CS 2200
Lab 3

Today's Topic:
Datapath and Buses

Announcements

- **Project 1** due **Sunday, September 17 @ 11:59 PM**
- **Homework 1** due tonight **@ 11:59 PM** as per extension
- **Homework 2** releases tonight, due **Tuesday, September 12 @ 11:59 PM**
- **Test 1** is next **Wednesday, September 13** in lab
    - **Bring your buzzcard!!**

Test Logistics

- 60 minutes (First hour of lab)
- Open book, open note, open google
  - No AI use or communication
  - Download all notes as PDF (No text editors open)
  - No Honorlock (this time)
- Everything up to and including datapath (No FSM/microcontroller)
- Bring buzzcard for attendance
- ODS: Testing center on 9/13, 9/14

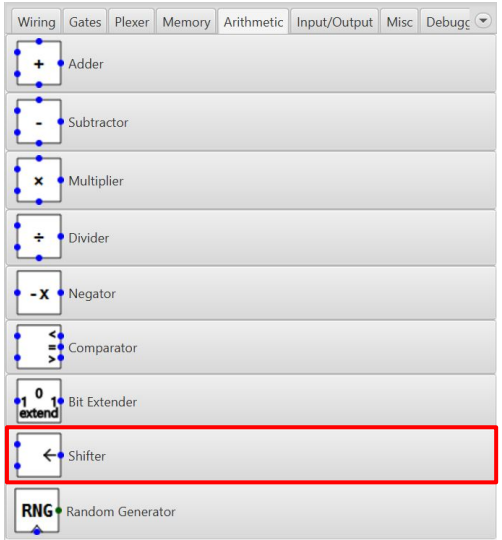
Exercise: ALU Implementation

We want to implement an ALU that can compute all sorts of **bit shifts.** Here's what we want to support:

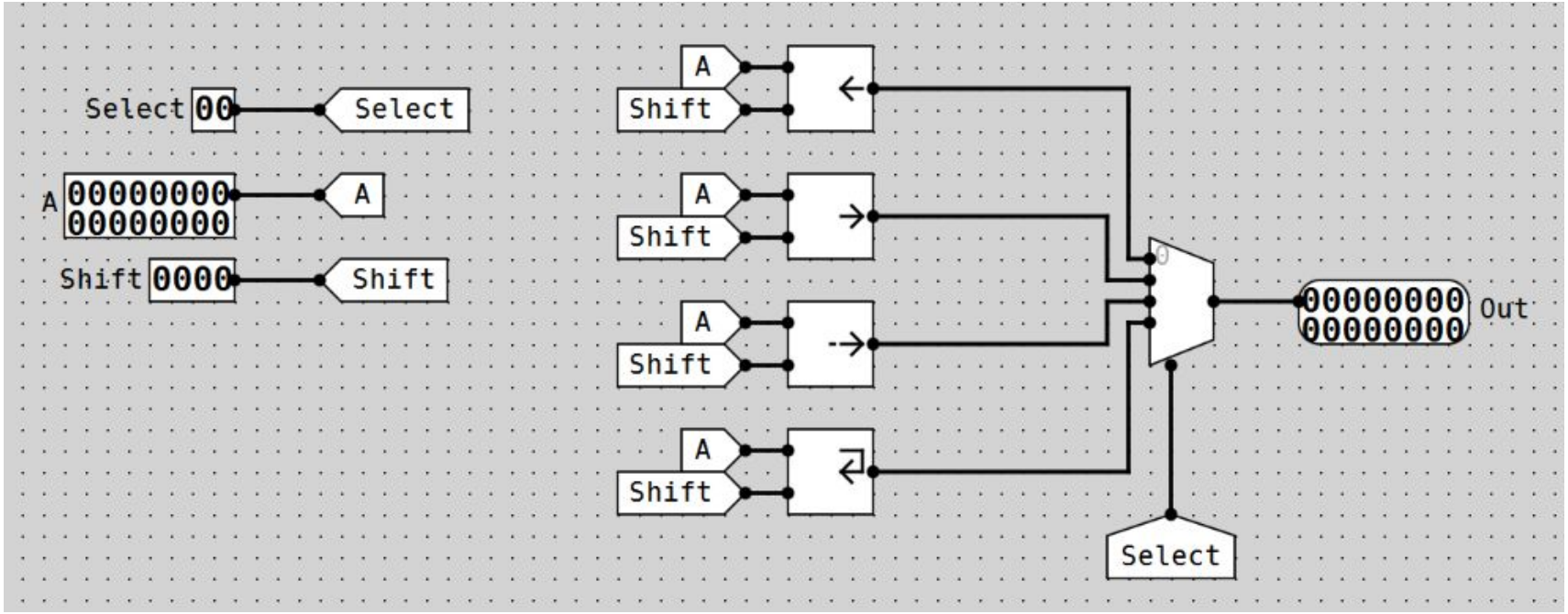
1. Left shift
2. Logical right shift (fill with zeros)
3. Arithmetic right shift (fill with MSB)
4. Left rotate

Need a refresher on what these operations entail? This might help.

Spend **10-15 minutes** implementing this in CircuitSim. Take a look at the **Shifter** component. Assume **16-bit** architecture.

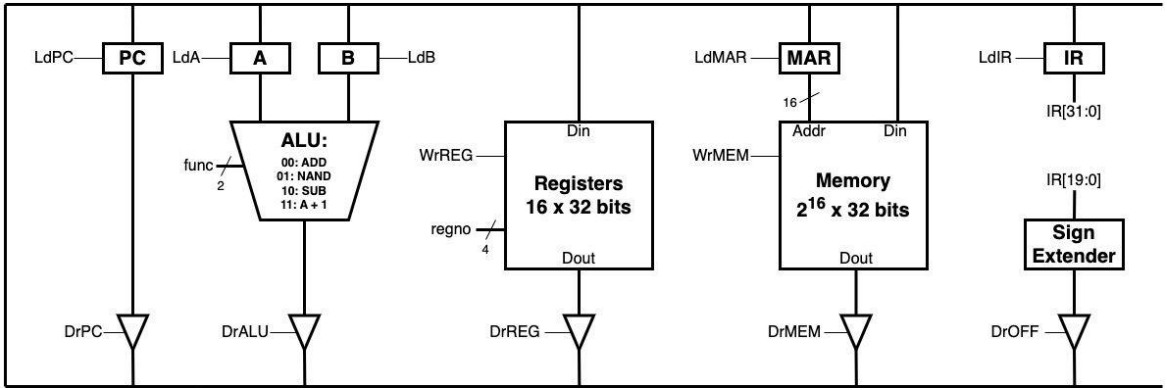


Answer: ALU Implementation



Exercise: Implement NAND on LC-2200 datapath

How many cycles and what control signals are required to implement NAND on the LC-2200 datapath? Spend **5 minutes** tracing each cycle.



Anatomy of an R-type instruction.

Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused

Implementing NAND

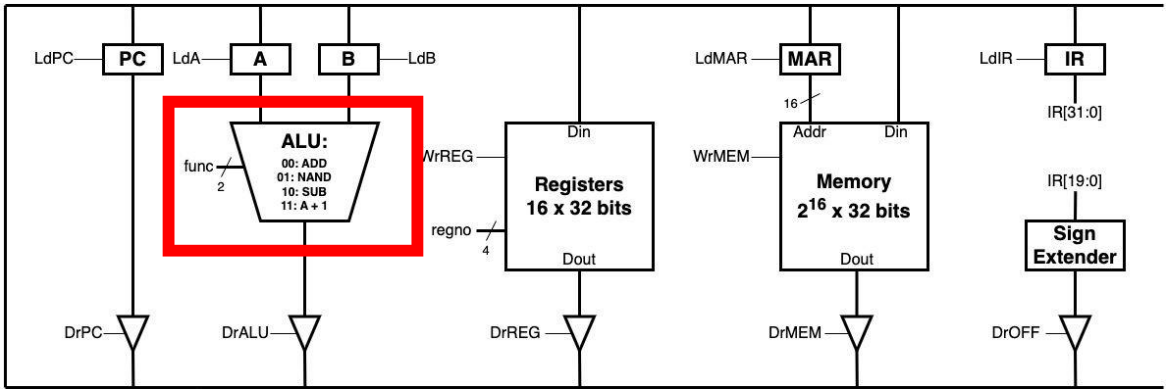
1. Store register Y in A
2. Store register Z in B
3. NAND A and B and store in register X

R-type instructions (add, nand):

bits 31-28:	opcode
bits 27-24:	reg X (destination)
bits 23-20:	reg Y (source)
bits 19-4:	unused (should be all 0s)
bits 3-0:	reg Z (source)

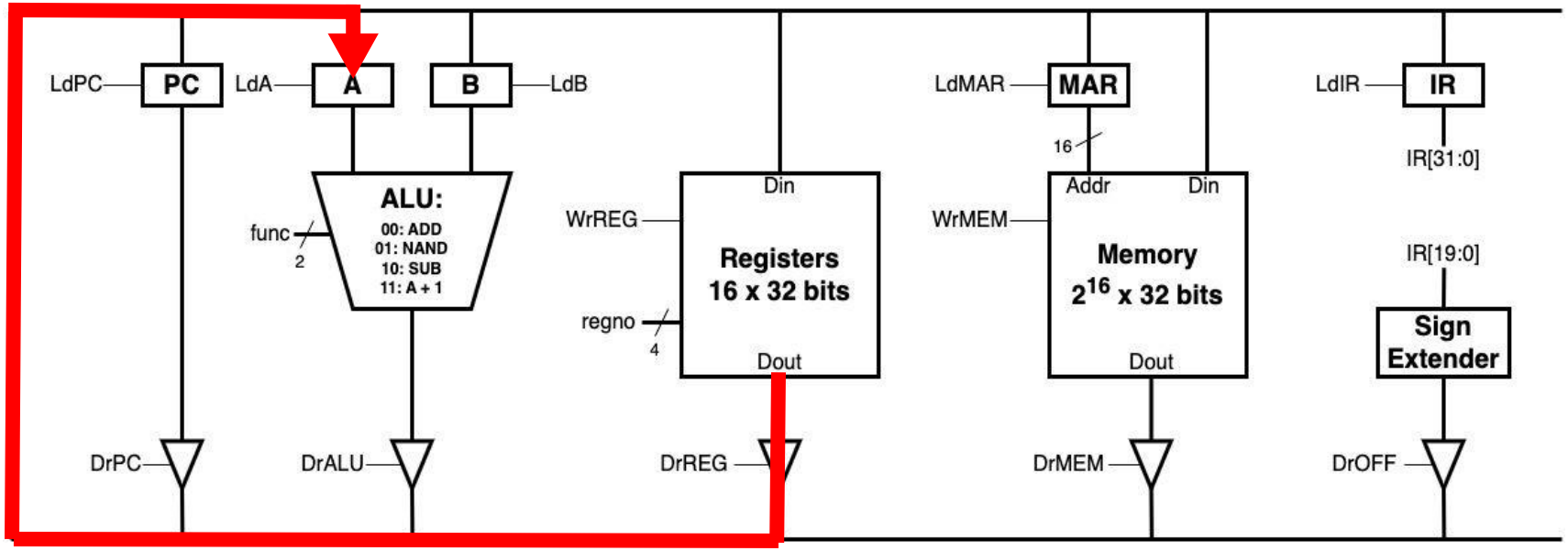
Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused



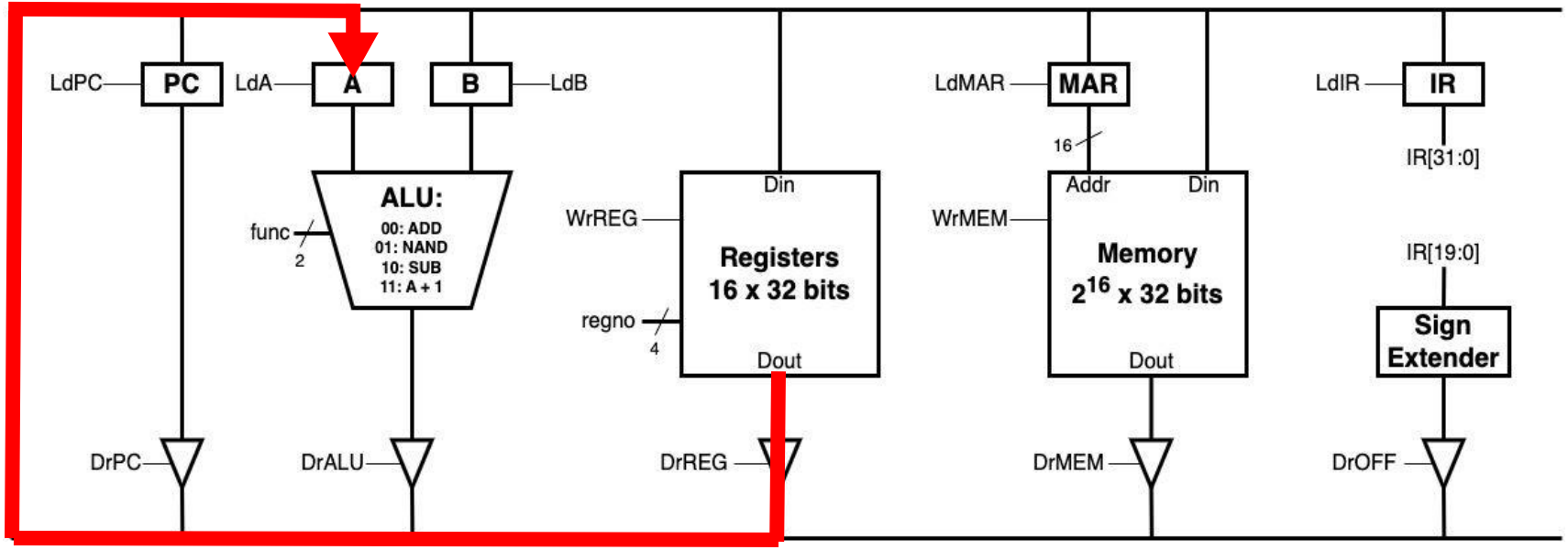
NAND Cycle 1: Ry → A

What signals need to be asserted?



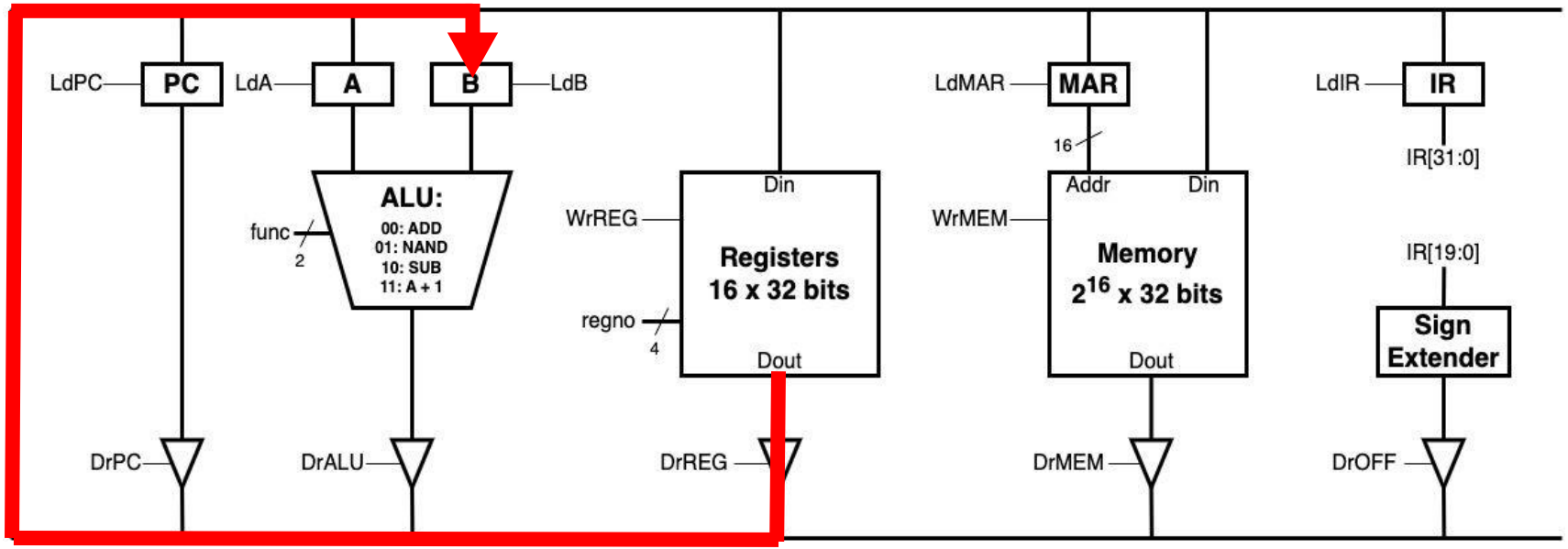
NAND Cycle 1: Ry → A

DrReg, LdA, RegSel = 01



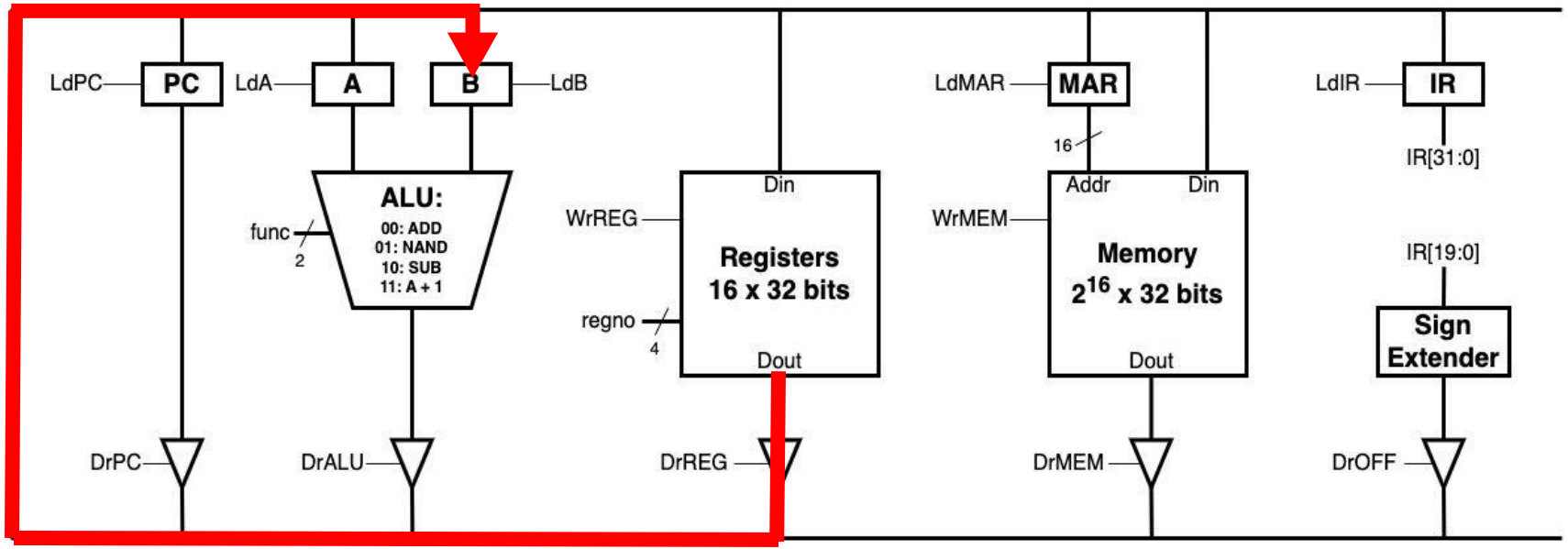
NAND Cycle 2: Rz → B

What signals need to be asserted?



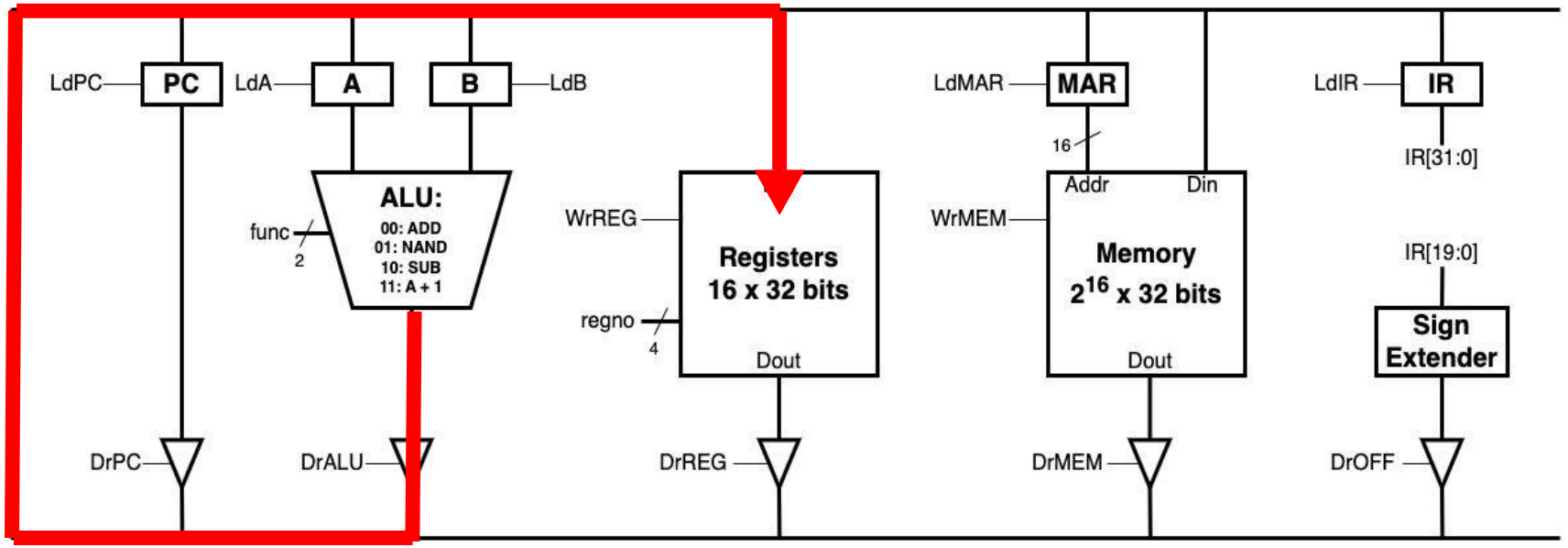
NAND Cycle 2: Rz → B

DrReg, LdB, RegSel = 10



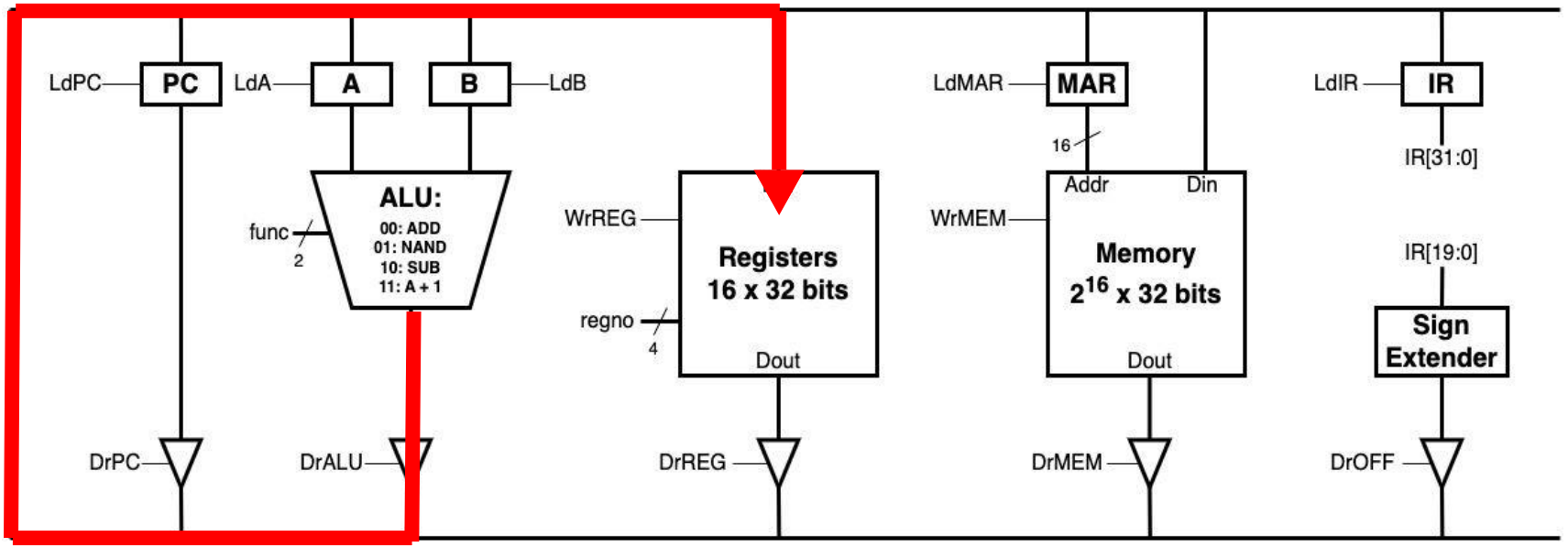
NAND Cycle 3: ALUOut → Rx

What signals need to be asserted?



NAND Cycle 3: ALUOut → Rx

DrALU, WrREG, func = 01, RegSel=00

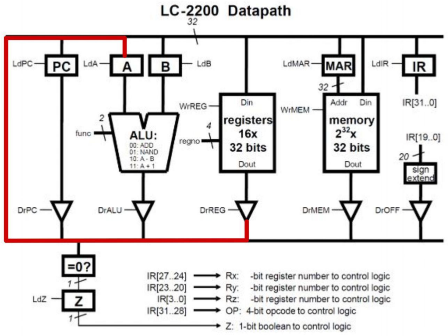


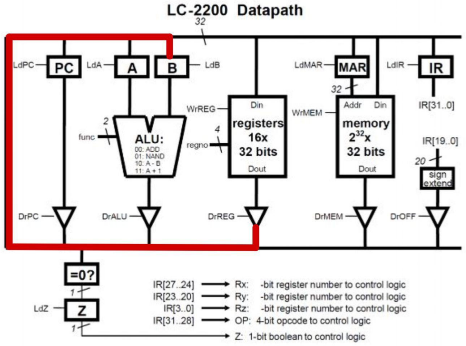
Implementing NAND

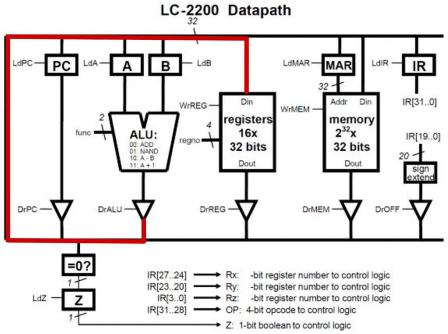
Cycle 1: DrReg, LdA, RegSel = 01

Cycle 2: DrReg, LdB, RegSel = 10

Cycle 3: DrALU, WrReg, func = 01, RegSel=00

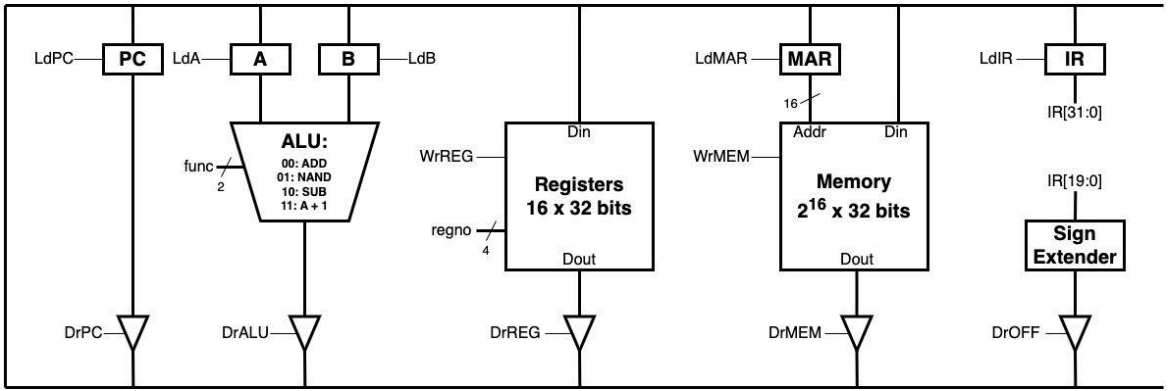






Exercise: Implement JALR on LC-2200 datapath

How many cycles and what control signals are required to implement JALR on the LC-2200 datapath? Spend **5 minutes** tracing each cycle.



Anatomy of a J-type instruction.

Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused

Implementing JALR

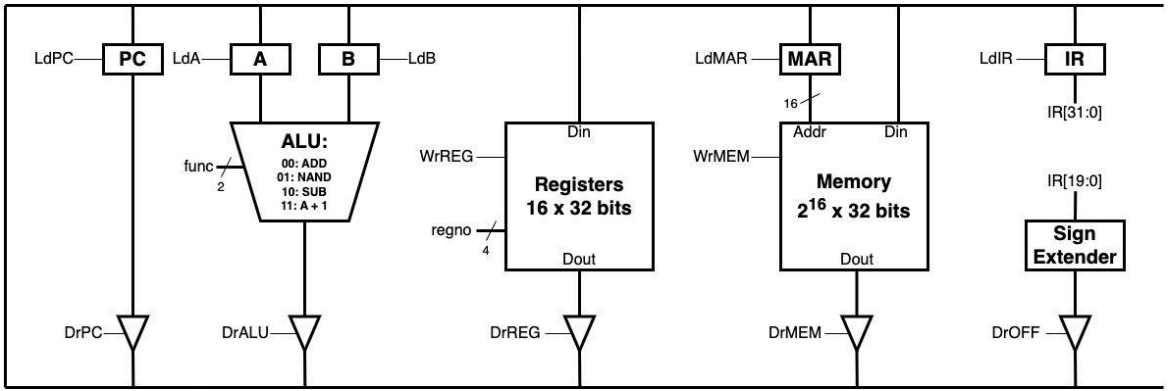
Store PC + 1 in Ry
Branch to the address in Rx

J-type instructions (jalr):[6]

bits 31-28:	opcode
bits 27-24:	reg X (target of the jump)
bits 23-20:	reg Y (link register)
bits 19-0:	unused (should be all 0s)

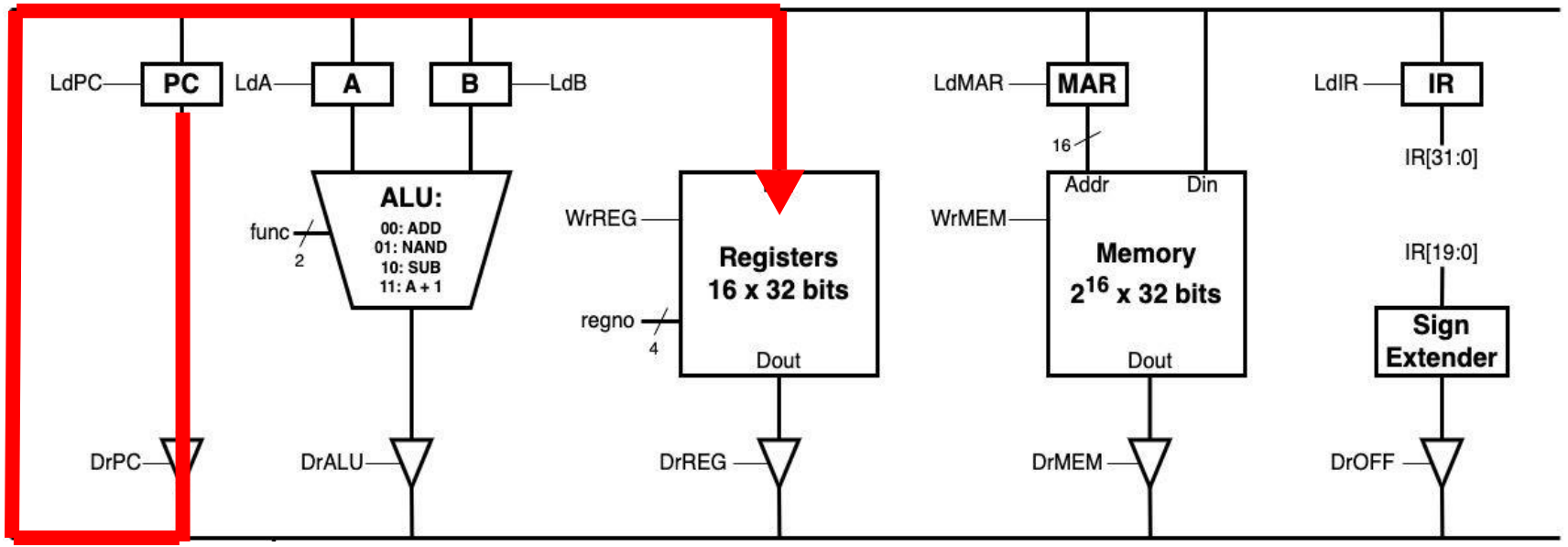
Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused



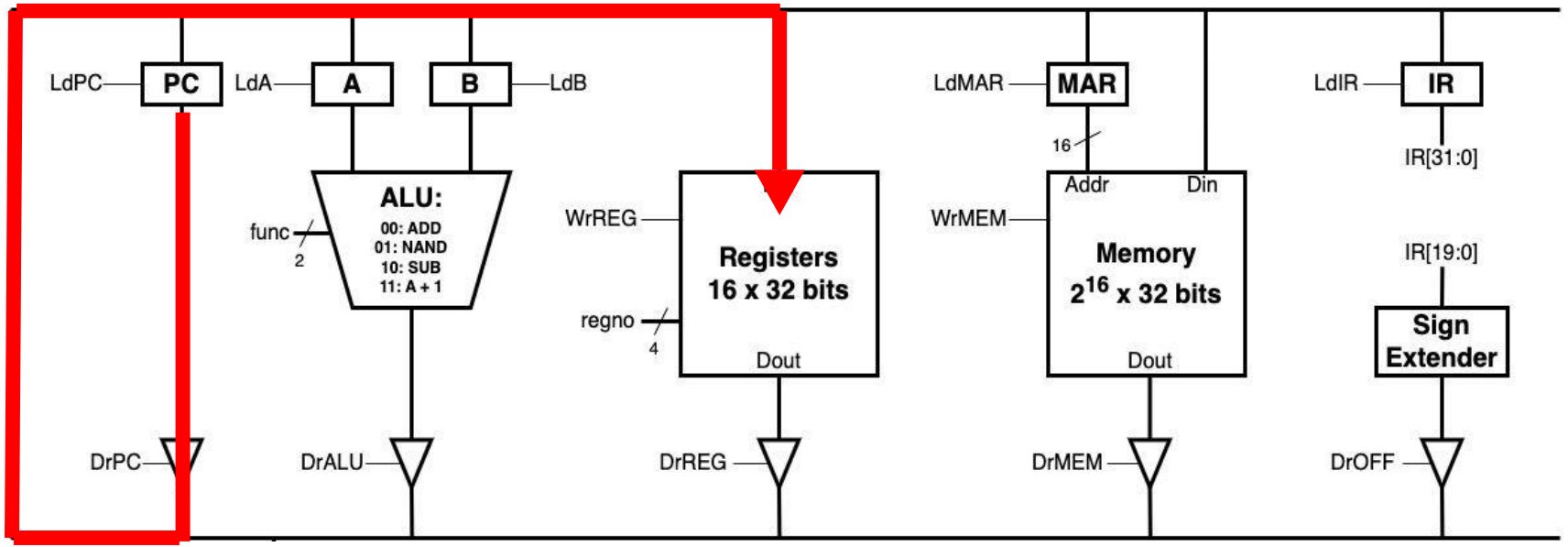
JALR Cycle 1: PC → Ry

What signals need to be asserted?



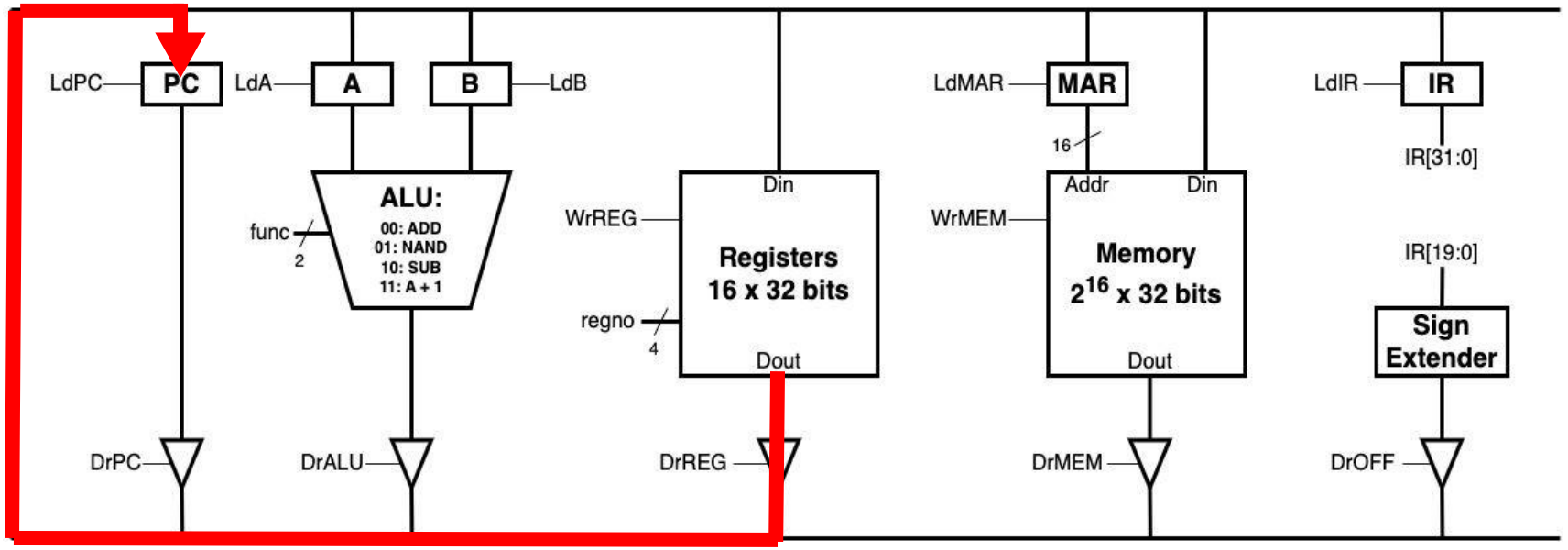
JALR Cycle 1: PC → Ry

DrPC, WrREG, RegSel = 01



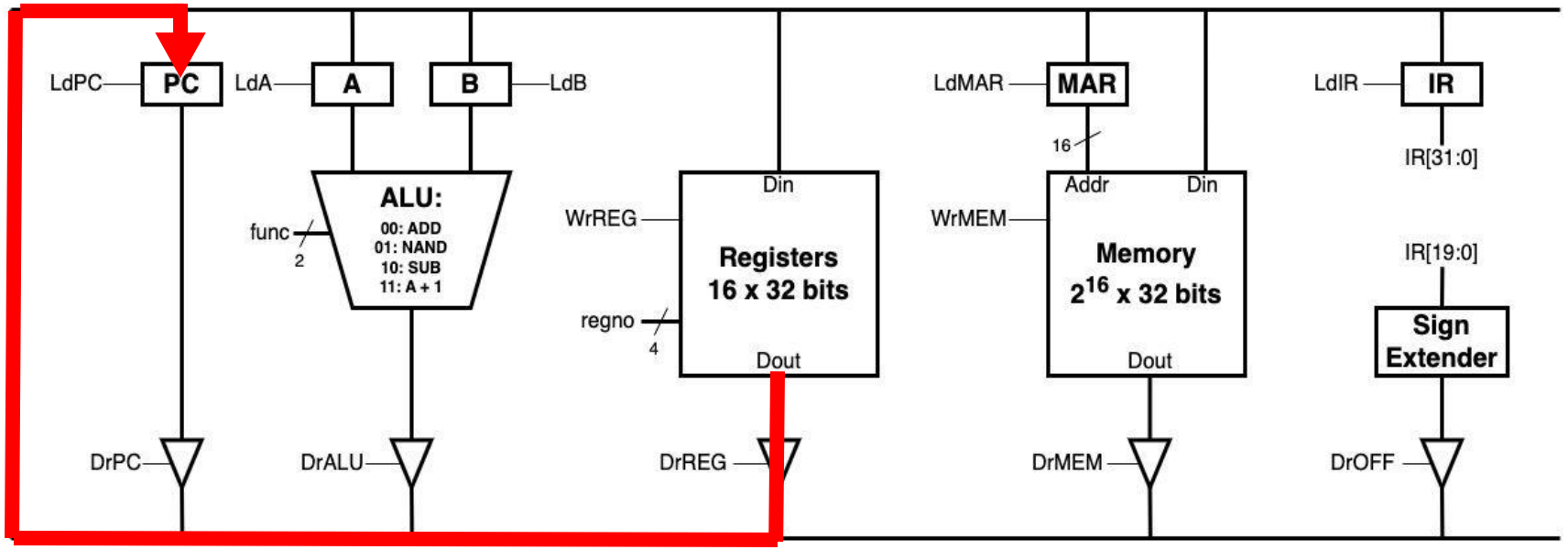
JALR Cycle 2: Rx → PC

What signals need to be asserted?



JALR Cycle 2: Rx → PC

DrREG, LdPC, regSel = 00

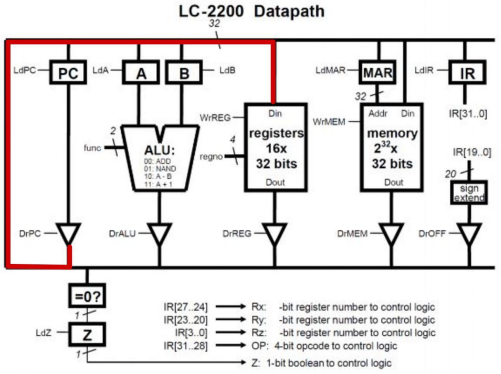


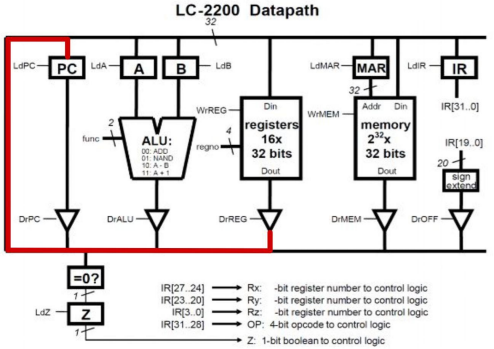
Implementing JALR

Ry ← PC + 1 *
PC ← Rx

Cycle 1: DrPC, WrREG, regSel = 01
Cycle 2: DrREG, LdPC, regSel = 00

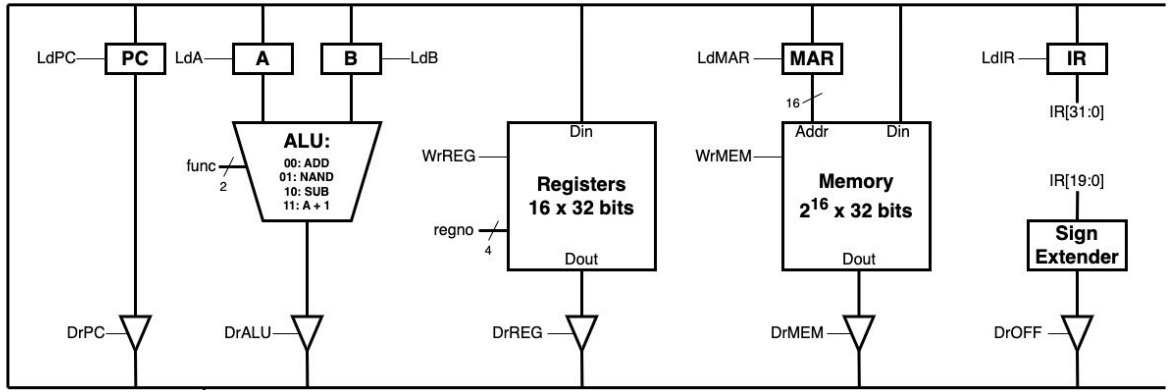
Note: By the time an instruction executes, the PC has *already* been incremented! We don't have to do that again during the execution macrostate.





Exercise: Implement ADDI on LC-2200 datapath

How many cycles and what control signals are required to implement ADDI on the LC-2200 datapath? Spend **5 minutes** tracing each cycle.



Anatomy of an I-type instruction.

Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused

Implementing ADDI

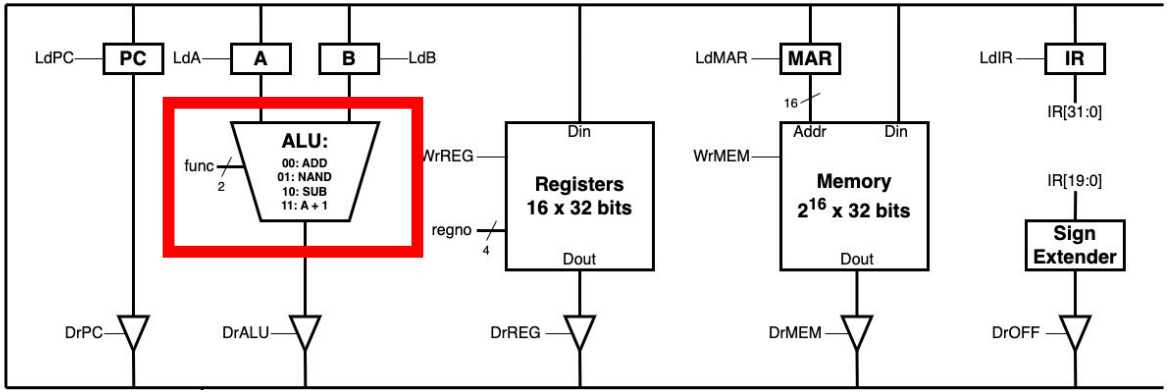
1. Load Ry into A
2. Load sign extended offset into B
3. Add A and B and store in Rx

I-type instructions (addi, lw, sw, beq):

bits 31-28: opcode
bits 27-24: reg X
bits 23-20: reg Y
bits 19-0: Immediate value or address offset

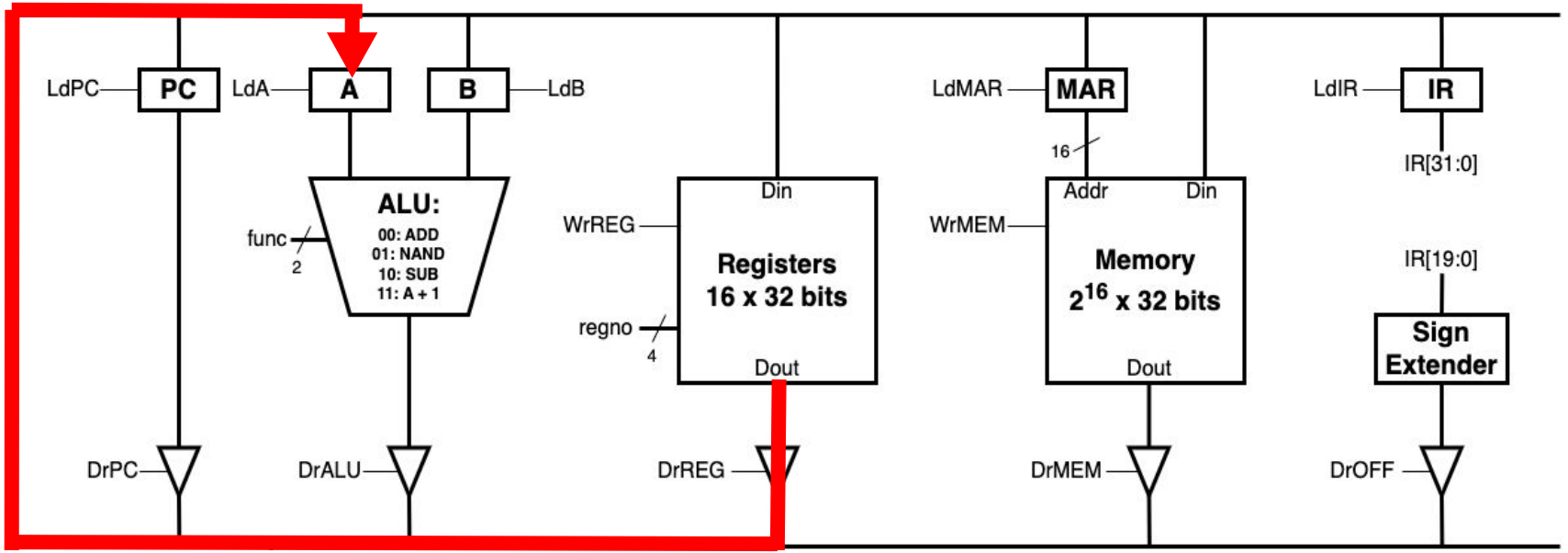
Table 4: Register Selection Map

RegSelHi	RegSelLo	Register
0	0	RX (IR[27:24])
0	1	RY (IR[23:20])
1	0	RZ (IR[3:0])
1	1	unused



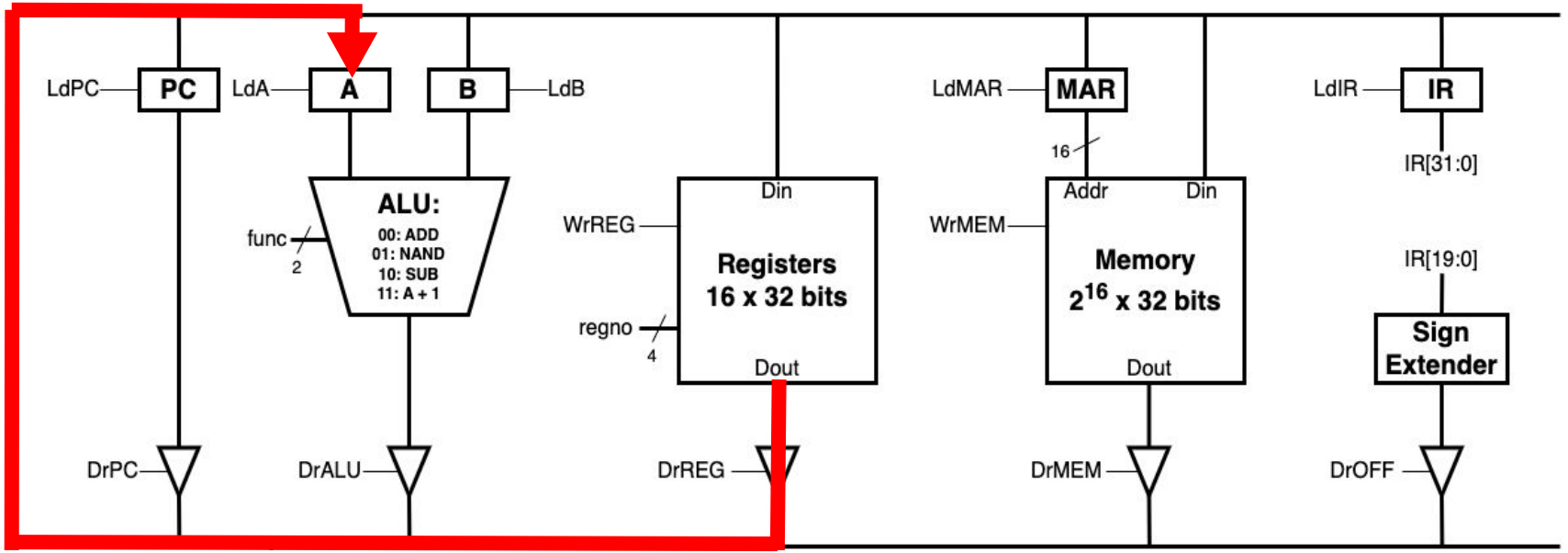
ADDI Cycle 1: Ry → A

What signals need to be asserted?



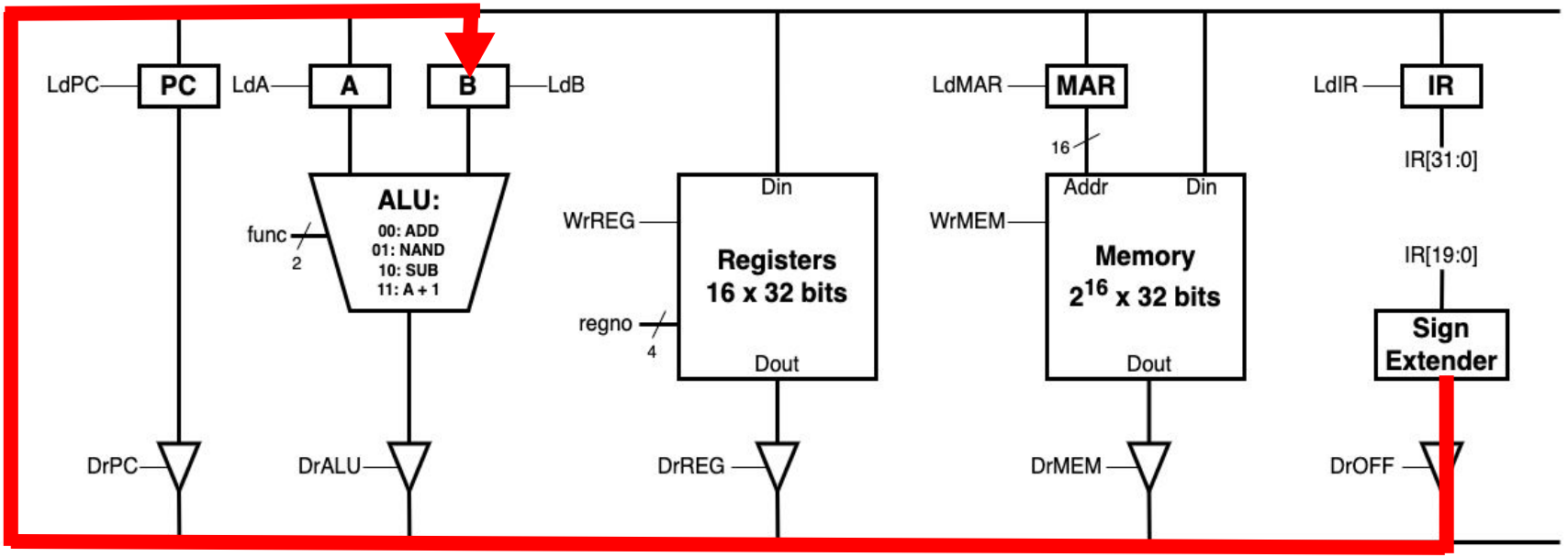
ADDI Cycle 1: Ry → A

DrREG, LdA, RegSel = 01



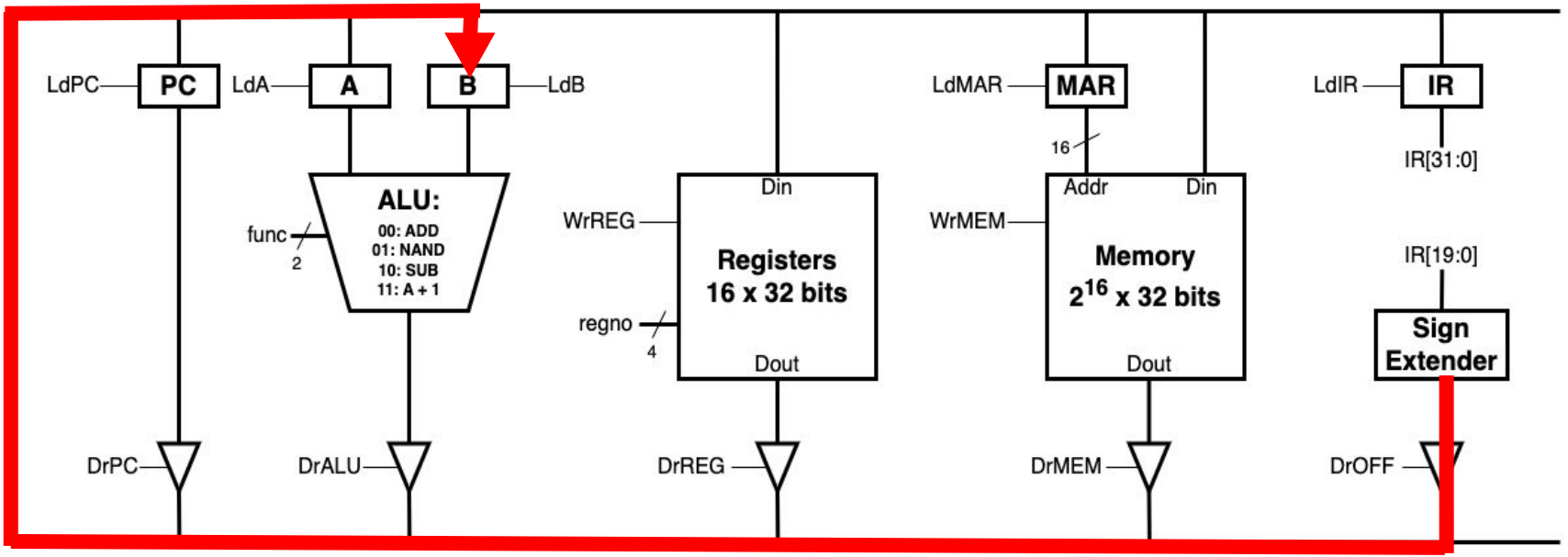
ADDI Cycle 2: OFF → B

What signals need to be asserted?



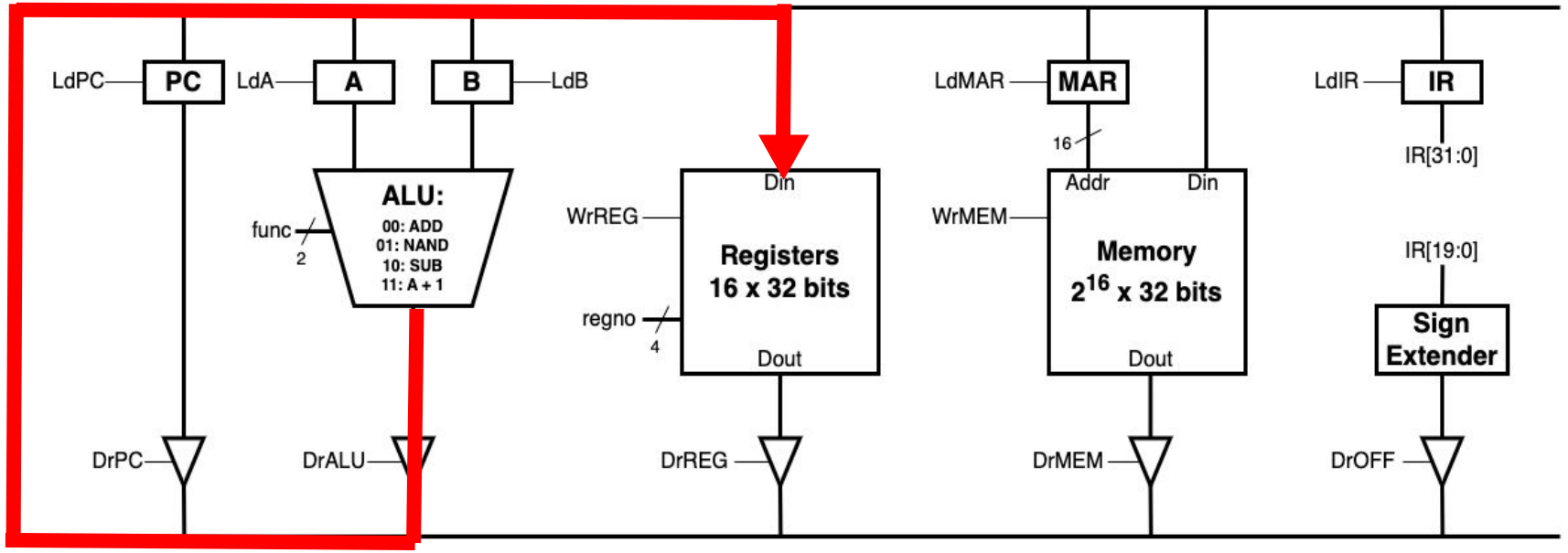
ADDI Cycle 2: OFF → B

DrOFF, LdB



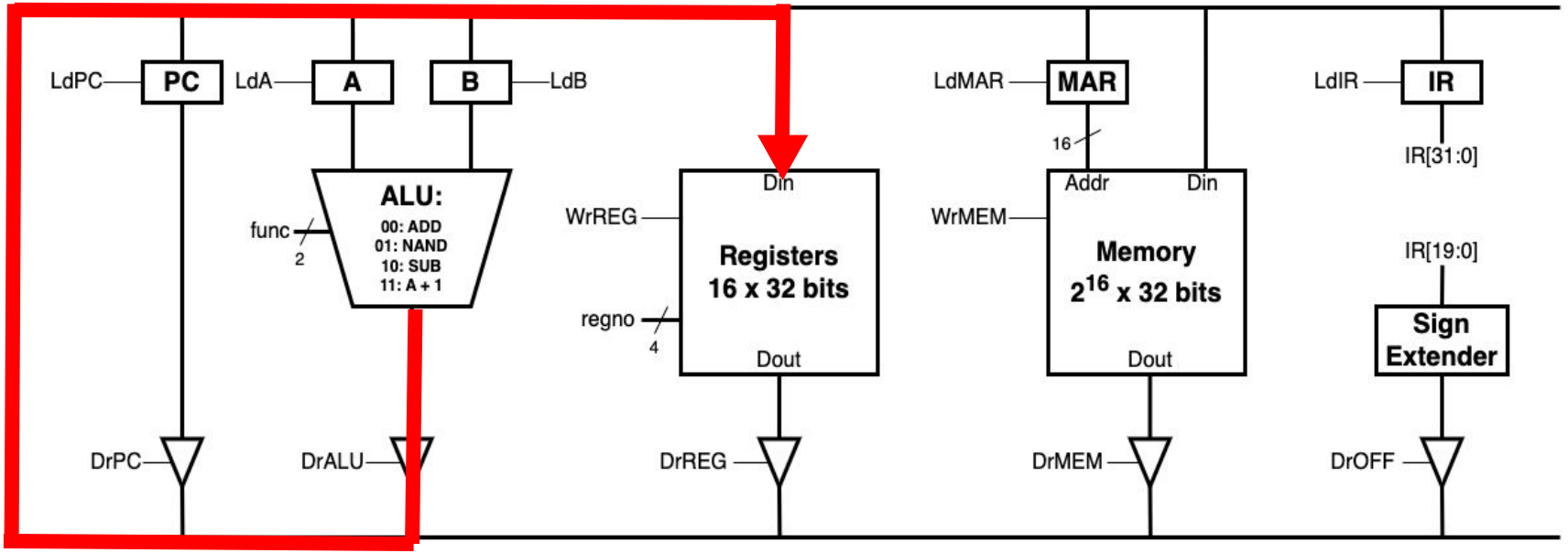
ADDI Cycle 3: ALUout → Rx

What signals need to be asserted?



ADDI Cycle 3: ALUout → Rx

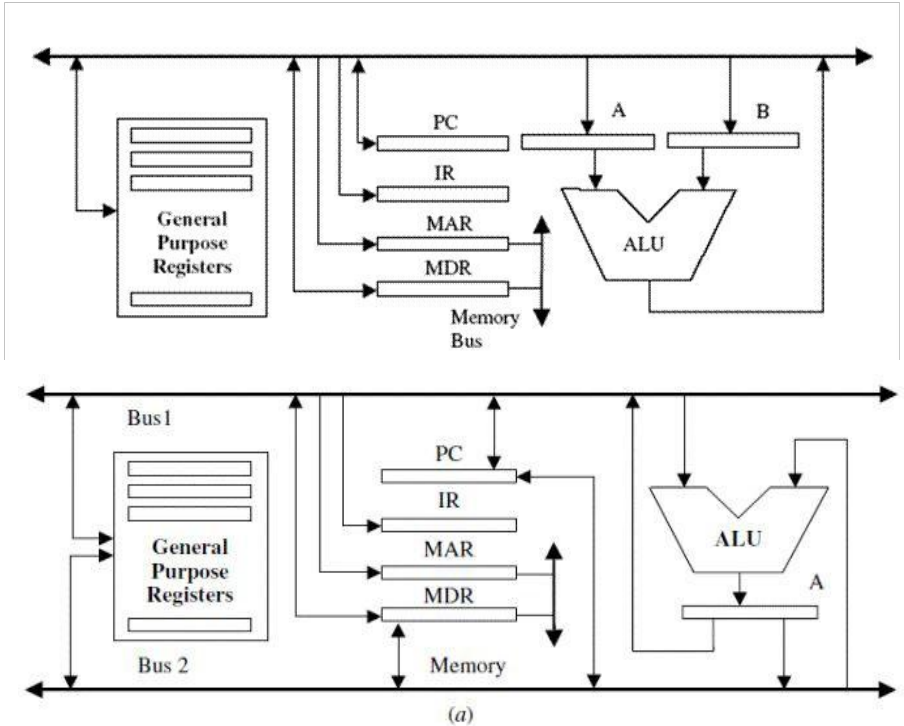
func = 00, DrALU, WrREG, RegSel = 00



Datapath Design

Single vs. Double Bus Design

- **Single Bus Design**
  - One bus wraps around the entire datapath
  - Components take turns to orderly communicate
  - More clock cycles per instruction
- **Double Bus Design**
  - Two sets of bus wires.
  - Two sets of data can communicate simultaneously.
  - More hardware = greater price and heat



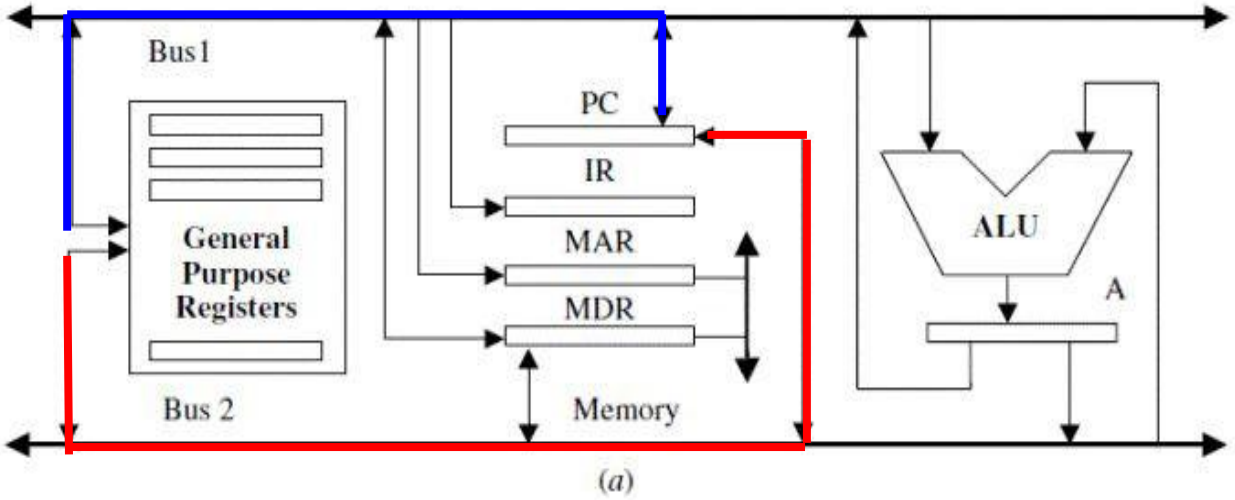
The Power of 2 Buses: JALR example

Now in 1 cycle:

- Ry ← PC + 1
- PC ← Rx

DrPCBus1, LdReg,
RegSelectIn = 01

RegSelectOut = 00,
DrRegBus2,
LdPC



Using the Bus

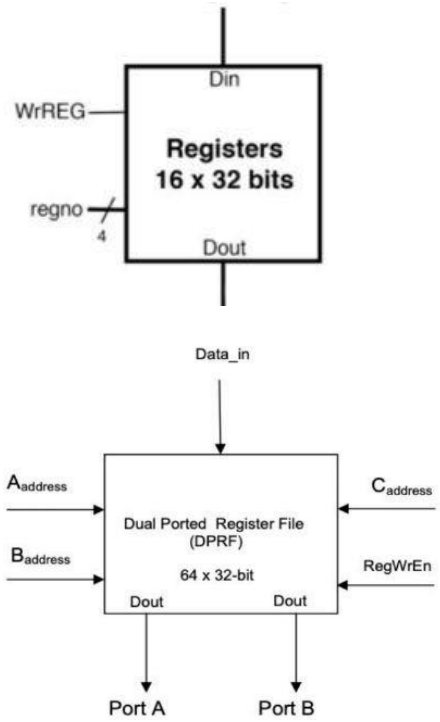
Regardless of bus design, we need the following for the datapath to work.

- **One component outputs data on one bus at one time.** Every other component will be listening to the output on that bus.
- We use tri-state buffers (bus drivers) to ensure that only one component is outputting to the bus at a time. **The tri-state buffers are hooked up to the DrX control signals.**
- We use the register “enable” input to ensure that only devices which want to store the data do so. **This is done using the LdXX control signals.**

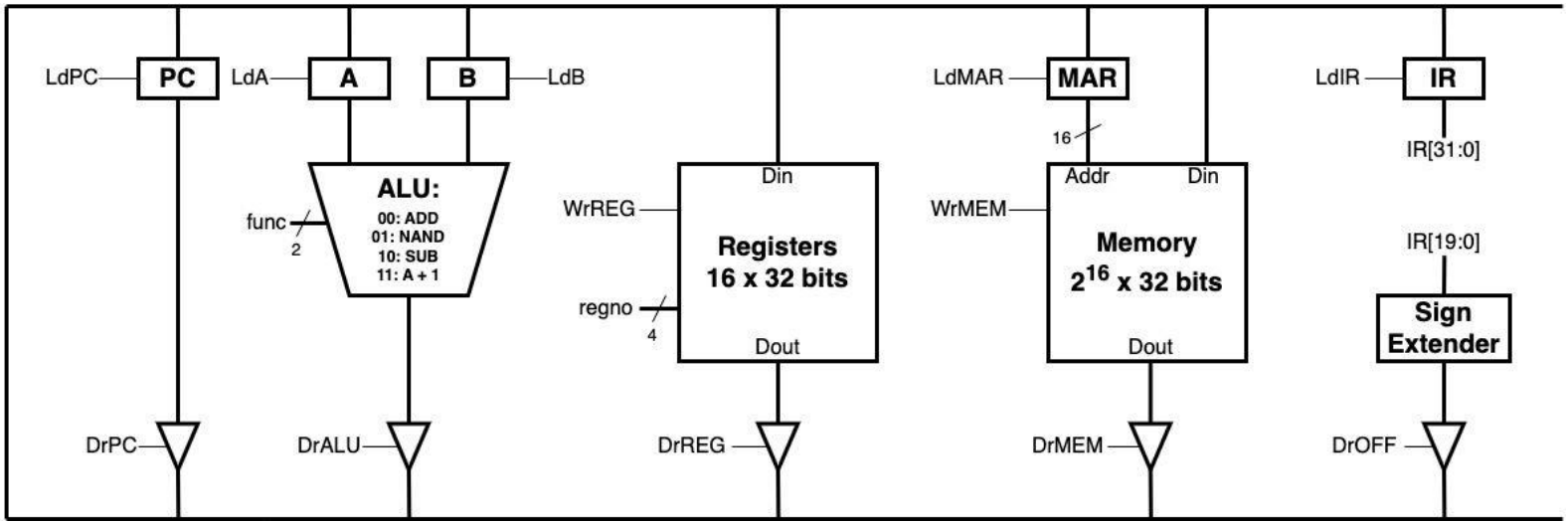
Single vs. Dual Ported Register File

Contains a collection of registers visible to the programmer.

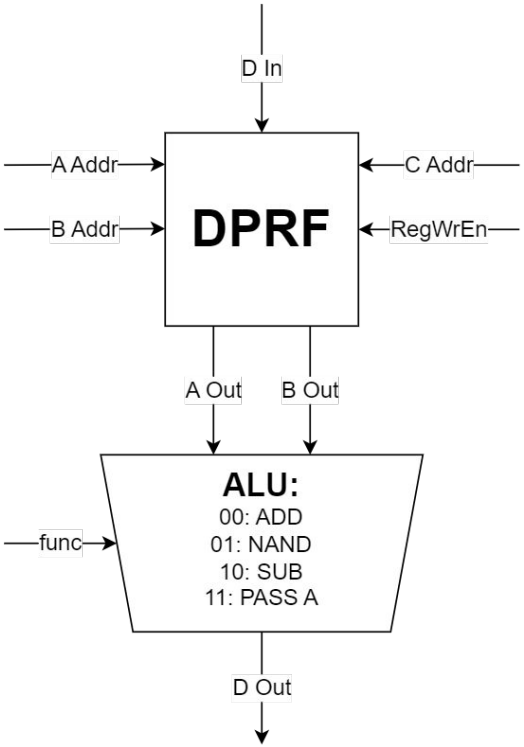
- **Single Ported Register File**
  - Allows for a single register read at a time
  - The LC-2200 register file is single-ported!
- **Dual Ported Register File**
  - Allows for two register reads at a time
  - Requires additional address input
  - Can still only write to one register at a time

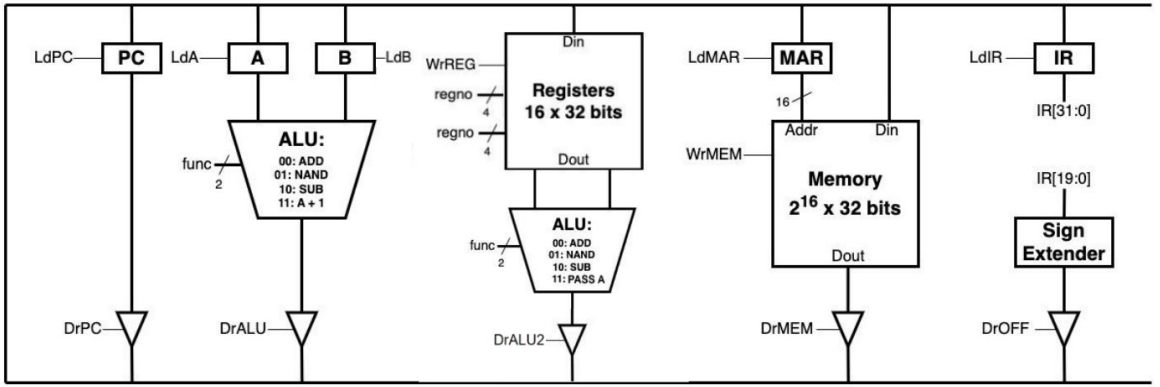


Exercise: What changes would we have to make to the datapath to support a DPRF?



Answer: Adding a 2nd ALU makes things work out!





2200

Number of the day

Project 1 Q&A

Thanks for attending!