = 37
MOV R7,R3    // register R7 contains 24
```

Cycle 1: raddr1=3, asel = 1, bsel=0, ALUop=000
Result is R3+0 = R3. loady=1, loads=1
Cycle 2: vsel=0, rf_wen=1, waddr=7
Result is write back to R7

Task 2: Top structure

- In Task 2, you are going to write the datapath.
- In this lab, we provided a demo project on FPGA so you can try the datapath yourself.



Task 2: Top structure

- In Task 2, you are going to write the datapath.
- In this lab, we provided a demo project on FPGA so you can try the datapath yourself.