# ENGN1640: Design of Computing Systems

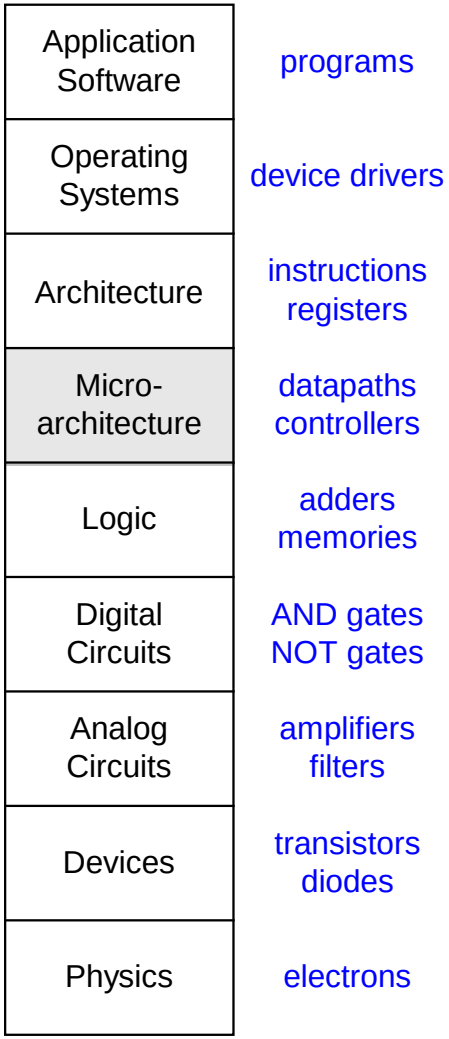
## Topic 04: Single-Cycle Processor Design

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2016

[ material from Harris ]

# Processor organization (microarchitecture)

- Multiple implementations for a single architecture:
  - Single-cycle
    - Each instruction executes in a single cycle
  - Multi-cycle
    - Each instruction is broken up into a series of shorter steps
  - Pipelined
    - Each instruction is broken up into a series of steps
    - Multiple instructions execute at once.
- Superscalar
    - Multiple instructions fetched, decoded and executed simultaneously.

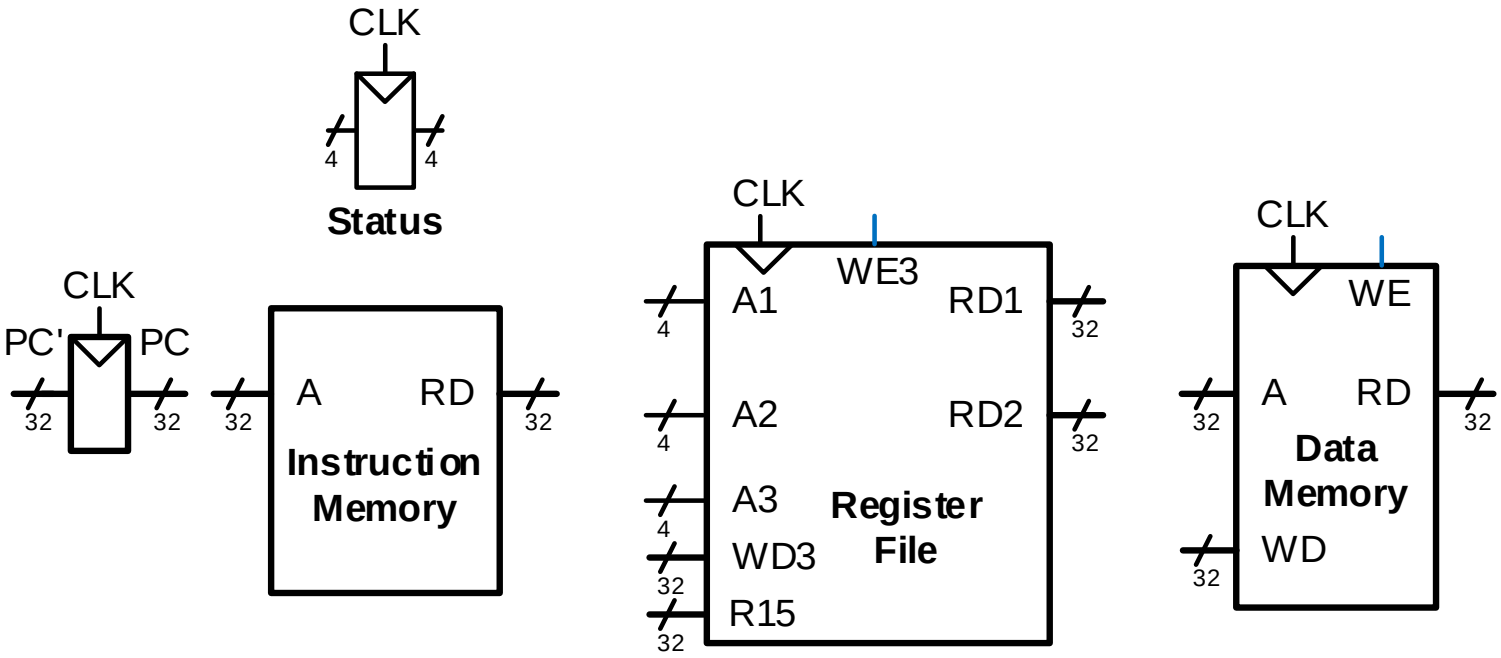


# Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine a number of ARM implementations
  - A simplified single-cycle version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: `LDR`, `STR`
  - Arithmetic/logical: `ADD, SUB, AND, ORR`
  - Control transfer: `B`
- Will not implement in lab post-index, pre-index or shifted operands.

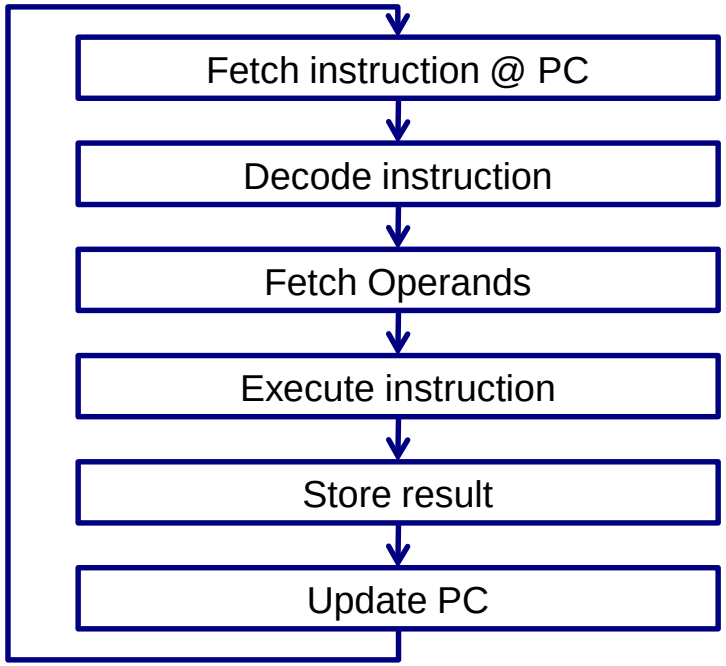
# Architectural state

- Determines everything about a processor:
  - PC
  - 16 registers
  - Memory



# Single-Cycle ARM Processor

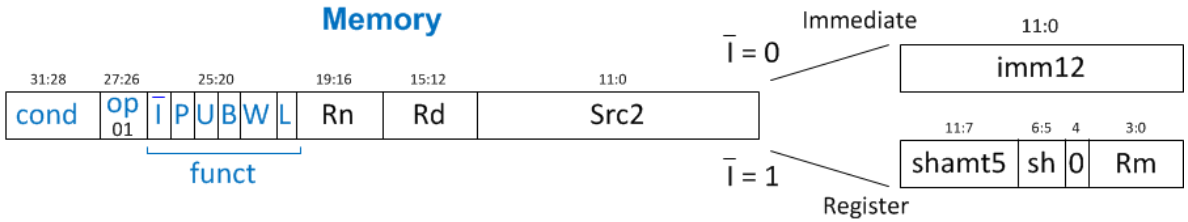
- Datapath
- Control




**Datapath:** start with LDR instruction

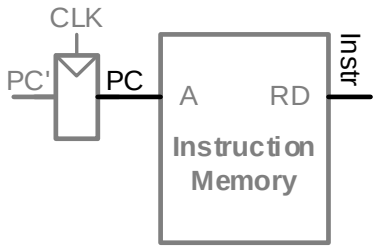
- **Example:**

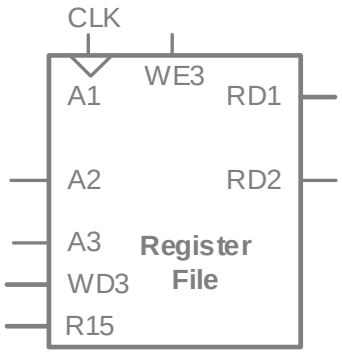
```
LDR R1, [R2, #5]
LDR Rd, [Rn, imm12]
```

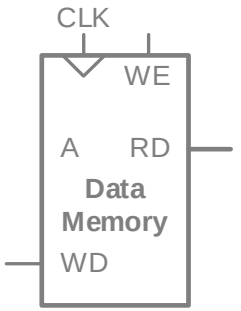


# Single-Cycle Datapath: LDR fetch

- **STEP 1:** Fetch instruction

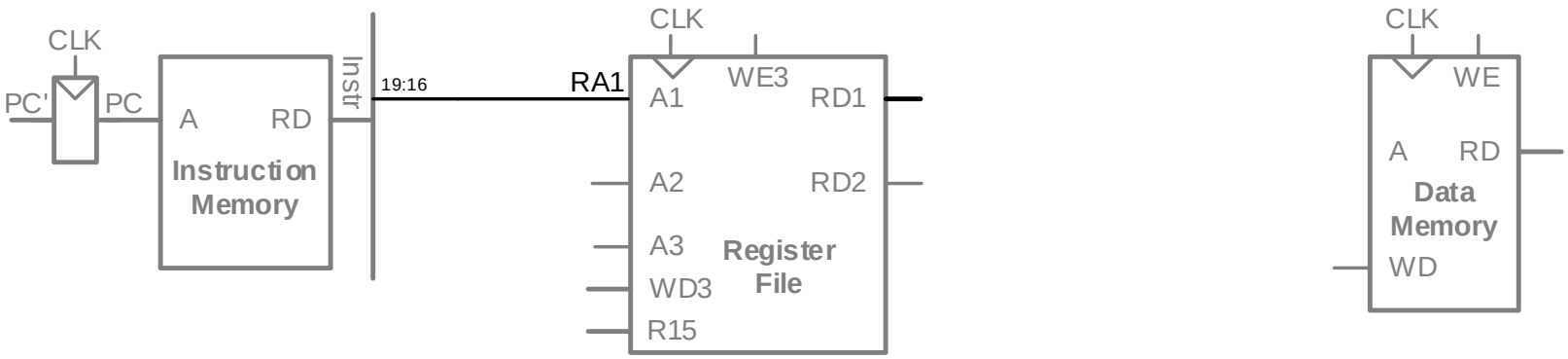


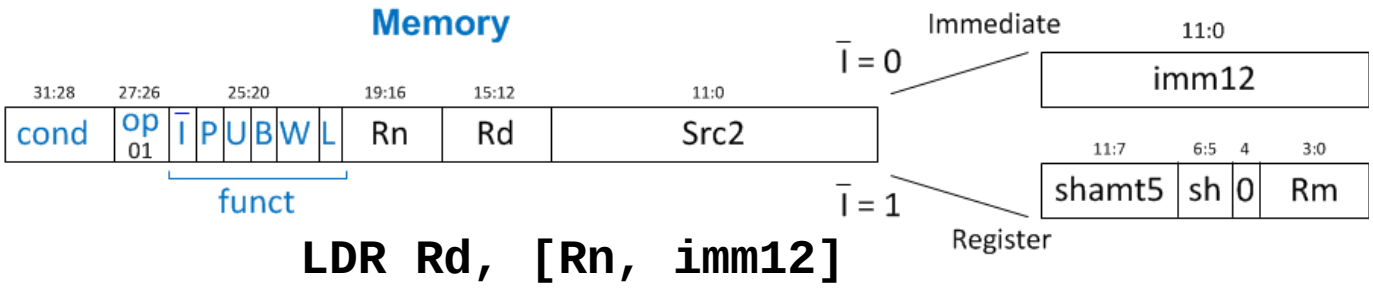



# Single-Cycle Datapath: LDR register read

## **STEP 2:** Read source operands from RF

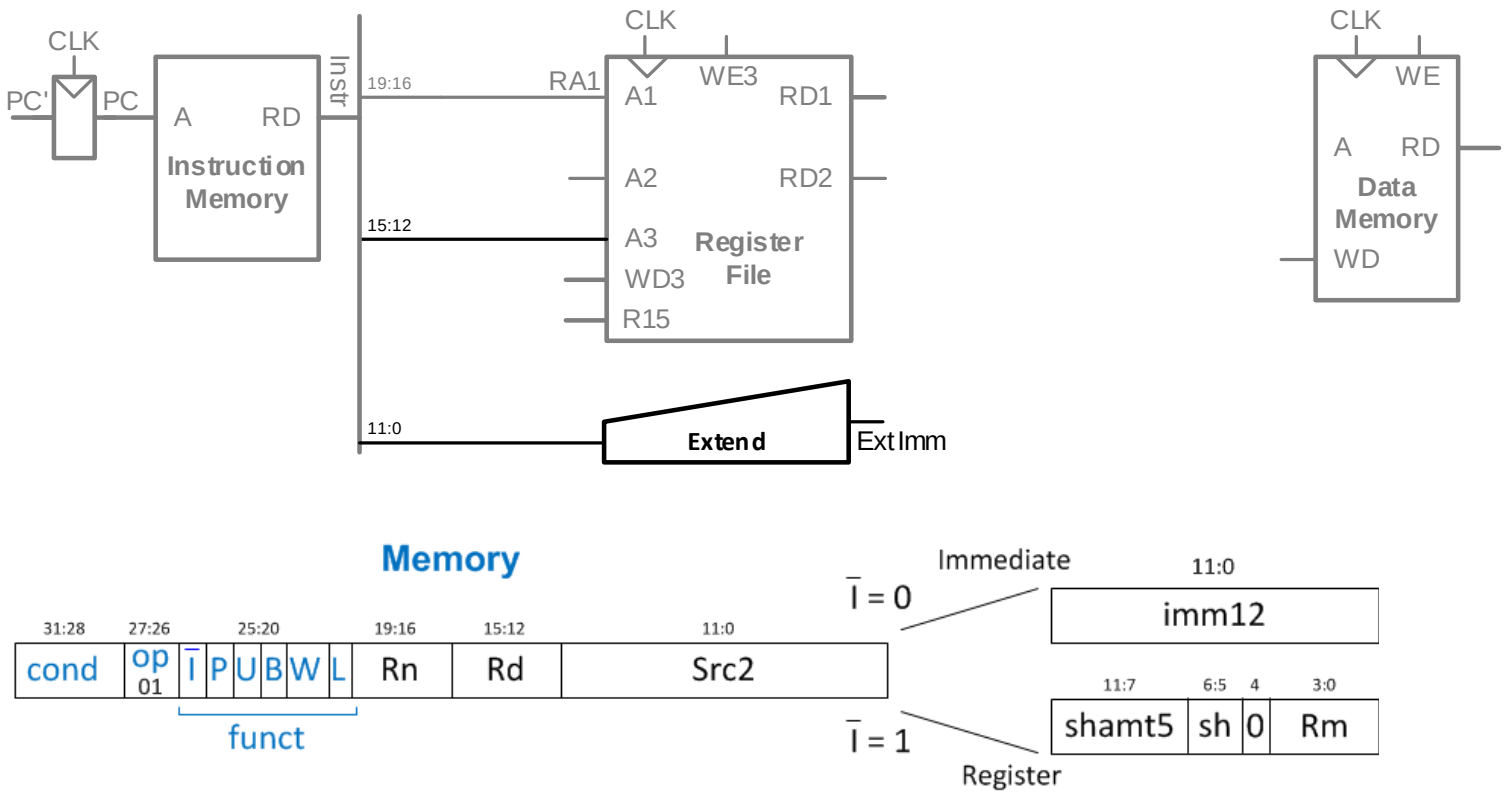



**LDR Rd, [Rn, imm12]**

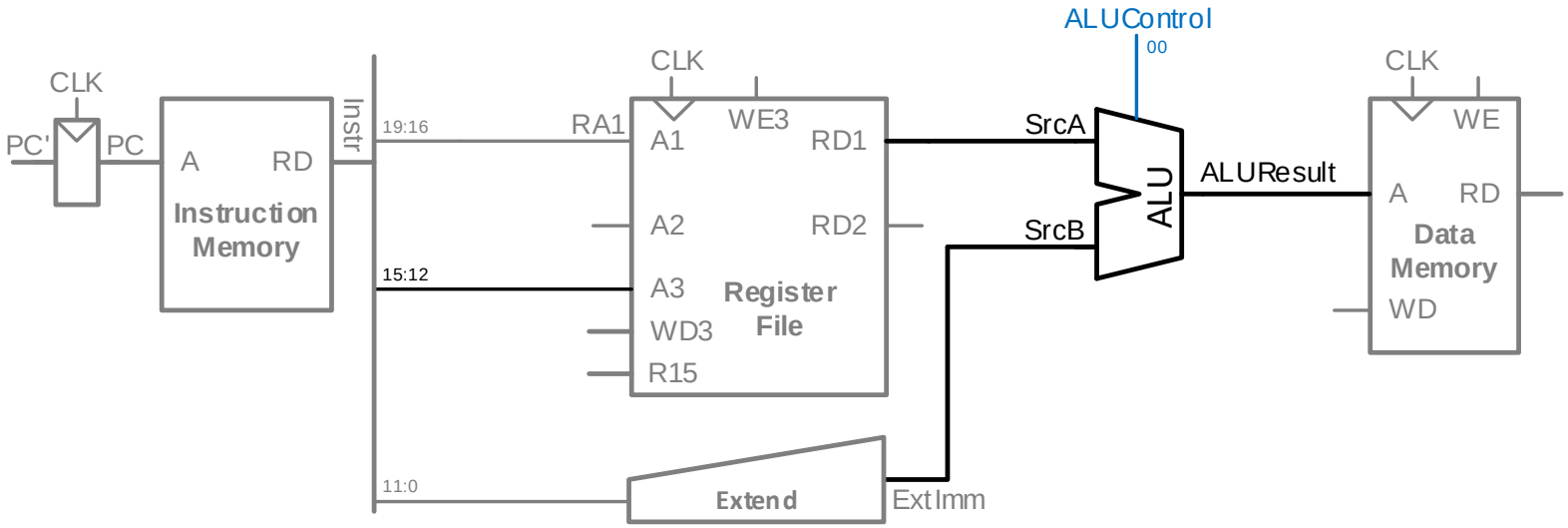
# Single-Cycle Datapath: LDR immediate

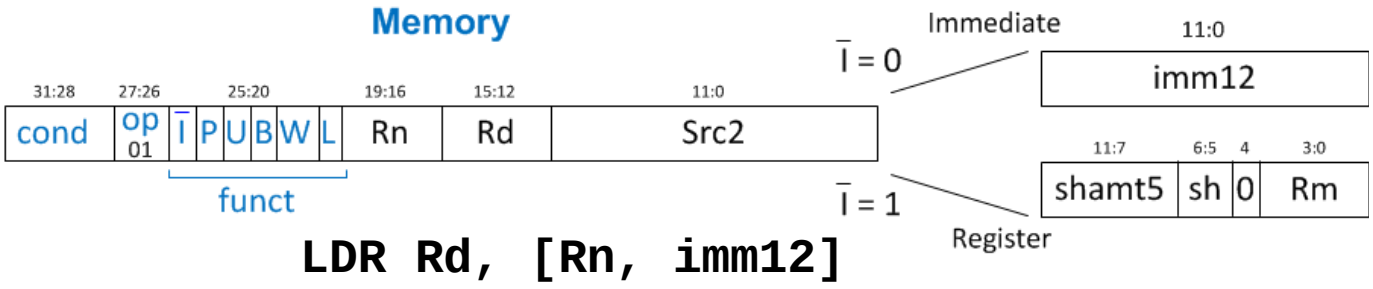
## **STEP 3:** Extend the immediate



# Single-Cycle Datapath: LDR address

## **STEP 4:** Compute the memory address

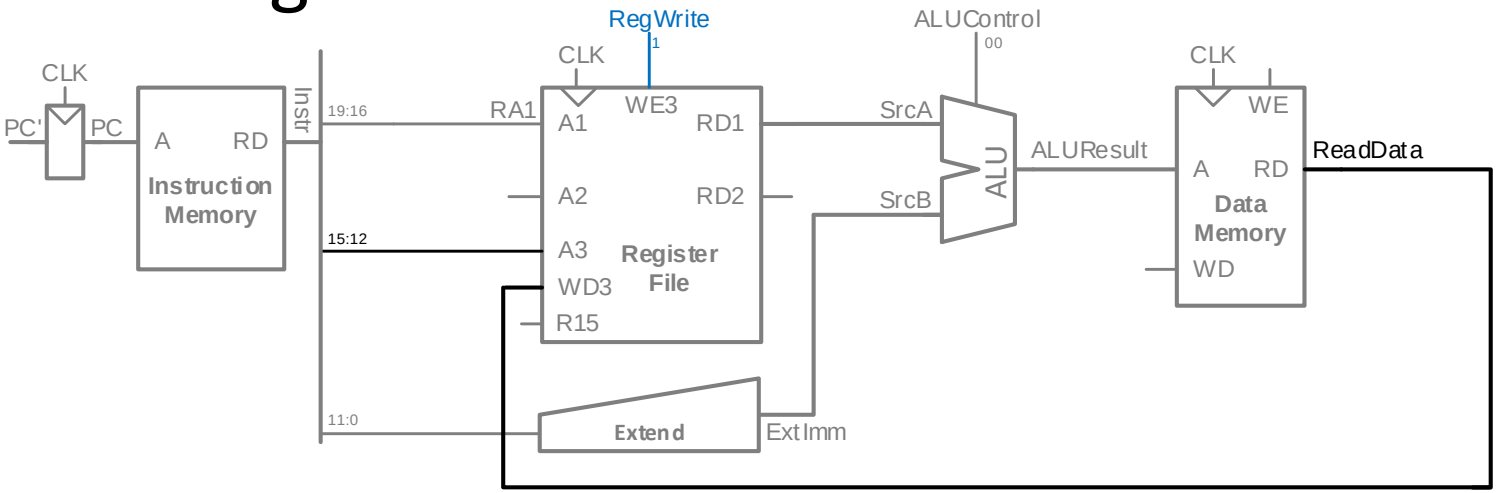


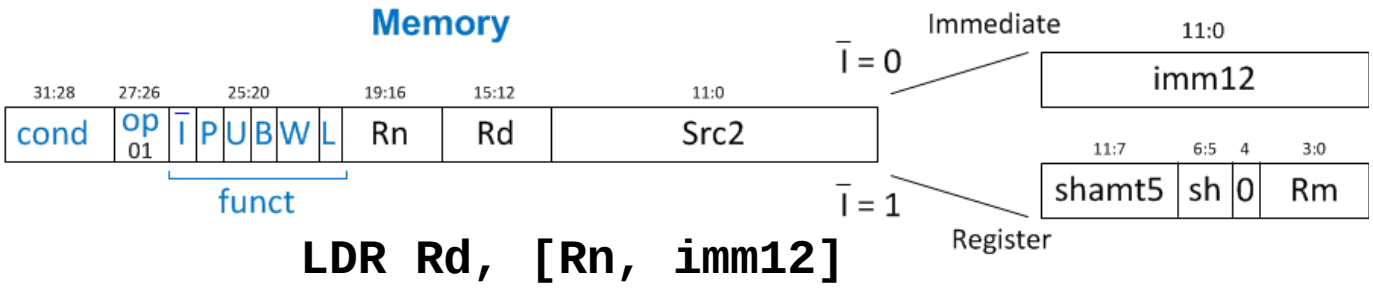




**LDR Rd, [Rn, imm12]**

# Single-Cycle Datapath: LDR memory read

**STEP 5:** Read data from memory and write it back to register file

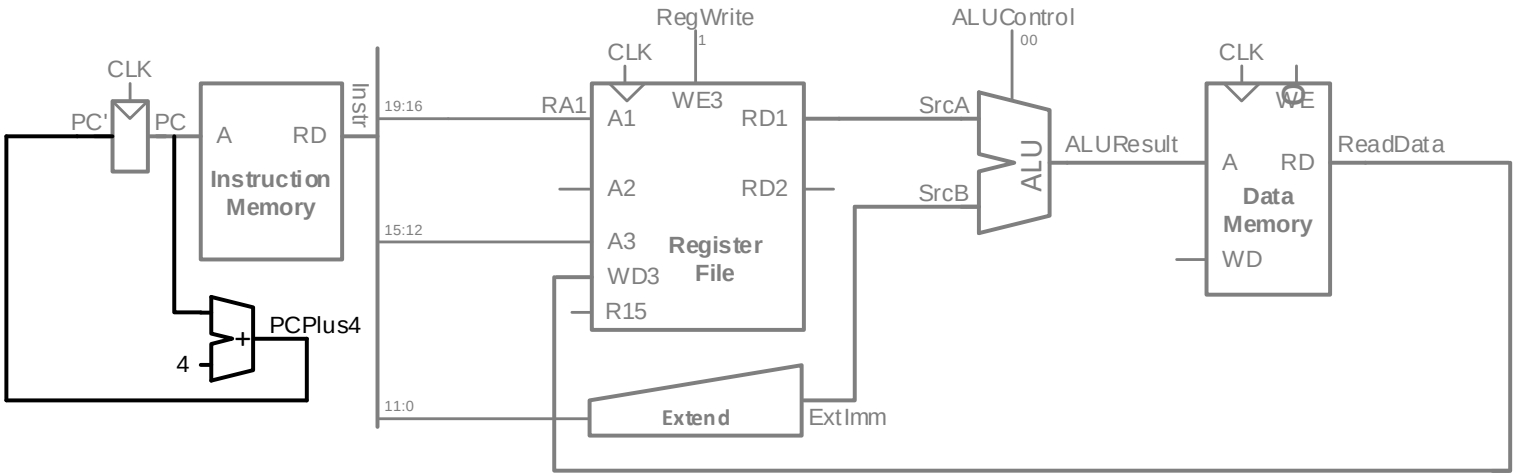



**LDR Rd, [Rn, imm12]**

# Single-Cycle Datapath: LDR PC increment

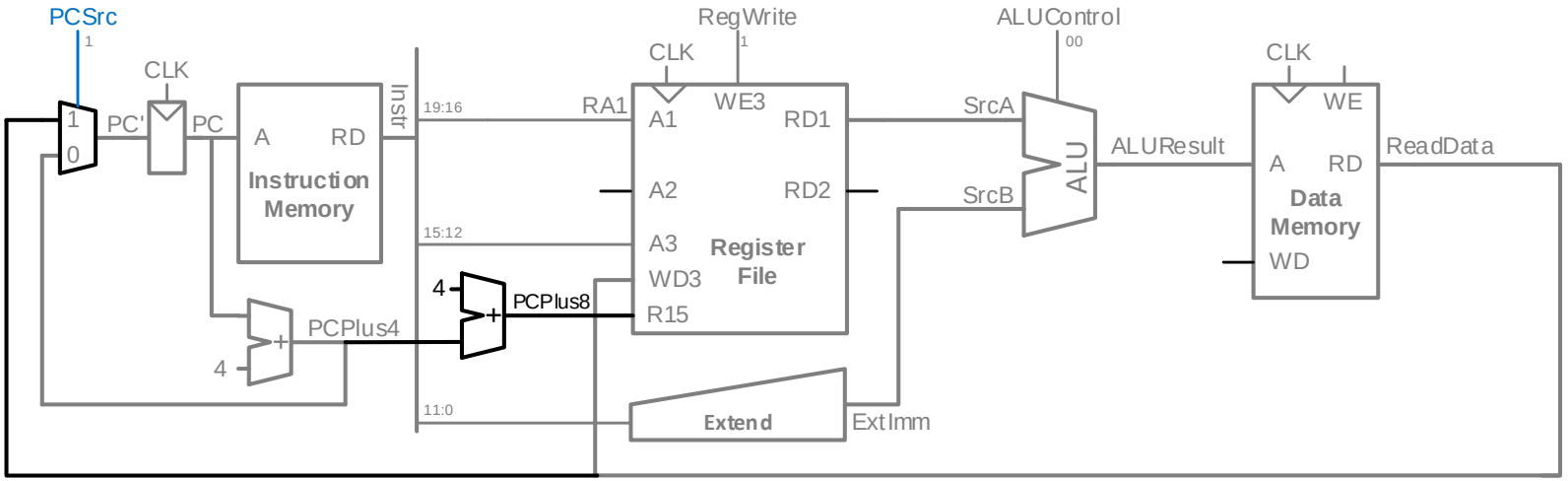
## **STEP 6:** Determine address of next instruction



# Single-Cycle Datapath: Access to PC

PC can be source/destination of instruction

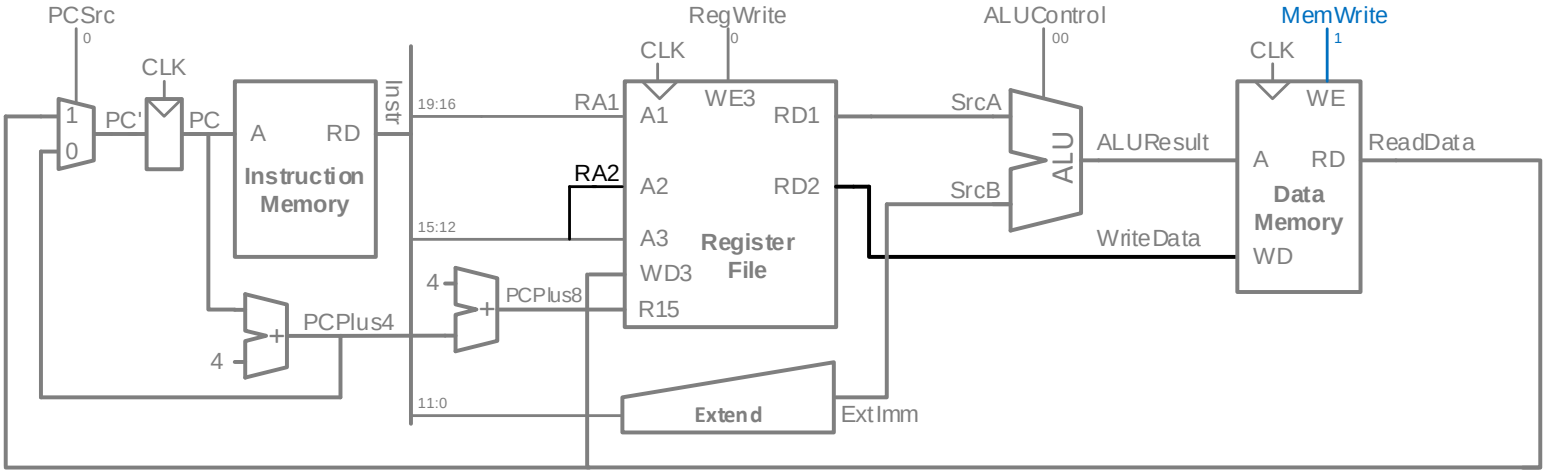
- **Source:** R15 must be available in Register File
  - **PC** is read as the current **PC plus 8**
- **Destination:** Be able to write result to PC

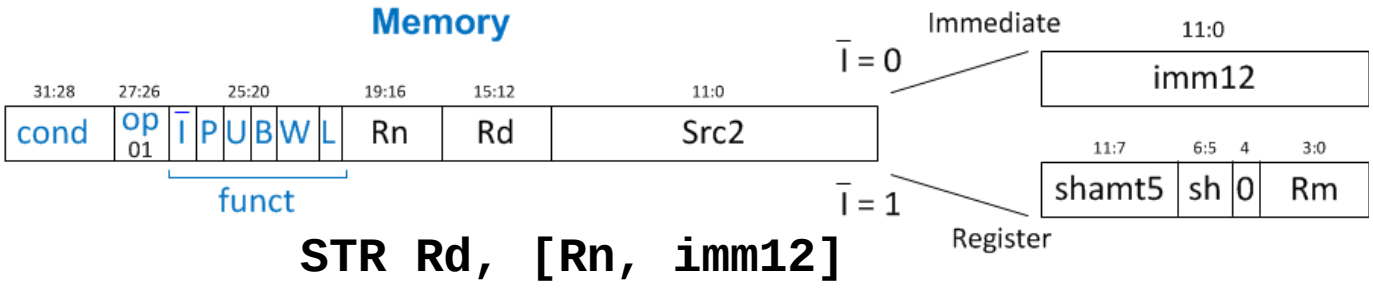


# Single-Cycle Datapath: STR

## **Expand datapath** to handle STR:

- Write data in Rd to memory



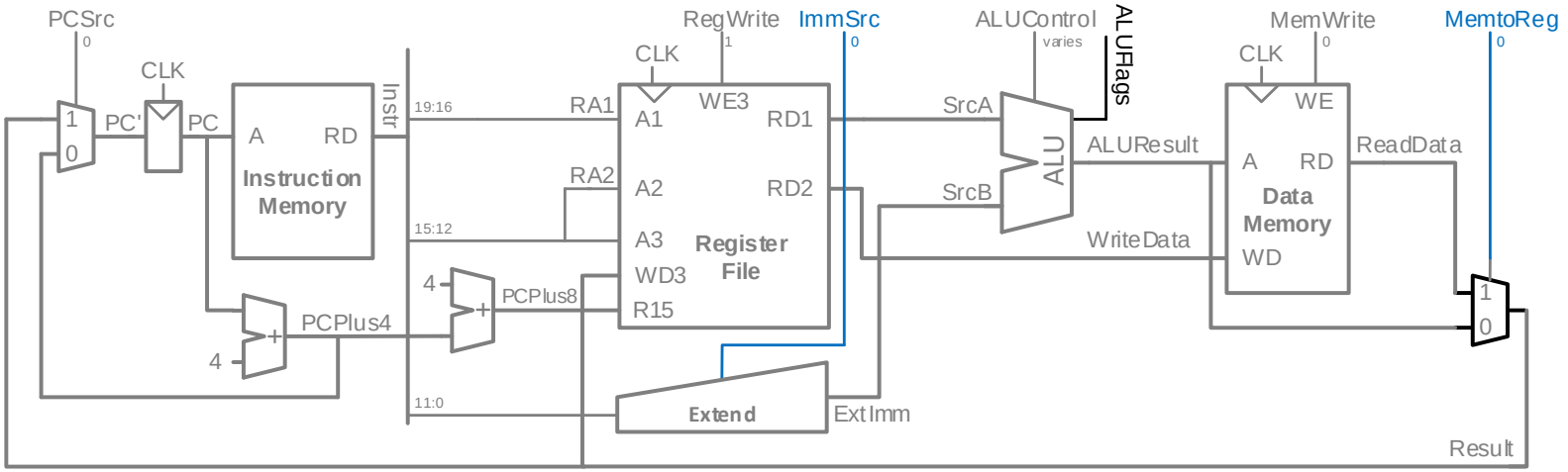




**STR Rd, [Rn, imm12]**

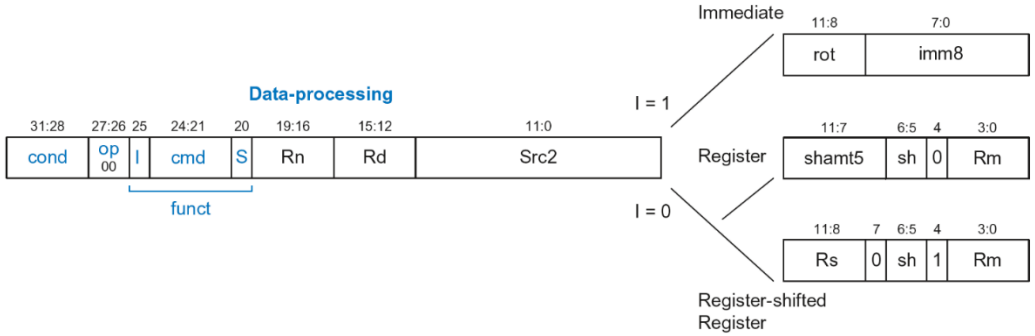
# Single-Cycle Datapath: data processing

**With immediate Src2:**

- Read from Rn and `Imm8` (*ImmSrc* chooses the zero-extended `Imm8` instead of `Imm12`)
- Write *ALUResult* to register file
- Write to Rd



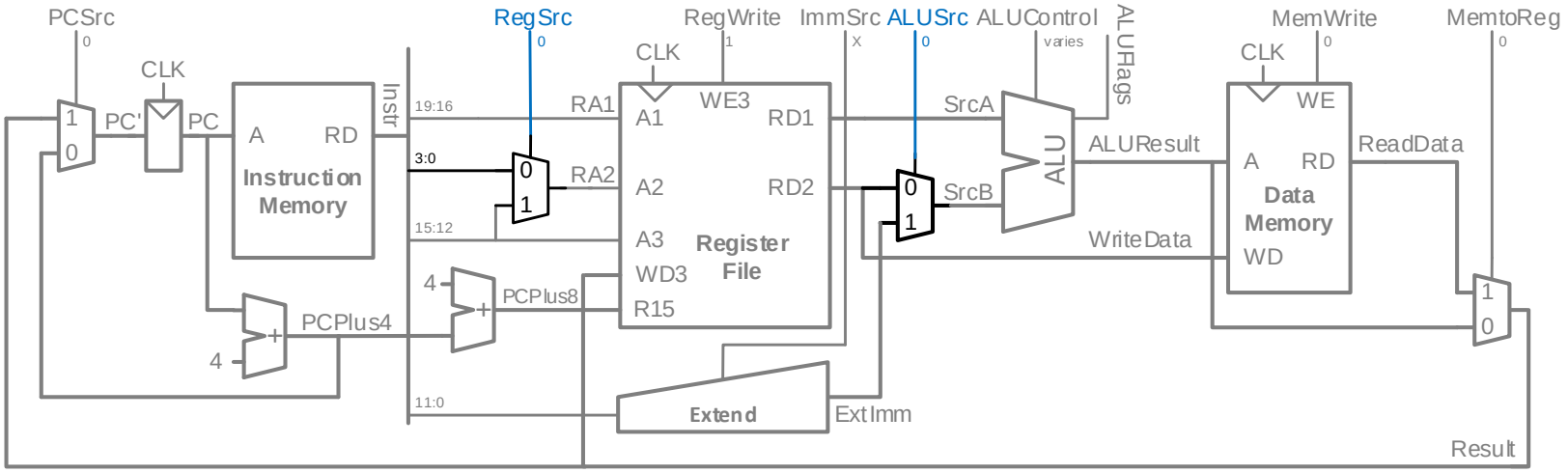

**`ADD Rd, Rn, imm8`**



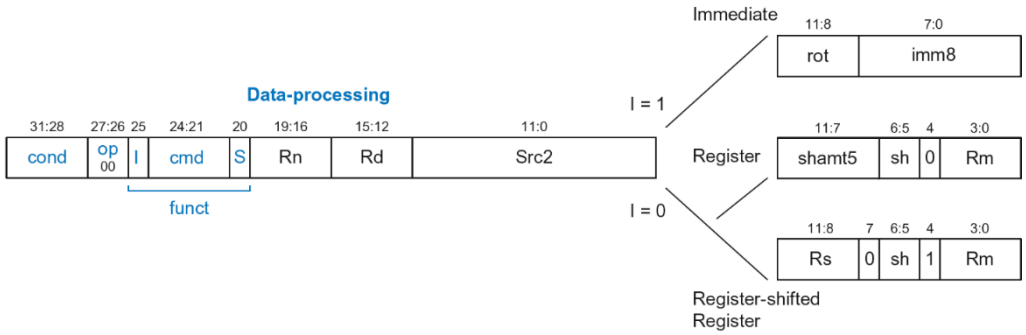
# Single-Cycle Datapath: data processing

**With register Src2:**

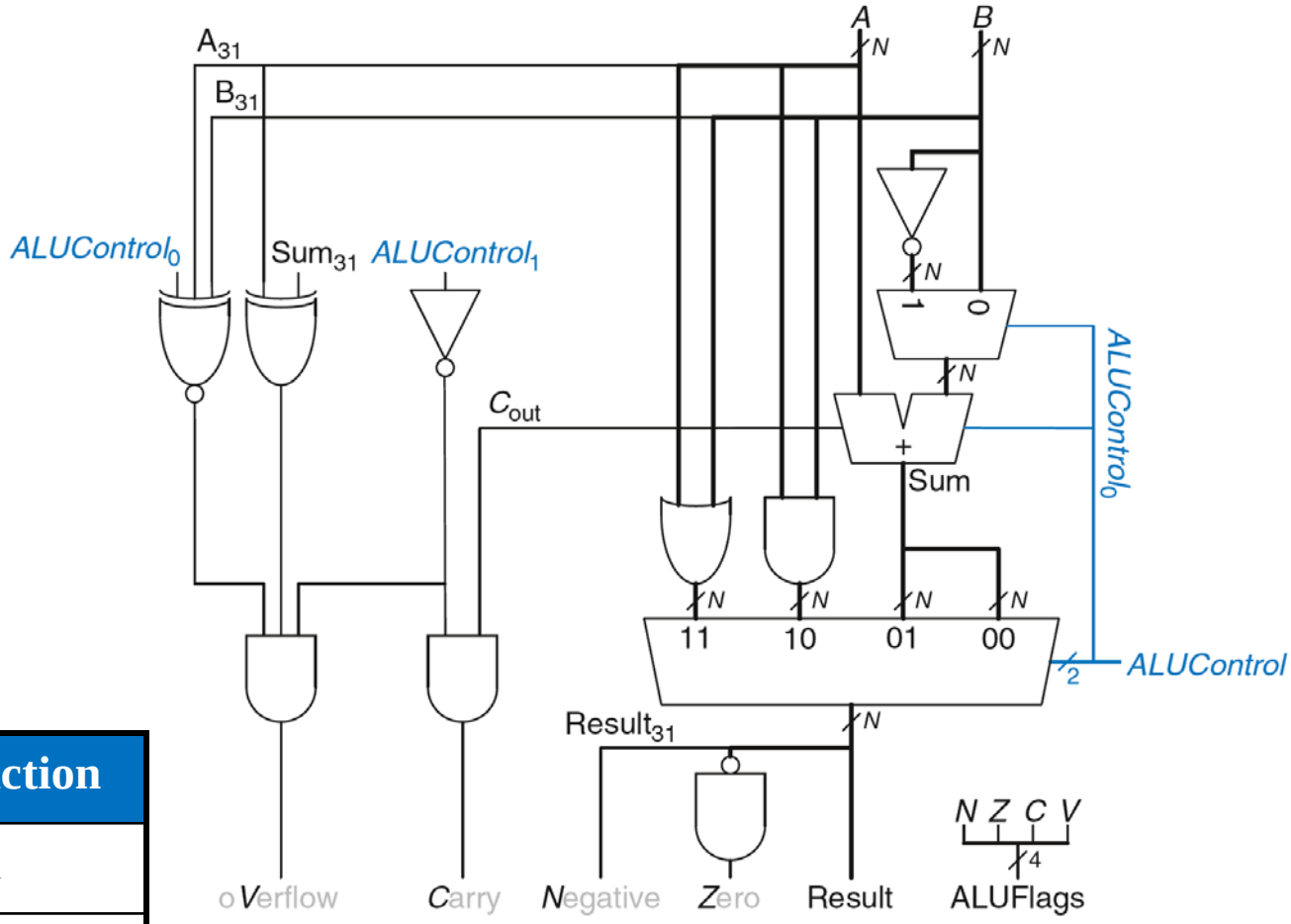
- Read from Rn and Rm (instead of `Imm8`)
- Write *ALUResult* to register file
- Write to Rd




**`ADD Rd, Rn, Rm`**



# ALU




| $ALUControl_{1:0}$ | Function |
|---|---|
| 00 | Add |
| 01 | Subtract |
| 10 | AND |
| 11 | OR |

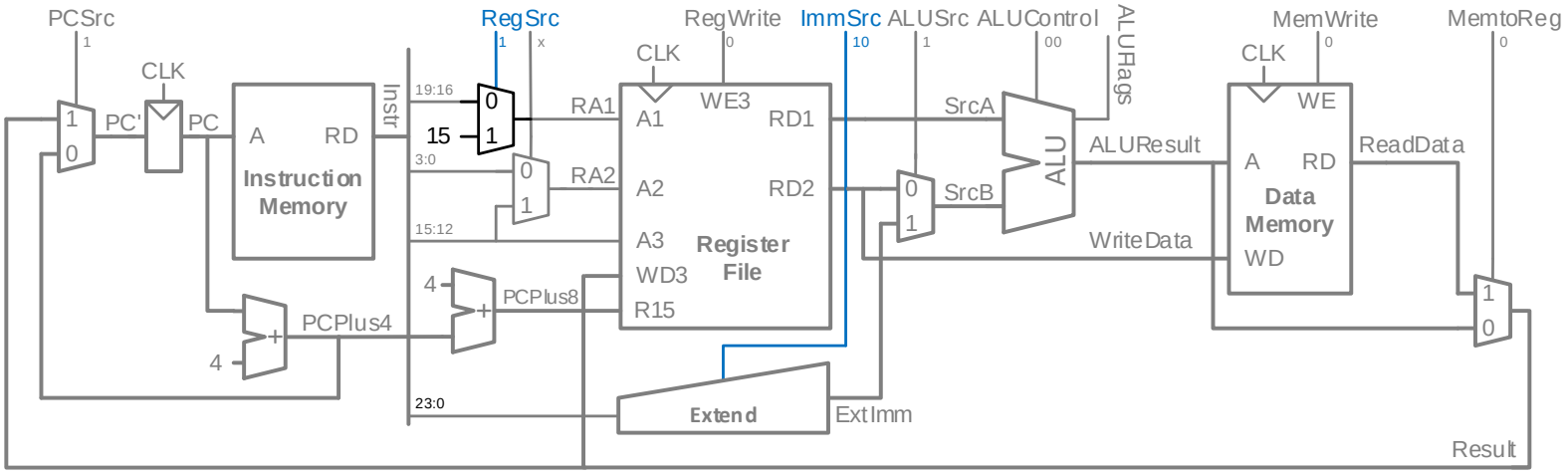
**Flags must be stored in FFs if required by by instruction (e.g., CMP, TST, SUBS, ADDS, etc)**

# Single-Cycle Datapath: B

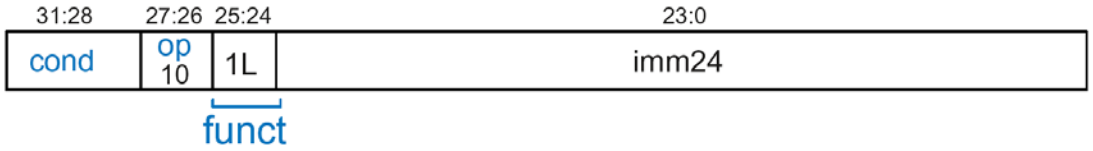
## Calculate branch target address:

BTA = (*ExtImm*) + (PC + 8)

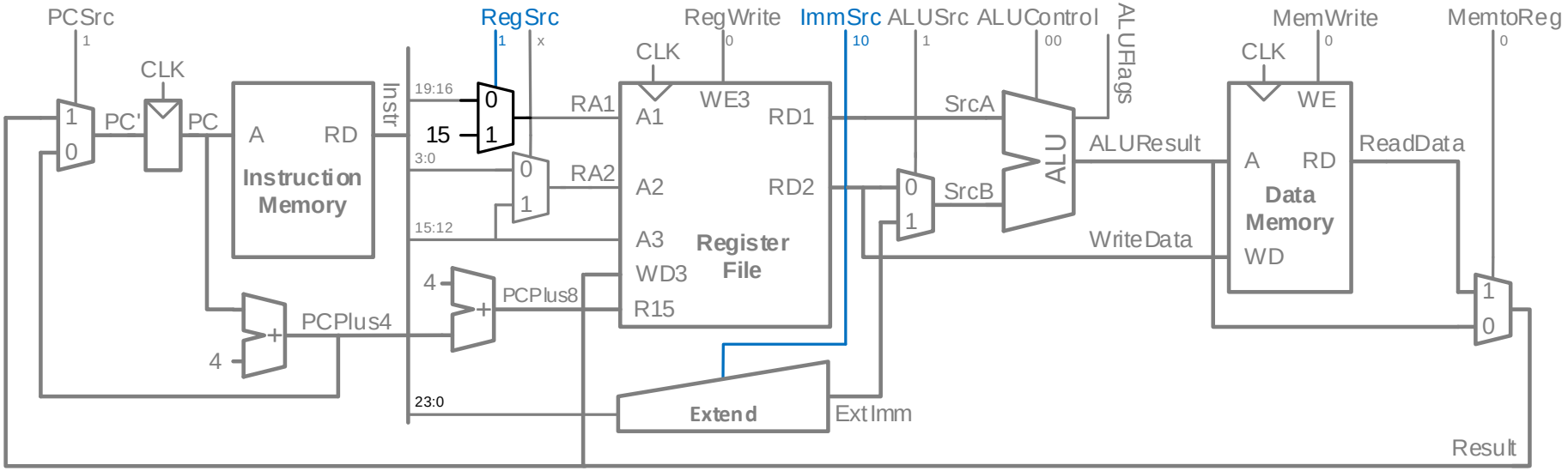
*ExtImm* = Imm24 << *2* and sign-extended




**Branch**

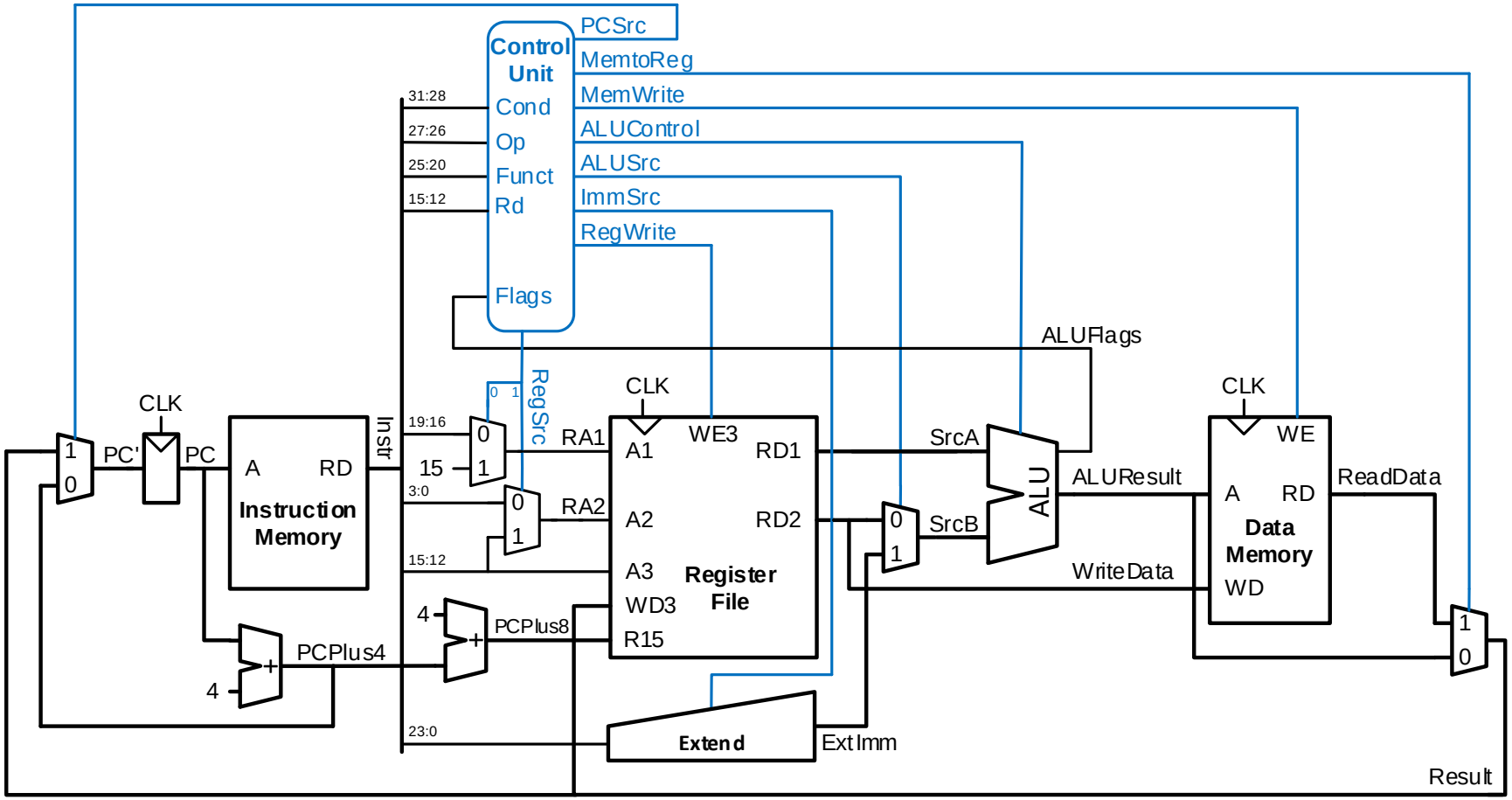


# Single-Cycle Datapath: ExtImm

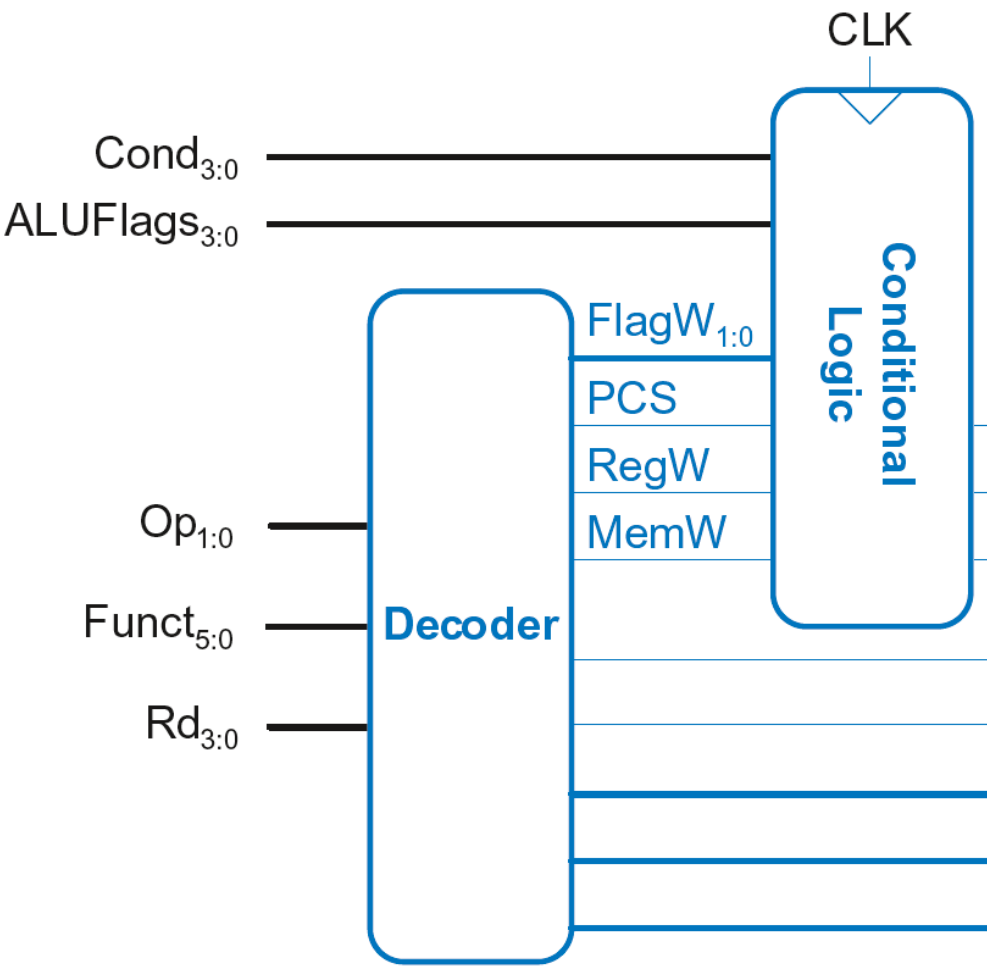



| $ImmSrc_{1:0}$ | ExtImm | Description |
|---|---|---|
| 00 | {24'b0, $Instr_{7:0}$} | Zero-extended *imm8* |
| 01 | {20'b0, $Instr_{11:0}$} | Zero-extended *imm12* |
| 10 | {6{$Instr_{23}$}, $Instr_{23:0}$} | Sign-extended *imm24* |

# Single-Cycle ARM processor



# Single-Cycle Control

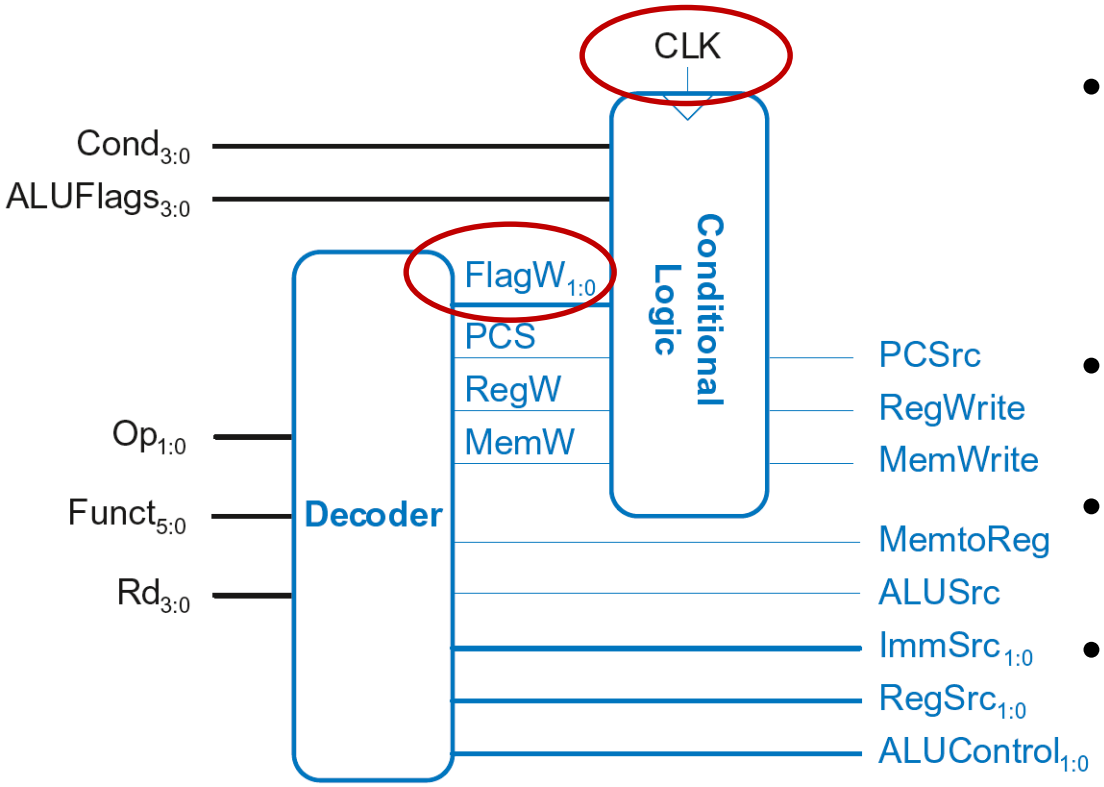



- These signals **change the state** (PC, RF, Memory)
- If instruction shouldn't execute, **forced to 0**

**Sent through Conditional Logic first, then to datapath**

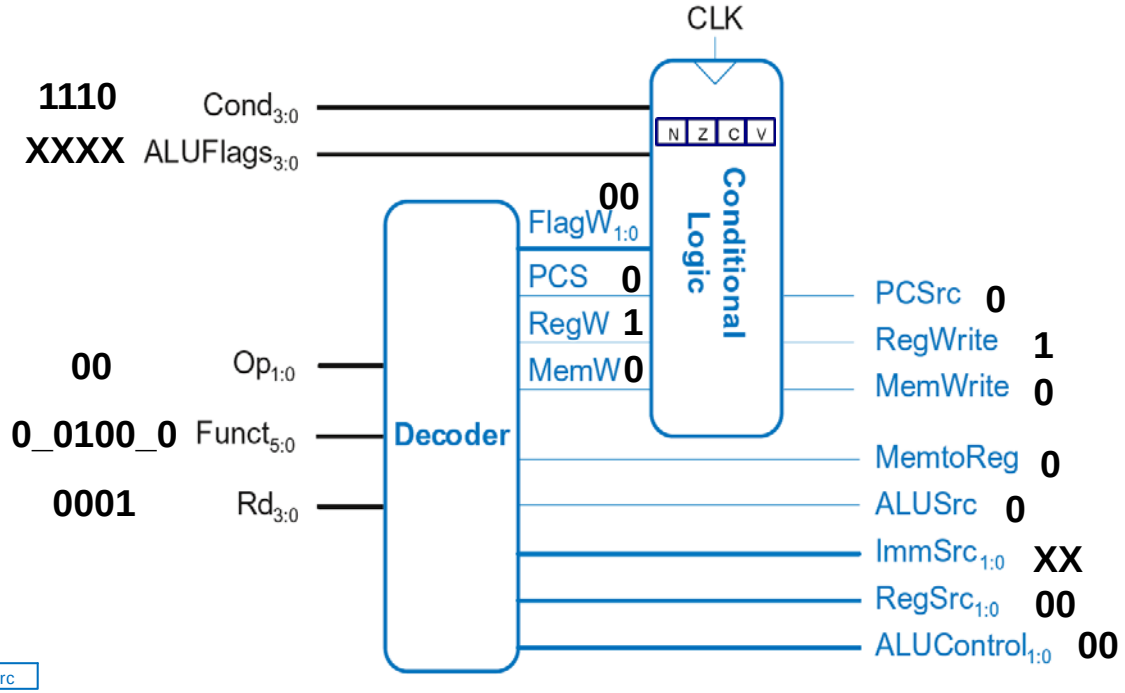
**Sent directly to datapath**

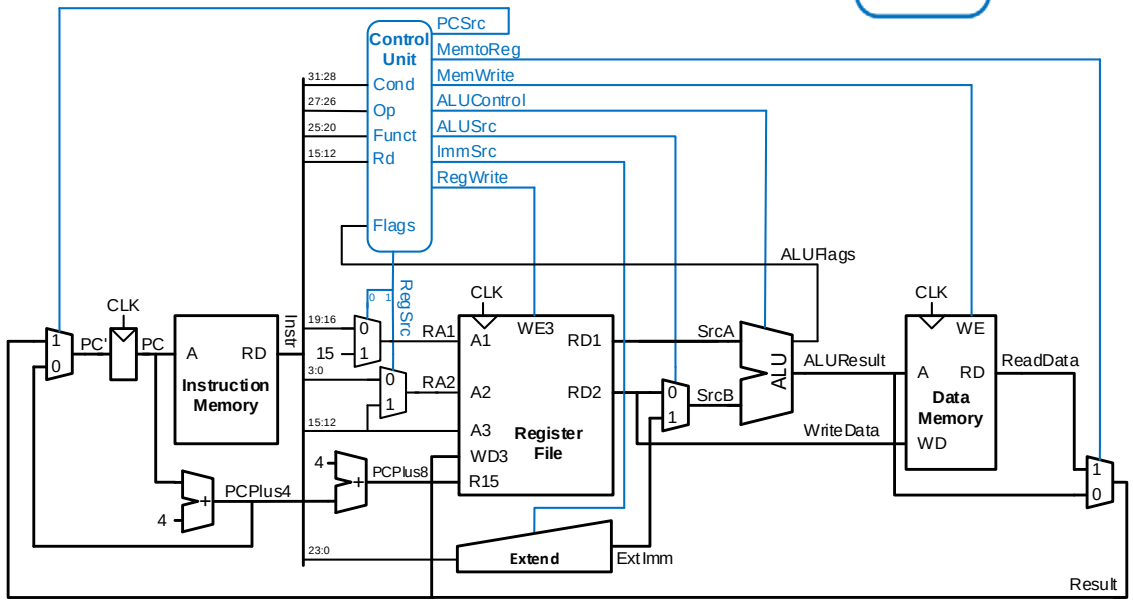
# Single-Cycle control



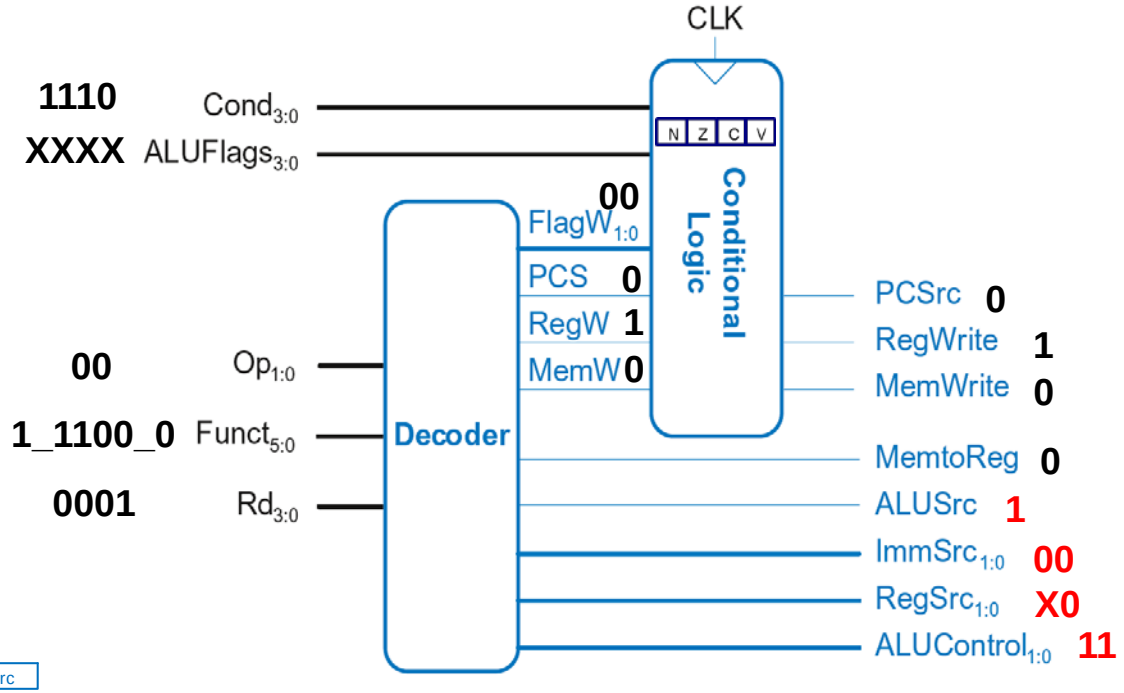

- ***FlagW*$_{1:0}$:** Flag Write signal, asserted when *ALUFlags* should be saved (i.e., on instruction with S=1)
- ADD, SUB update all flags (**NZCV**)
- AND, ORR only update **NZ** flags
- So, two bits needed:
  ***FlagW*$_1$** = 1: *NZ* saved (*ALUFlags*$_{3:2}$ saved)
  ***FlagW*$_0$** = 1: *CV* saved (*ALUFlags*$_{1:0}$ saved)

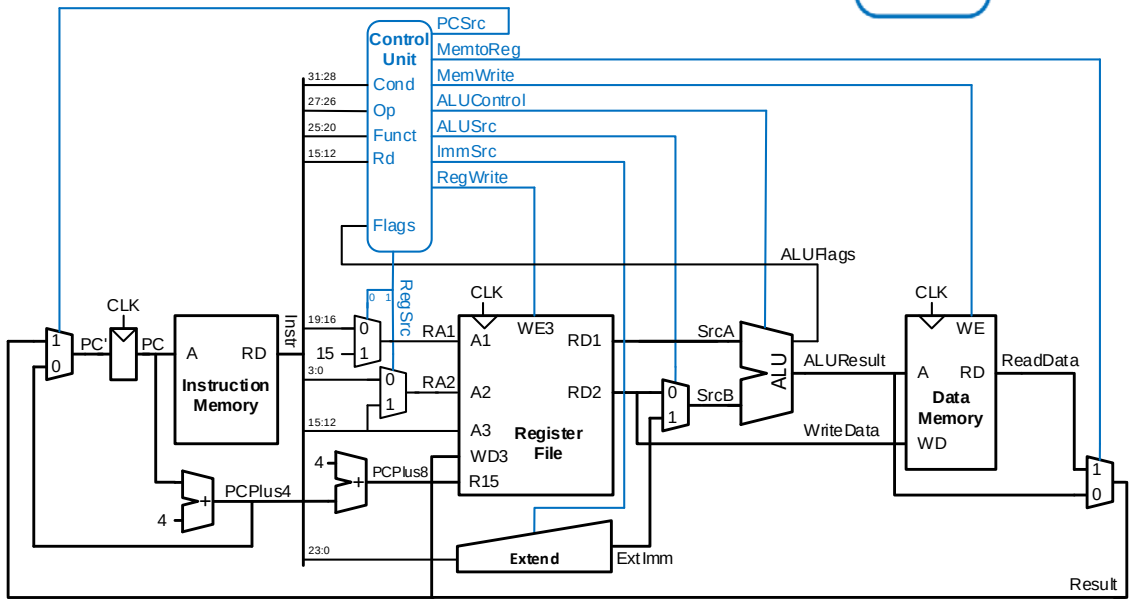
# Example of control logic: ADD R1, R2, R3



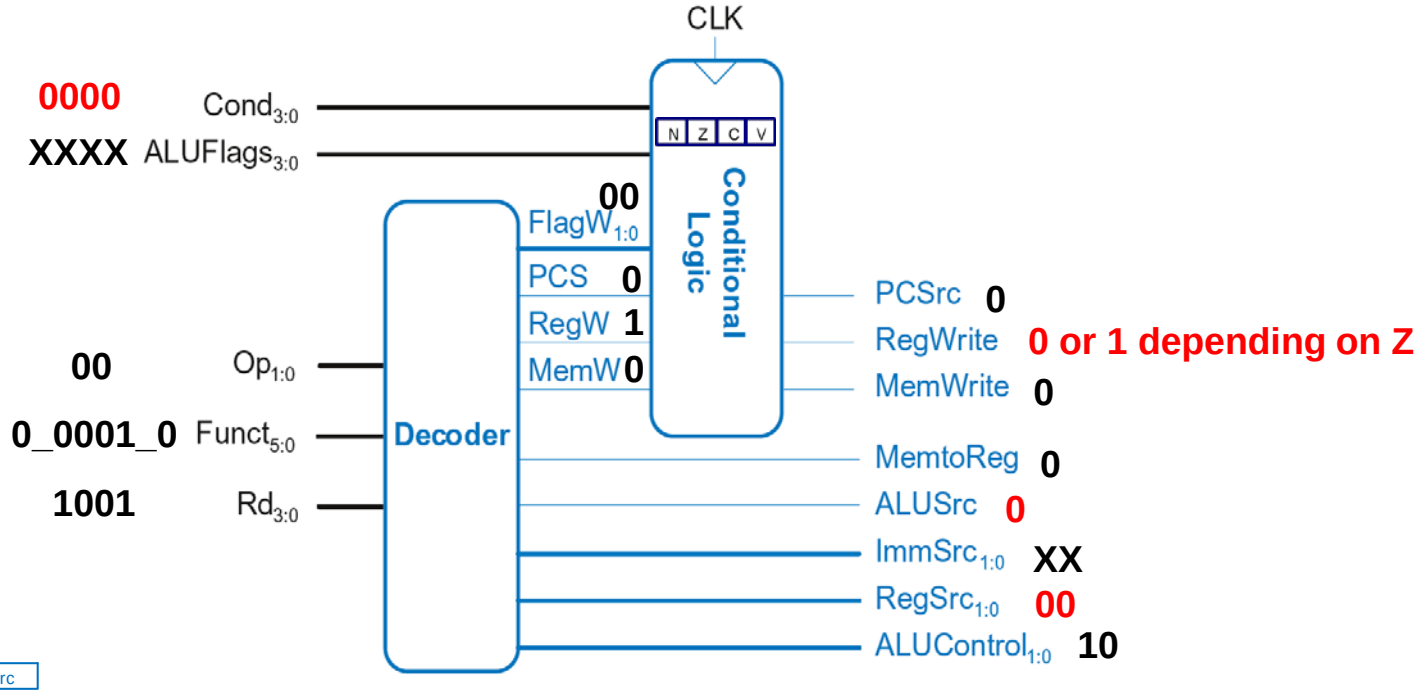



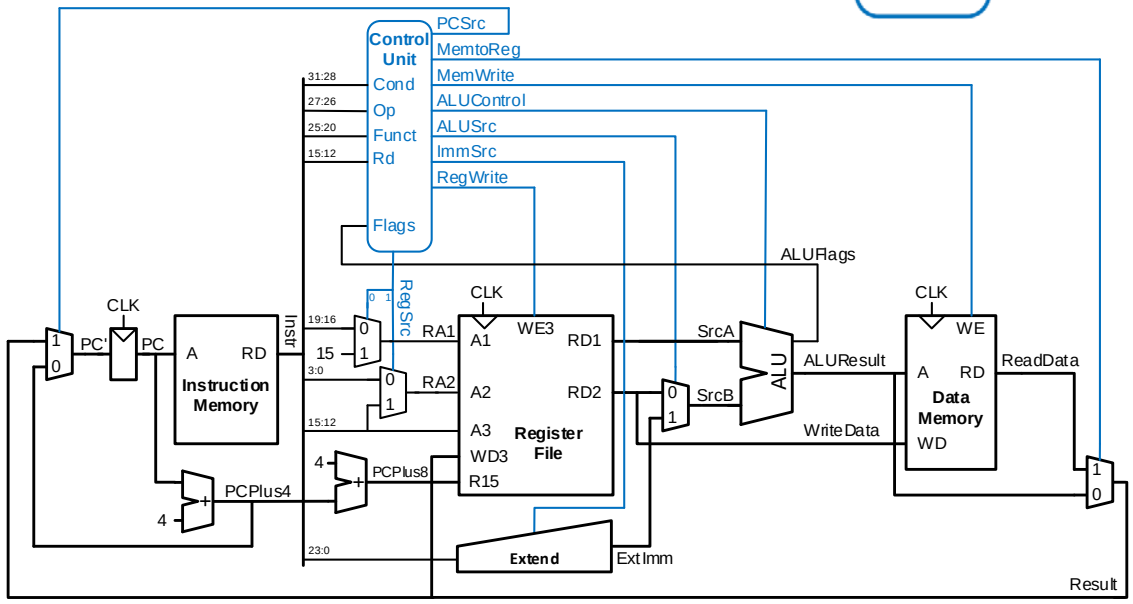
# Example of control logic: ORR R1, R2, #F0



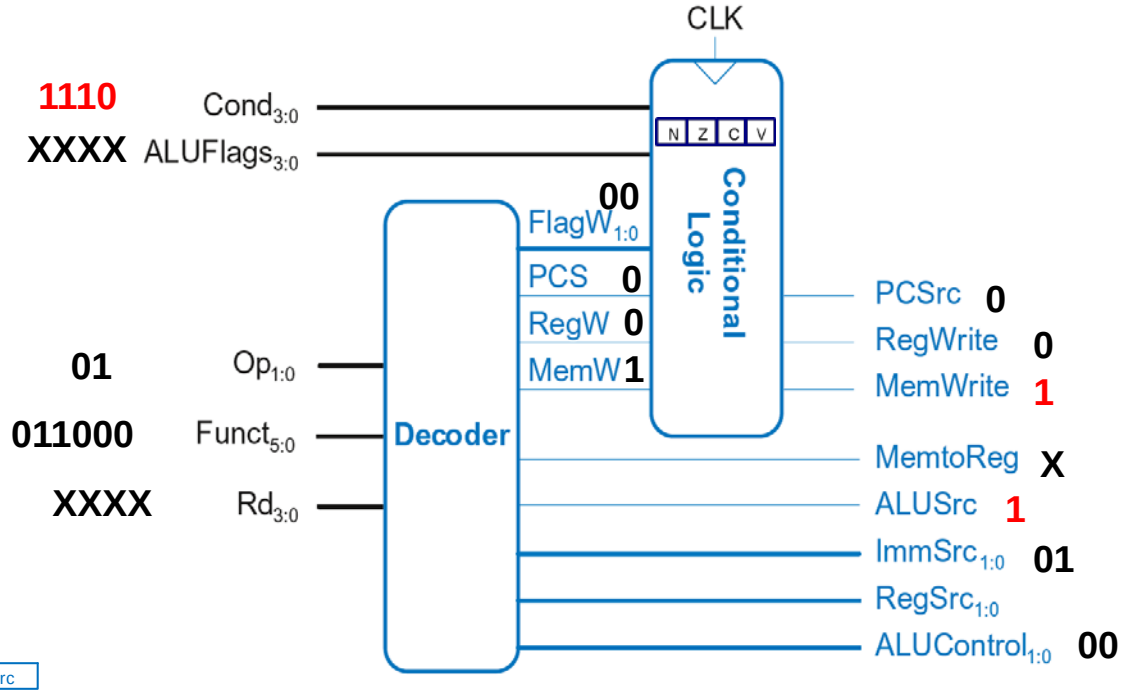



# Example of control logic: ANDEQ R5, R6, R7

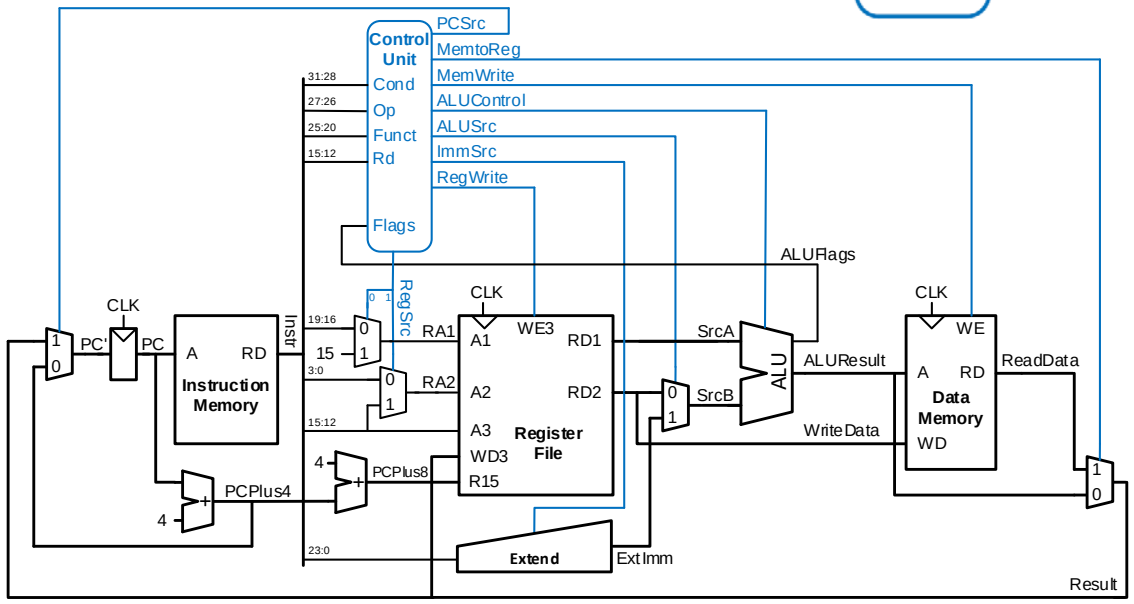


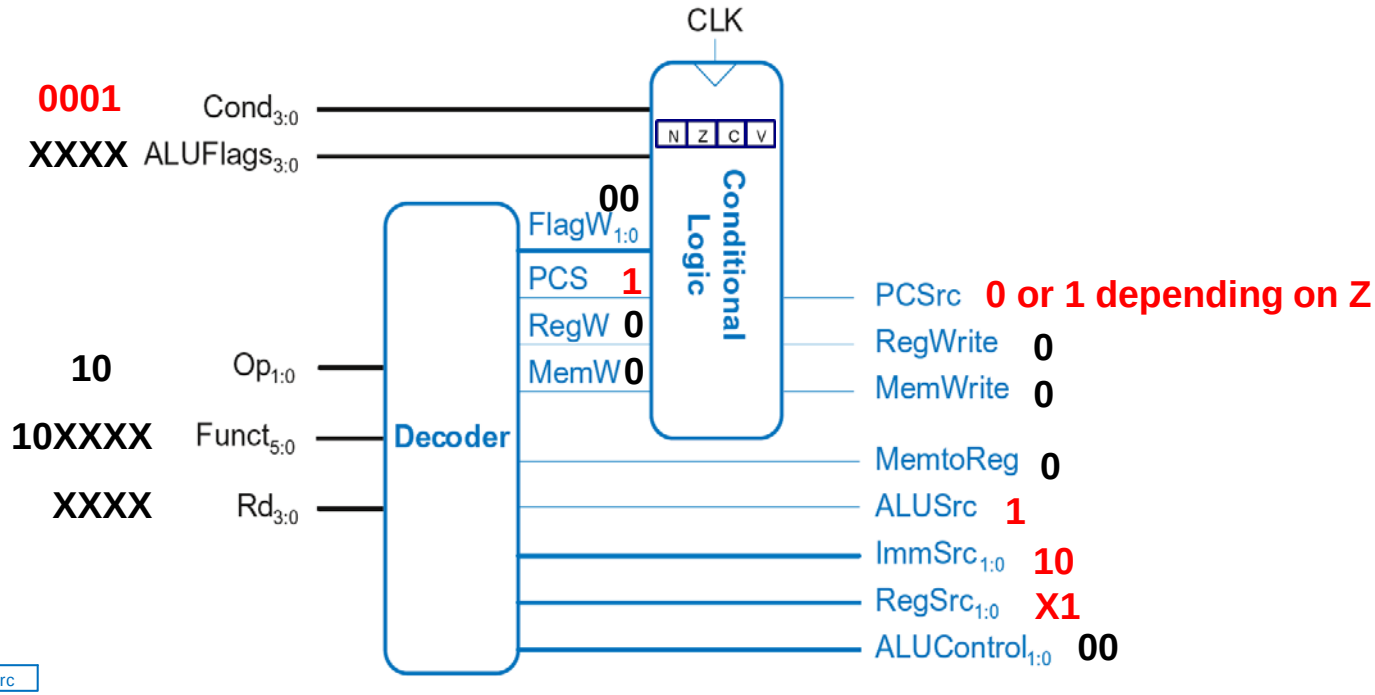
# Example of control logic: STR R11, [R5, #8]

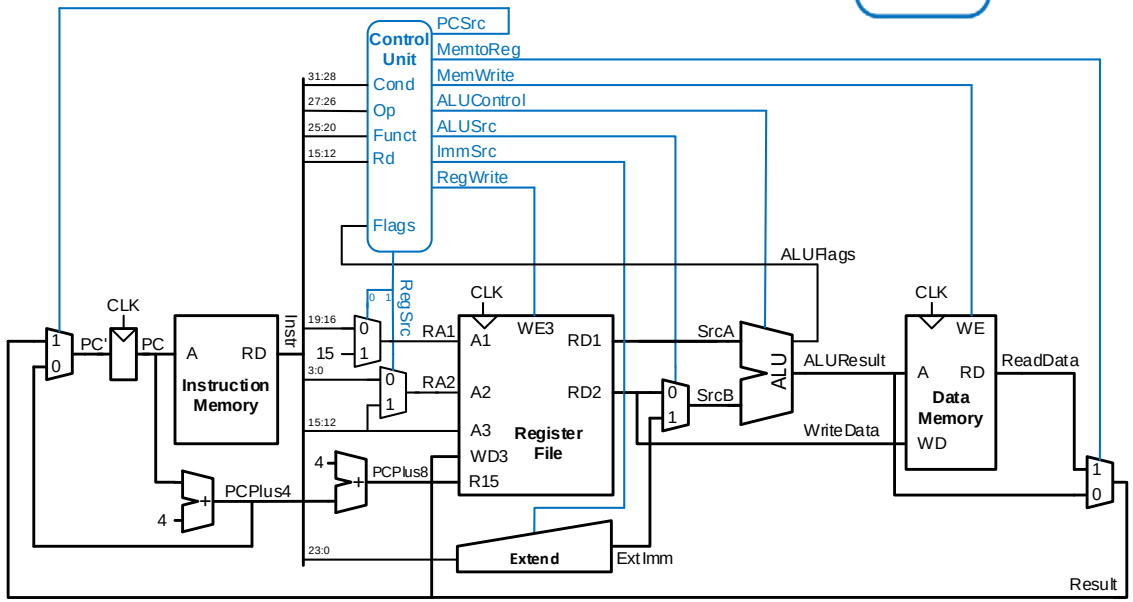



10



# Example of control logic: BNE THERE

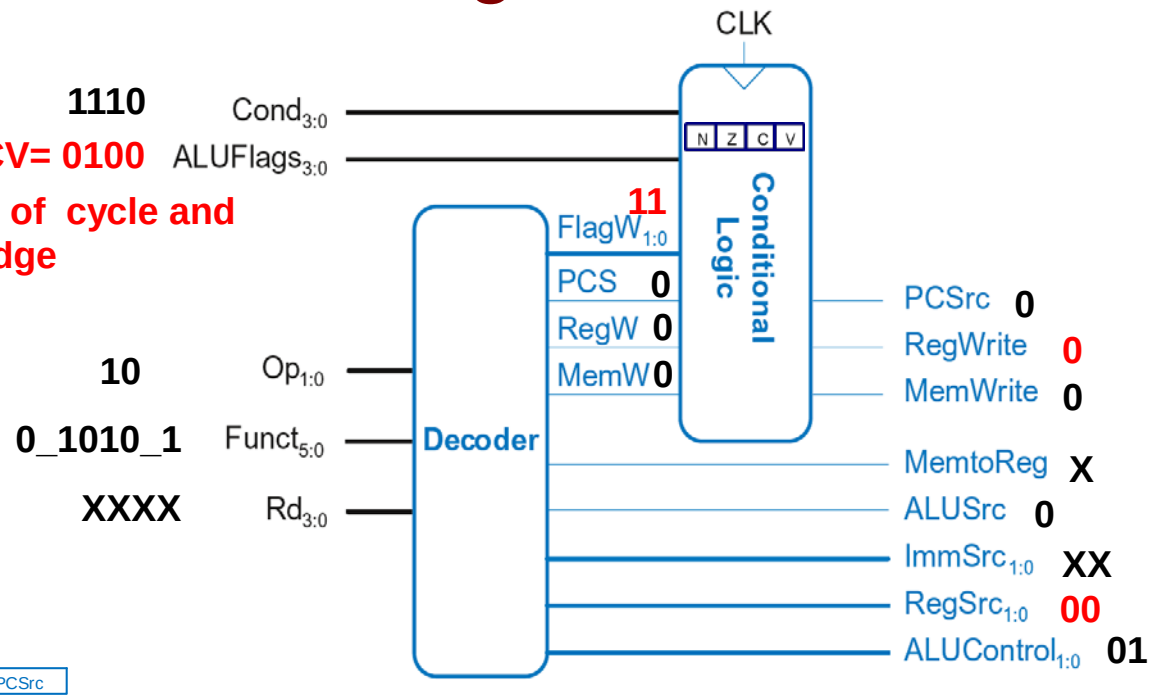


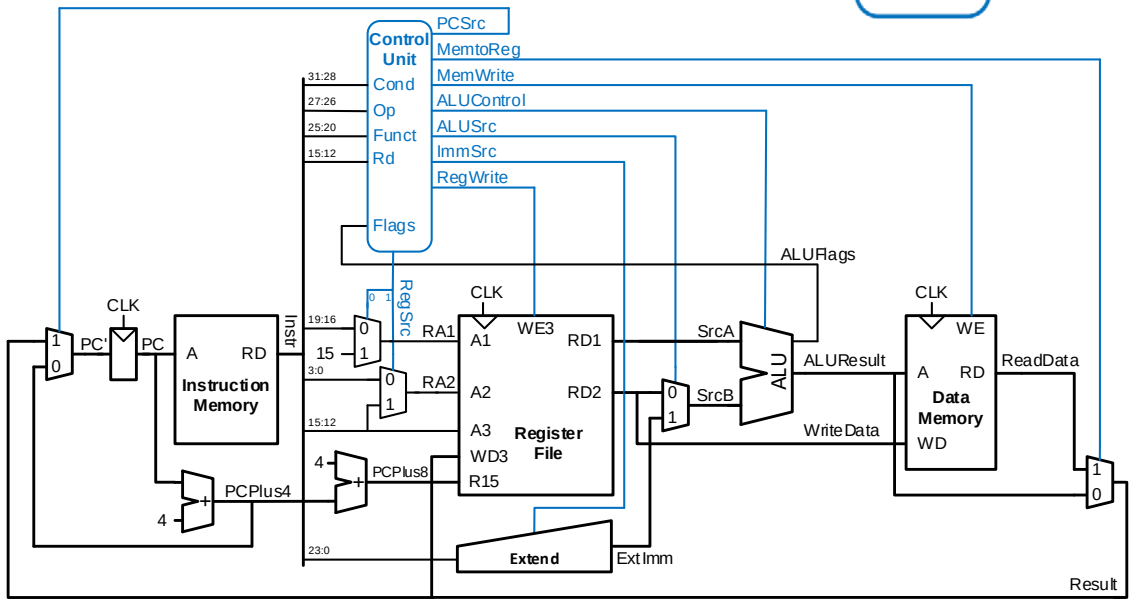



# Example of control logic: CMP R1, R2

**R1=10, R2 = 10 → NZCV= 0100**

**available by end of cycle and latched at +ve edge**



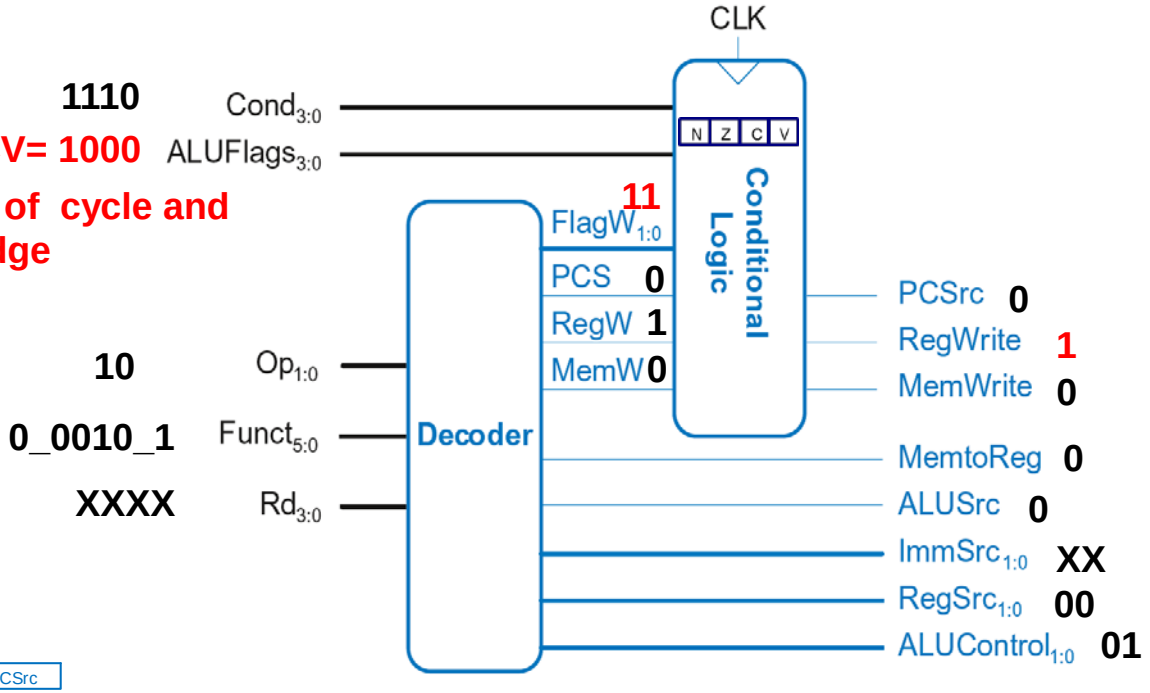



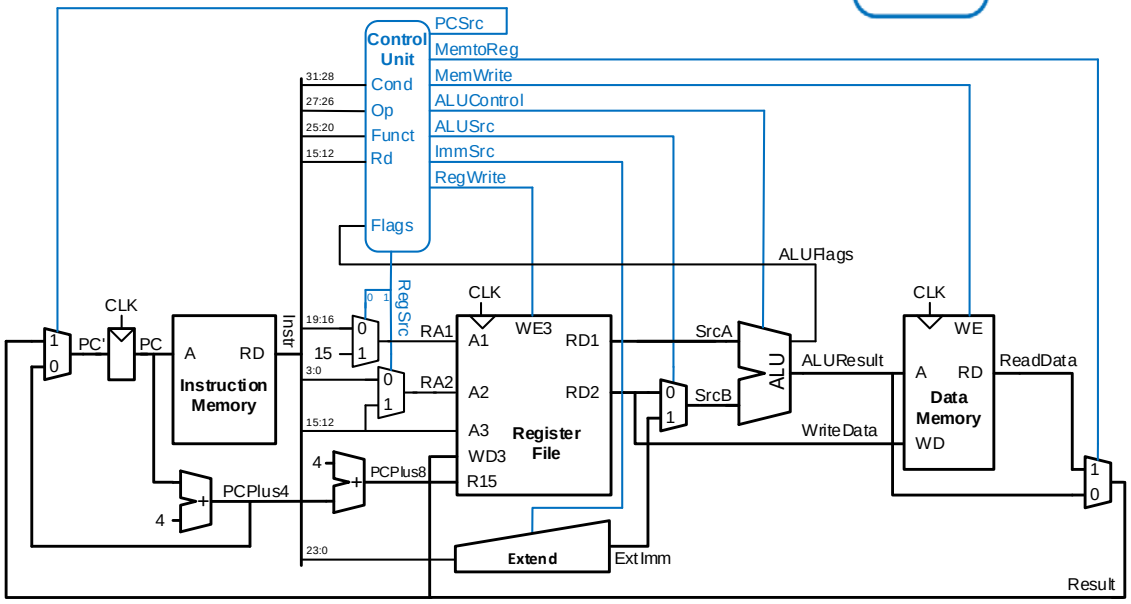
# Example of control logic: SUBS R3, R1, R2

**1110**

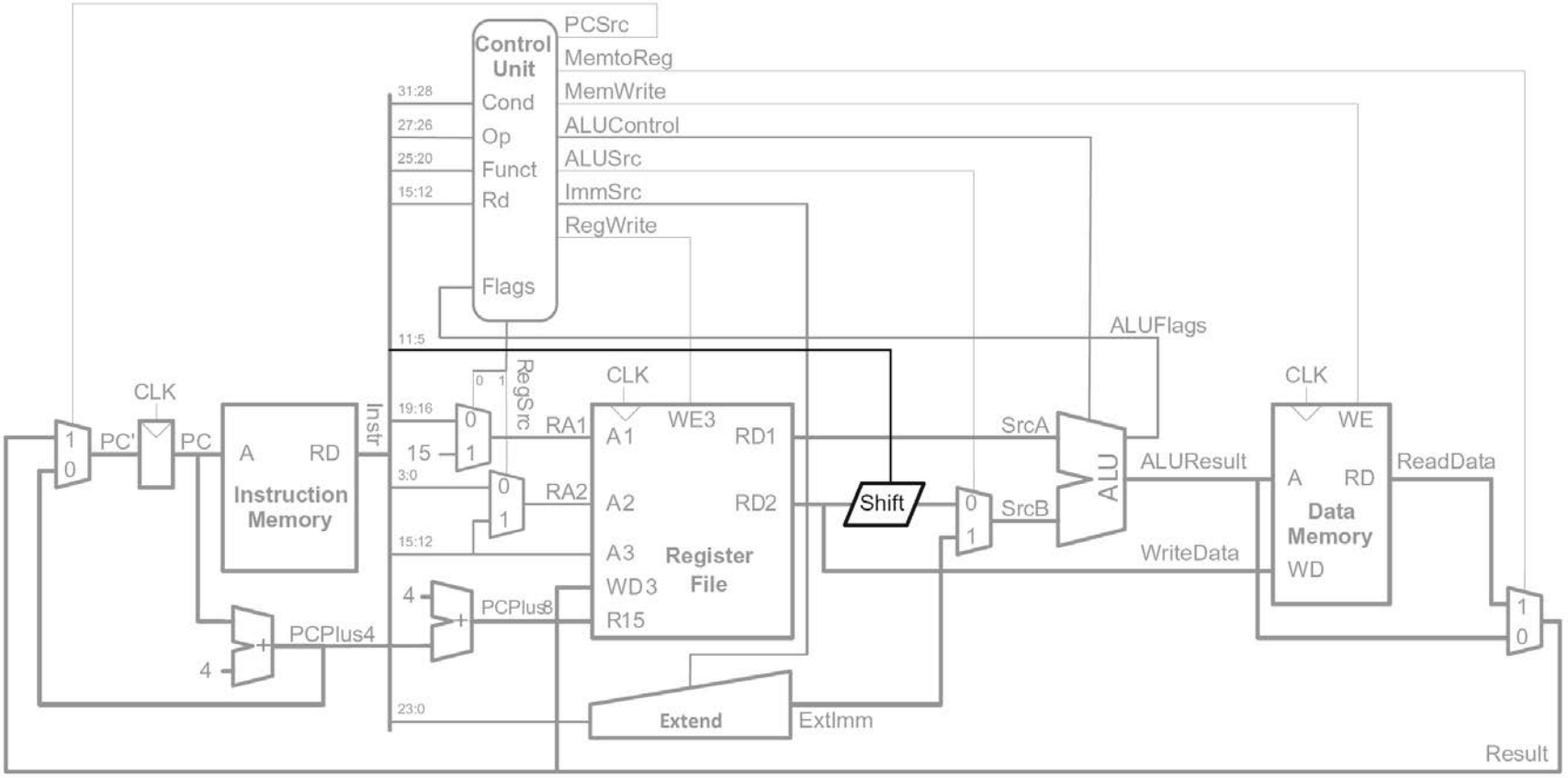
**R1=4, R2=5 → NZCV= 1000**

**available by end of cycle and latched at +ve edge**



# Extended functionality: shifted register (by constant)




**Assembly Code**

**Field Values**

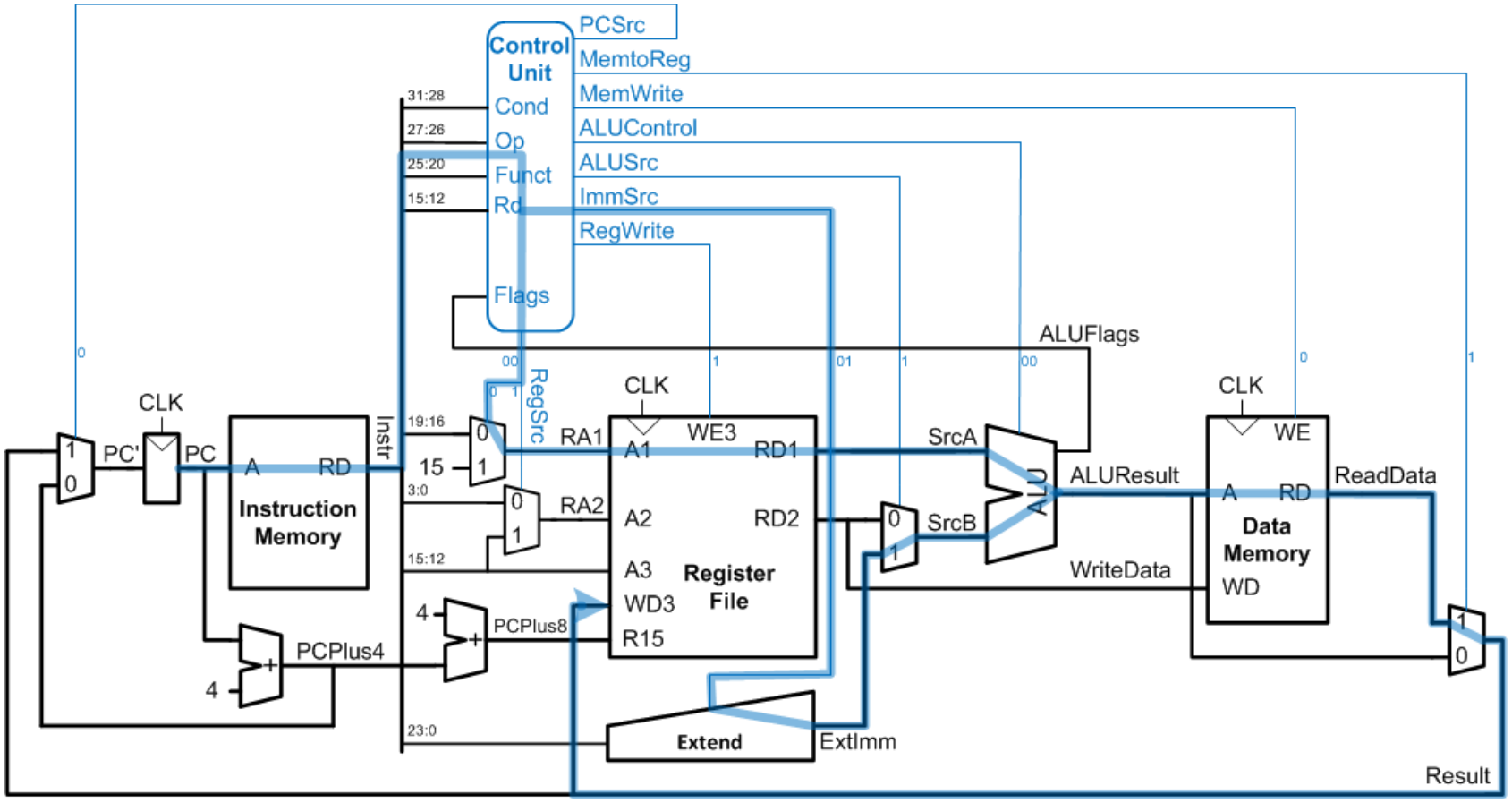
ADD R7, R2, R12, LSR #5

| 31:28 | 27:26 | 25 | 24:21 | 20 | 19:16 | 15:12 | 11:7 | 6:5 | 4 | 3:0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 0 | 0 | 4 | 0 | 2 | 7 | 5 | $01_2$ | 0 | 12 |
| cond | op | I | cmd | S | rn | rd | shamt5 | sh | | rm |

**No change to controller**

# Critical path

- $T_C$ is limited by the critical path (LDR)




- Single-cycle critical path:

$$T_{c1} = t_{pcq\_PC} + t_{\text{mem}} + t_{\text{dec}} + \max[t_{mux} + t_{RF\text{read}}, t_{sext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

# Critical path delay

- Single-cycle critical path:

  $T_{c1} = t_{pcq\_PC} + t_{\text{mem}} + t_{\text{dec}} + \max[t_{mux} + t_{RF\text{read}}, t_{sext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$

- In most cases:
  - $T_{c1} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{dec} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$

| Element | Parameter | Delay (ps) |
|---|---|---|
| Register clock-to-Q | $t_{pcq\_\text{PC}}$ | 40 |
| Register setup | $t_{\text{setup}}$ | 50 |
| Multiplexer | $t_{\text{mux}}$ | 25 |
| ALU | $t_{\text{ALU}}$ | 120 |
| Decoder | $t_{\text{dec}}$ | 70 |
| Memory read | $t_{\text{mem}}$ | 200 |
| Register file read | $t_{RF\text{read}}$ | 100 |
| Register file setup | $t_{RF\text{setup}}$ | 60 |

$T_{c1} = t_{pcq\_PC} + 2t_{\text{mem}} + t_{dec} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$
$= [50 + 2(200) + 70 + 100 + 120 + 2(25) + 60]$ ps
$= 840$ ps

# Summary

- Single-cycle processor design is simple
- Plenty of room for improvement:
  - Multi-cycle
  - Pipelining
  - Superscalar
- In Lab05 you will design, implement and boot your first processor!