ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

Australian National University

Plan: Week 6

Last week: Instruction Set Architecture (specification)

This Week: Microarchitecture (implementation)

Machine Language

- Instructions are stored in memory as fixed-size words
  - For ARMv4, each instruction is 32 bits wide
- ARM defines three instruction formats
  - Data processing (DP)
  - Memory
  - Branch
- Instruction *fields* encode the instruction operation and its operands
- Understanding binary encoding of instructions is necessary for building the CPU

Instruction Format – 1: DP

31:28	27:26	25:20	19:16	15:12	11:0
cond	op	funct	Rn	Rd	Src2

- Operands
  - Rn: first source operand register (0000, 0001, ..., 1111)
  - Src2: second source register or immediate
  - Rd: destination register
- Control fields
  - cond: specifies conditional execution (1110 for unconditional)
  - op: the operation code or opcode (00)
  - funct: the function/operation to perform

Instruction Format – 1: DP

31:28	27:26	25:20	19:16	15:12	11:0
cond	op	funct	Rn	Rd	Src2

31:28	27:26	25	24:21	20	19:16	15:12	11:0
cond	00	I	cmd	S	Rn	Rd	Src2

- op = 00 for DP instructions
- cmd specifies the specific DP instruction (0100 for ADD and 0010 for SUB)
- I-bit
  - I = 0: Src2 is a register
  - I = 1: Src2 is an immediate
- S-bit: 1 if the instruction sets the condition flags

DP with Src2 as Immediate

- Bit 25 (I) informs the CPU how to interpret Src2
  - I = 1, CPU interprets Src2[7:0] as an unsigned 8-bit constant

- Format (Src2 = immediate)

ADD R0, R1, #16

↓ ↓ ↓

ADD Rd, Rn, #imm8

31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

DP with Src2 as Register

- Bit 25 informs the CPU how to interpret Src2
  - I = 0, CPU interprets Src2[3:0] as a register

- Format (Src2 = Register)

ADD R0, R1, R3

↓ ↓ ↓

ADD Rd, Rn, Rm

31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

Instruction Format – 2: Memory

31:28	27:26	25:20						19:16	15:12	11:0
cond	op	I	P	U	B	W	L	Rn	Rd	Src2

- op = 01
- Rn = base register (base address)
- Rd = destination (load), source (store)
- Src2 = offset (register, shifted register, <u>immediate</u>)
- funct = 6 control bits
  - I (Bit 25): Encoding of Src2
  - L (Bit 20): Load or Store
  - Remaining bits (ignore)

LDR with Src2 as Immediate

- I (Bit 25) = 1: Src2 = imm12 where imm2 is a 12-bit unsigned offset added to the value in the base register (Rn)

- Format of LoaD Register instruction

  LDR R0, [R1, #12]

  ↓ ↓ ↓

  LDR Rd, [Rn, #imm12]

- L (Bit 20) = 1: CPU performs an LDR

31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	1	Rn	Rd	imm12

STR with Src2 as Immediate

- I (Bit 25) = 1: Src2 = imm12 where imm2 is a 12-bit unsigned offset added to the value in the base register (Rn)

- Format of STore Register instruction

  STR R0, [R1, #12]

  ↓ ↓ ↓

  STR Rd, [Rn, #imm12]

- L (Bit 20) = 0: CPU performs an STR

31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	1	Rn	Rd	imm12

Instruction Format – 3: Branch

31:28	27:26	25:24	23:0
cond	op	1L	imm24

- op = 10
- imm24 = 24-bit **signed** immediate
- The two bits 25:24 form the funct field
  - Bit 25 is always 1
  - L bit: L = 0 for B (Branch)
  - L bit: L = 1 for BL (Branch and Link, ignore for now)

Branch with L = 0

31:28	27:26	25:24	23:0
cond	10	10	imm24

- op = 10
- imm24 = 24-bit **signed** immediate
- The two bits 25:24 form the funct field
  - Bit 25 is always 1
  - L bit: L = 0 for B (Branch)

- Format

  B TARGET

  ↓

  B imm24

Branch Target Address (BTA)

31:28	27:26	25:24	23:0
cond	10	10	imm24

- The 24-bit two's complement **imm24** field specifies the instruction address relative to **PC + 8**
  - Why **PC + 8**? **(**historical reasons**)**
- **BTA**: The address of the next instruction to execute if the branch is taken
- The imm8 field is the number of instructions between the BTA and PC + 8 (two instructions past the branch)

BTA Calculation Example

suppose PC points here → PC

PC + 4

PC + 8

3 instructions

address					
0x80A0		BLT	THERE		
0x80A4		ADD	R0,	R1,	R2
0x80A8		SUB	R0,	R0,	R9
0x80AC		ADD	R3,	R0,	R1
0x80B0		ORR	R3,	R2,	R1
0x80B4	THERE	ADD	R1,	R1,	#78
0x80B8		ADD	R3,	R3,	#0x5

31:28	27:26	25:24	23:0
cond	10	10	imm24 = 3 (000000000000000000000011)

BTA Calculation

The processor calculates the BTA in three steps

1. Shift left imm24 by 2 (to convert words to bytes)
2. Sign-extend (copy $Instr_{25}$ into $Instr_{31:26}$)
3. Add PC + 8

																0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Summary of Formats

	31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
DP-I	cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

	31:28	27:26	25	24:21	20	19:16	15:12	11:4							3:0
DP-R	cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

	31:28	27:26	25:20						19:16	15:12	11:0
Mem	cond	01	1	1	1	0	0	L	Rn	Rd	imm12

	31:28	27:26	25:24	23:0
BR	cond	10	10	imm24

Exercise – 1

- Write the 32-bit ARM machine language representation for the following instruction?
  - SUB R8, R9, R10
  - For SUB, cmd is 0010; for unconditional exec, cond = 1110

	31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
DP-R	cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

Exercise – 1

- Write the 32-bit ARM machine language representation for the following instruction?
  - SUB R8, R9, R10
  - For SUB, cmd is 0010; for unconditional exec, cond = 1110

	31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
DP-R	cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

	31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
SUB	1110	00	0	0010	0	1001	1000	0	0	0	0	0	0	0	0	1010

HEX: E0 49 80 0A

Exercise – 2

- Write the 32-bit ARM machine language representation for the following instruction?
  - ADD R0, R1, #42
  - For ADD, cmd is 0100; for unconditional exec, cond = 1110

	31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
DP-I	cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

Exercise – 2

- Write the 32-bit ARM machine language representation for the following instruction?
  - ADD R0, R1, #42
  - For ADD, cmd is 0100; for unconditional exec, cond = 1110

	31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
DP-I	cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

	31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
ADD	1110	00	1	0100	0	0001	0000	0	0	0	0	00101010

HEX: E2 81 00 2A

Exercise – 3

- Write the 32-bit ARM machine language representation for the following instructions?
  - LDR R7, [R0, #16]

RISC vs. CISC Architectures

- Reduced Instruction Set Computer (RISC)
  - Small # of simple instructions
  - Simple hardware (e.g., easy to decode)
  - ARM, RISC-V, QuAC, MIPS
- Complex Instruction Set Computer (CISC)
  - Many more instructions, and some instructions are very complex
  - x86 DP/ALU instructions can have memory operands
  - One CISC instruction = Many RISC instructions
  - Complex instructions are costly to implement

RISC Principles

Simplicity favors regularity

- Consistent number of operands in DP instructions

Make the common case fast

- Include a few frequently used instructions

Smaller is faster

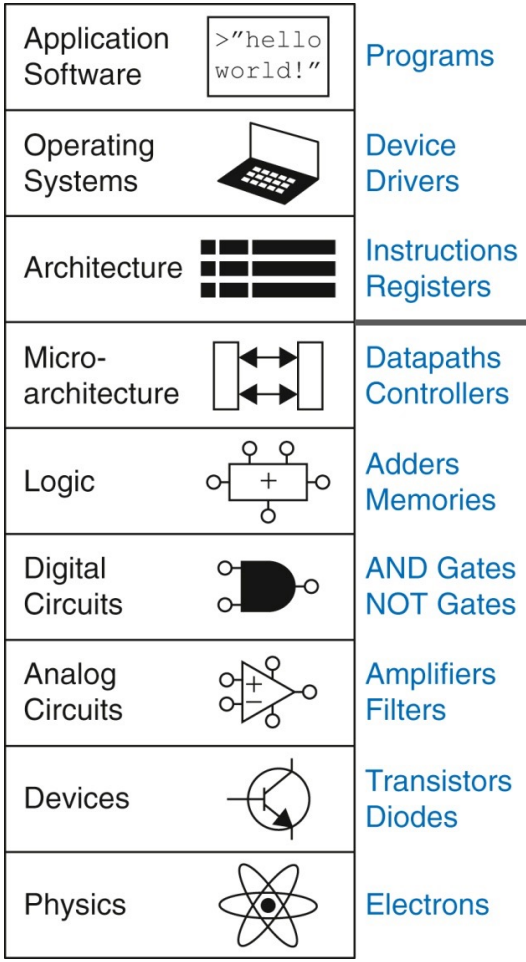
- Use a small and fast register file
- Large # registers = Large decoding circuitry

Good design demands good compromises

- Simplicity encourages a single instruction format, but that is too restrictive
- Define multiple formats, but have some regularity among instructions to simplify decoding

Patterson & Hennessy
2018 Turing Award

The Big Picture



- *The compiler translates high-level programs to assembly*
- *The assembler translates assembly code to machine code*

Software

Hardware

ISA is the Sw/Hw boundary (Contract/agreement)

- *Machine code resides in main memory*
- *CPU fetches/executes instructions*
- *Microarchitecture is the specific arrangement of adders, memories, registers, etc., to implement an ISA*

Plan for Weeks 6 – 7

Week 5

- We simplified the ISA by setting some control bits to 0 or 1

Week 6

- Build a CPU for the simplified ARM 32-bit instruction set

Week 7

- Study the remaining ISA features

Microarchitecture

- Two interacting parts
  - Datapath
  - Control unit
- Datapath *operate* on words of data
  - *Register file, ALU, memories*
- Control unit *informs* the datapath how to execute an instruction
  - *Reads the instruction and generate multiplexer selects, register enable, and memory write signals*

LDR Instruction

- We will add the logic for one instruction at a time beginning with the LDR instruction

- Format of LoaD Register instruction

  LDR R0, [R1, #12]

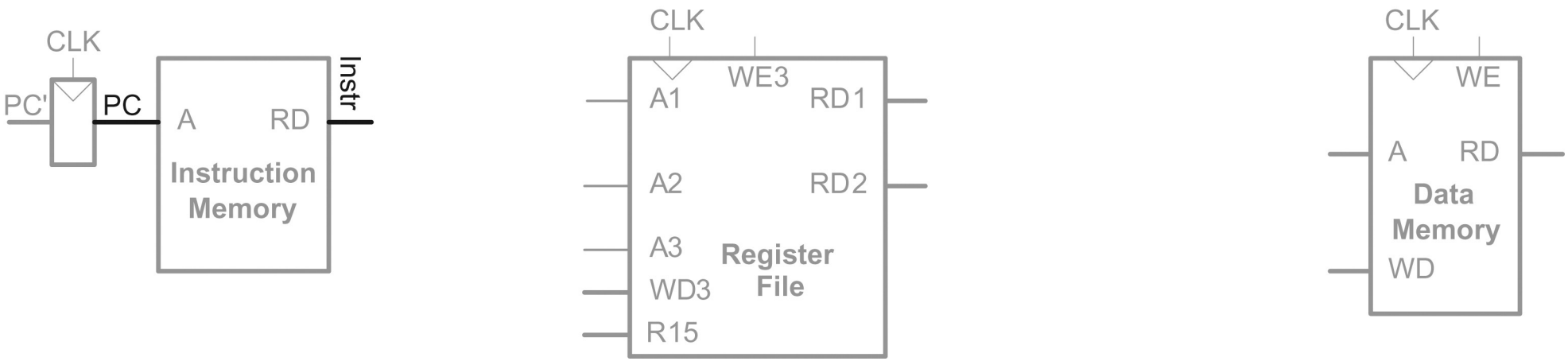
  ↓ ↓ ↓

  LDR Rd, [Rn, #imm12]

31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

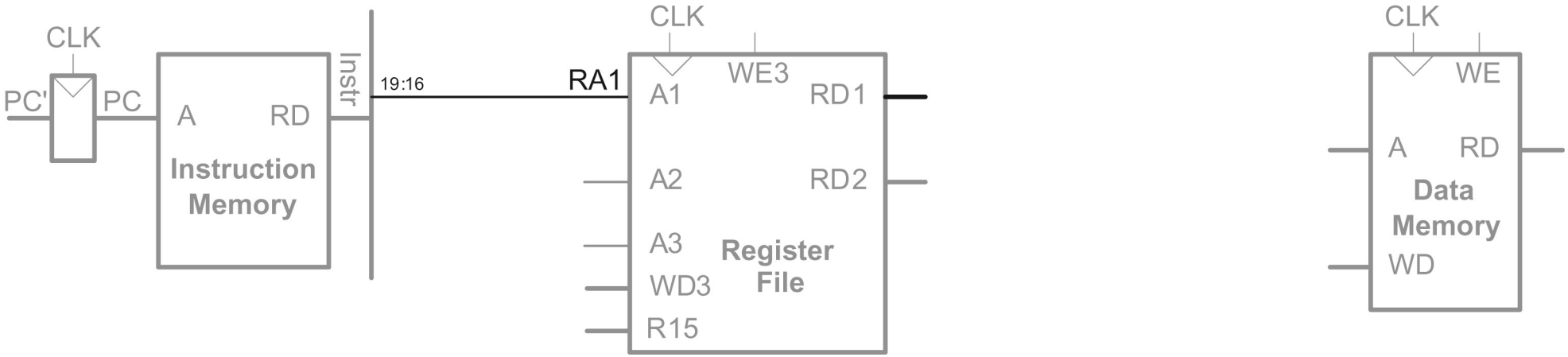
Step 1: Read (Fetch) instruction from memory



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

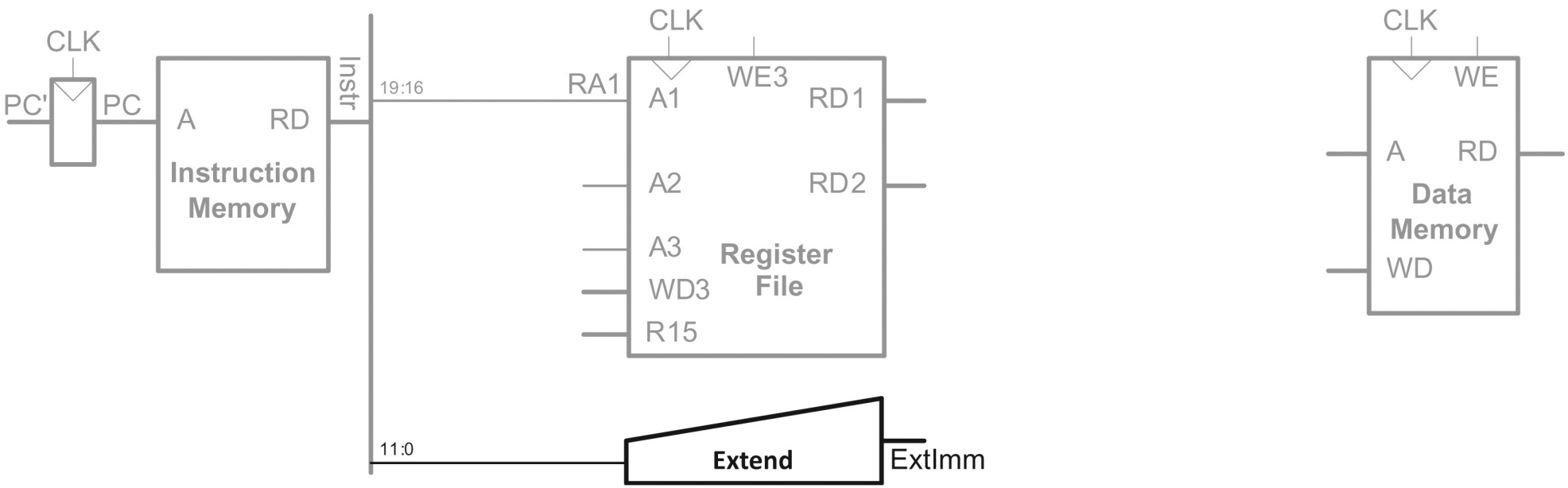
Step 2: Read source operand (base register) from register file



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

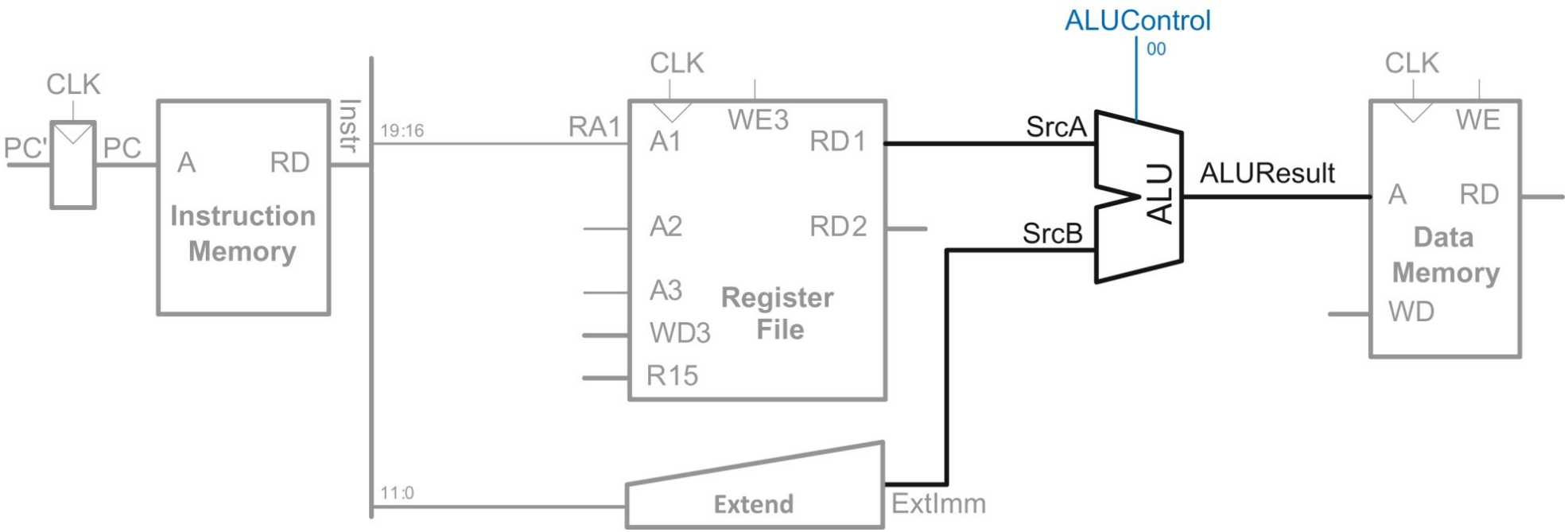
Step 3: Zero-extend the immediate field stored in $Instr_{11:0}$



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

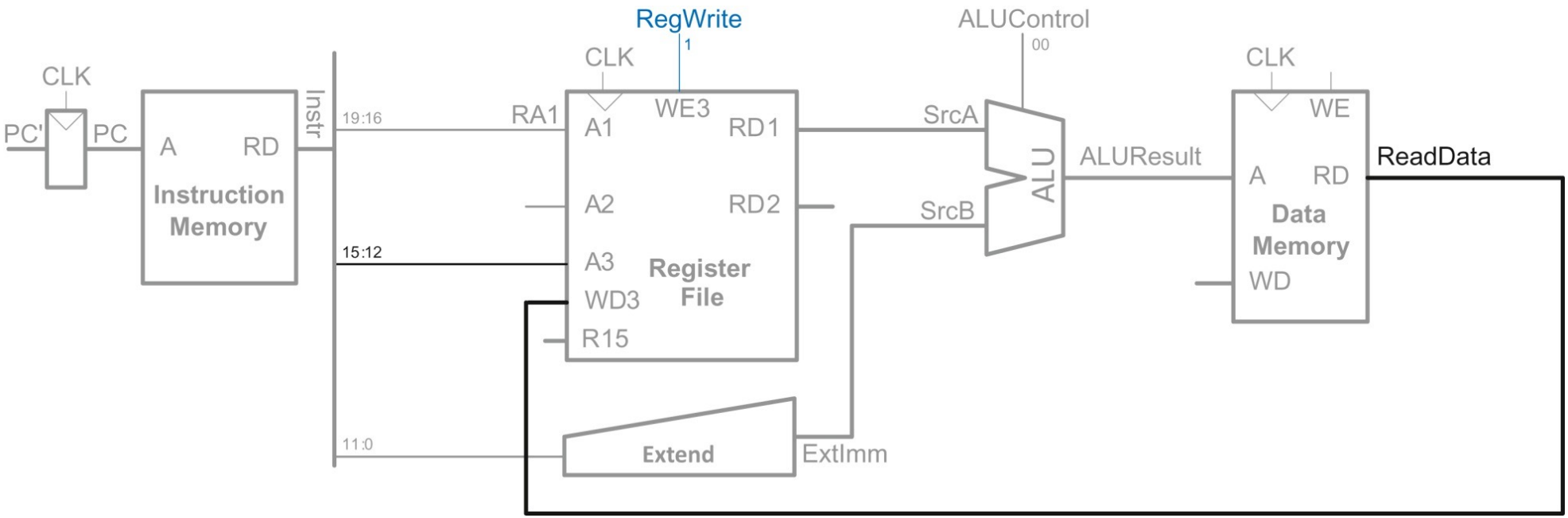
Step 4: Compute memory address (ALUControl = 00)



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

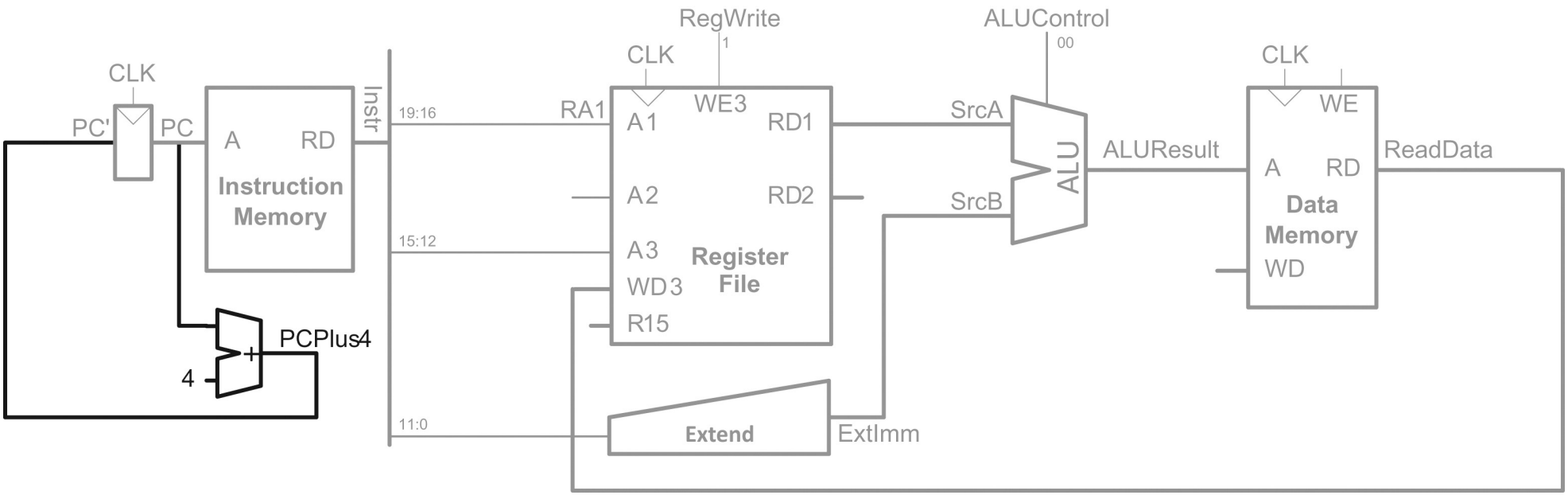
Step 5: Write back data from read by data memory to Rd in Reg File



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

Step 6: Compute address of next instruction (PC' = PC + 4)



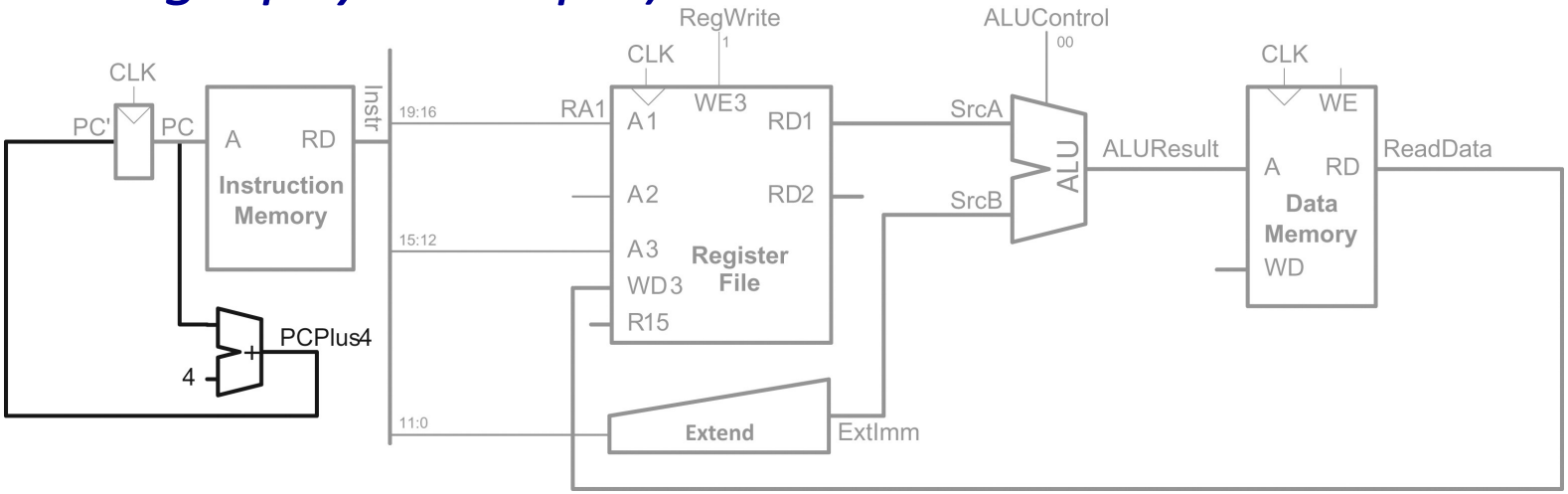
PC will become PC' the following cycle (recall photography example)

31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

Step 6: Compute address of next instruction (PC’ = PC + 4)

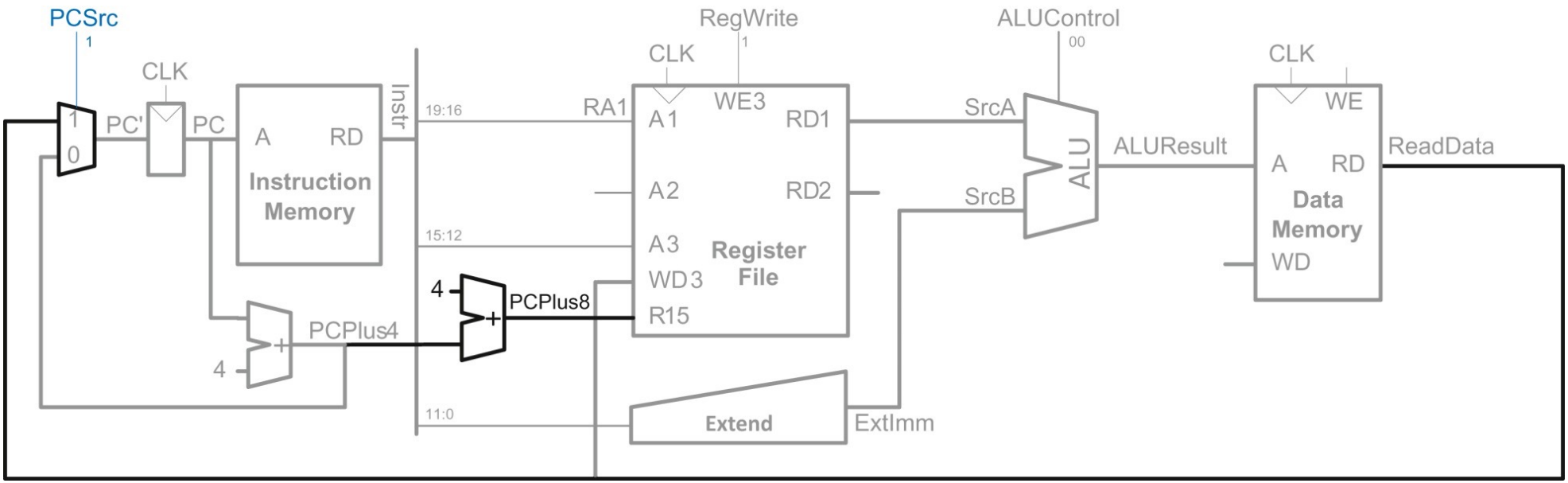
Important: PC will become PC’ the following cycle (recall photography example)



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

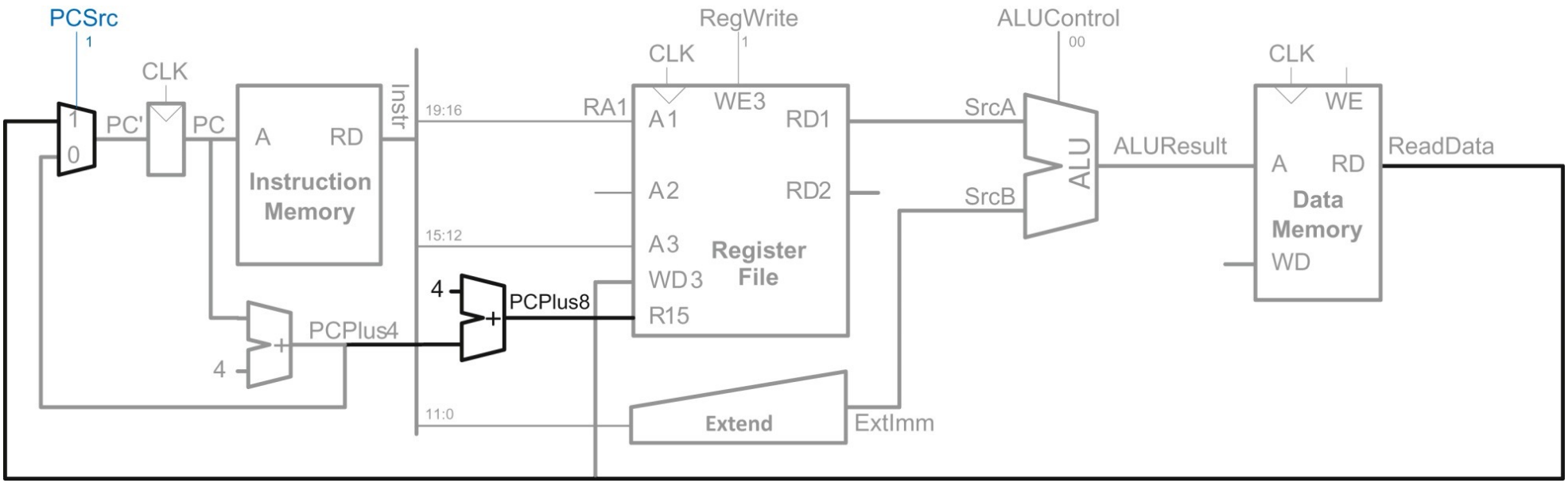
Step 7/a: Reading register R15 returns PC + 8



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The LDR Datapath

Step 7/b: Writing register R15 (PC may be an instruction's result)



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

STR Instruction

- STR instruction uses the same instruction format
- LDR and STR behave differently at the machine level
- Rd is a source operand (specifies the register to store to mem)

- Format of STore Register instruction

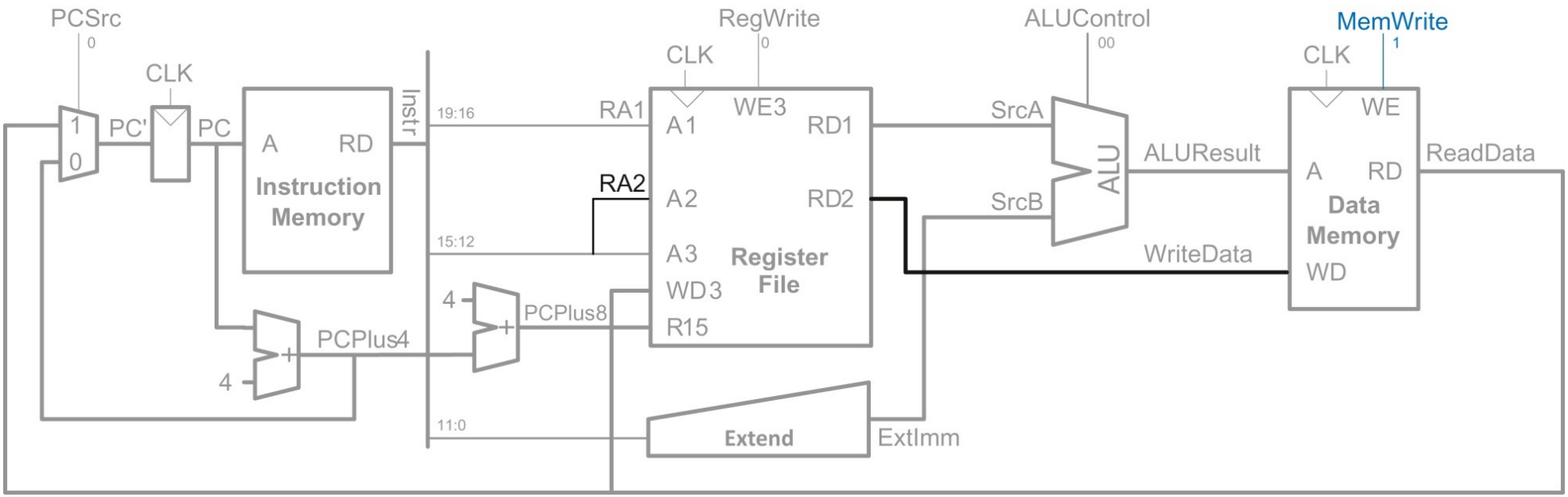
STR R0, [R1, #12]

STR Rd, [Rn, #imm12]

31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

The STR Datapath

Step 8: Read a second register (Rd) and write its value to memory



31:28	27:26	25:20						19:16	15:12	11:0
cond	01	1	1	1	0	0	L	Rn	Rd	imm12

DP Instructions: Immediate

- Like the LDR instruction, but two important differences
  - imm8 instead of imm12
  - The destination register stores the result of the ALU operation instead of memory access
- Format

ADD R0, R1, #16

↓ ↓ ↓

ADD Rd, Rn, #imm8

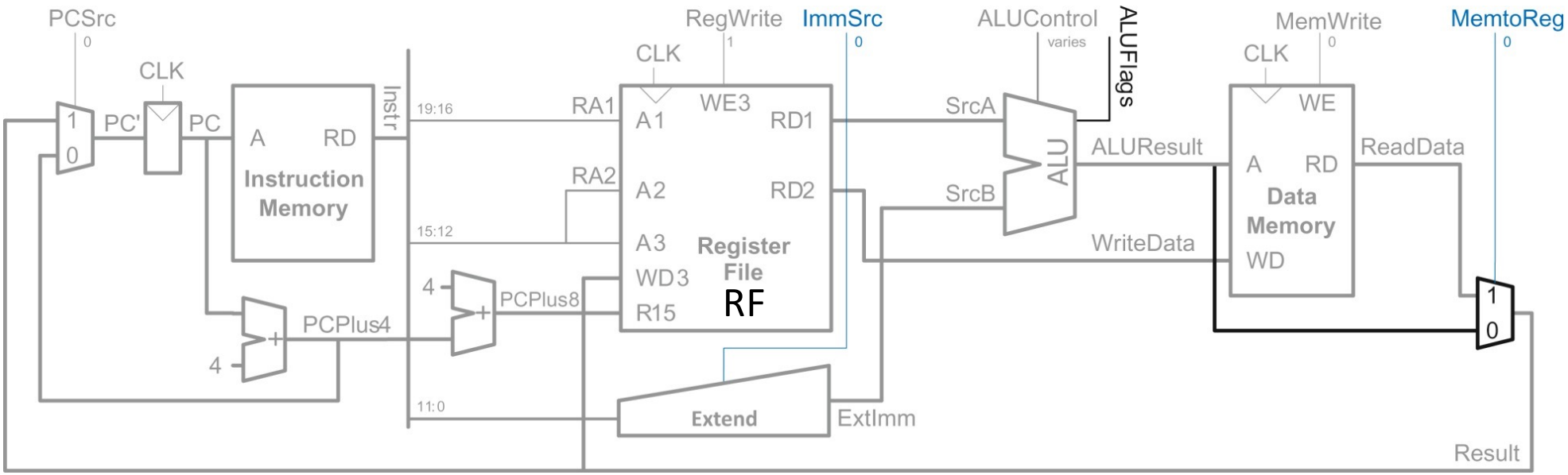
31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

Recall the ALU Functions

ALUControl	Function
00	ADD
01	SUB
10	AND
11	ORR

DP-Immediate Datapath

Step 9: Change extend block, and add signal to write ALU result to RF



31:28	27:26	25	24:21	20	19:16	15:12	11:8				7:0
cond	00	1	cmd	S	Rn	Rd	0	0	0	0	imm8

DP Instructions: Register

- The second source operand is Rm instead of an immediate
- Place Rm on the A2 port of the register file for DP instructions with register as the second operand

- Format

ADD R0, R1, R3

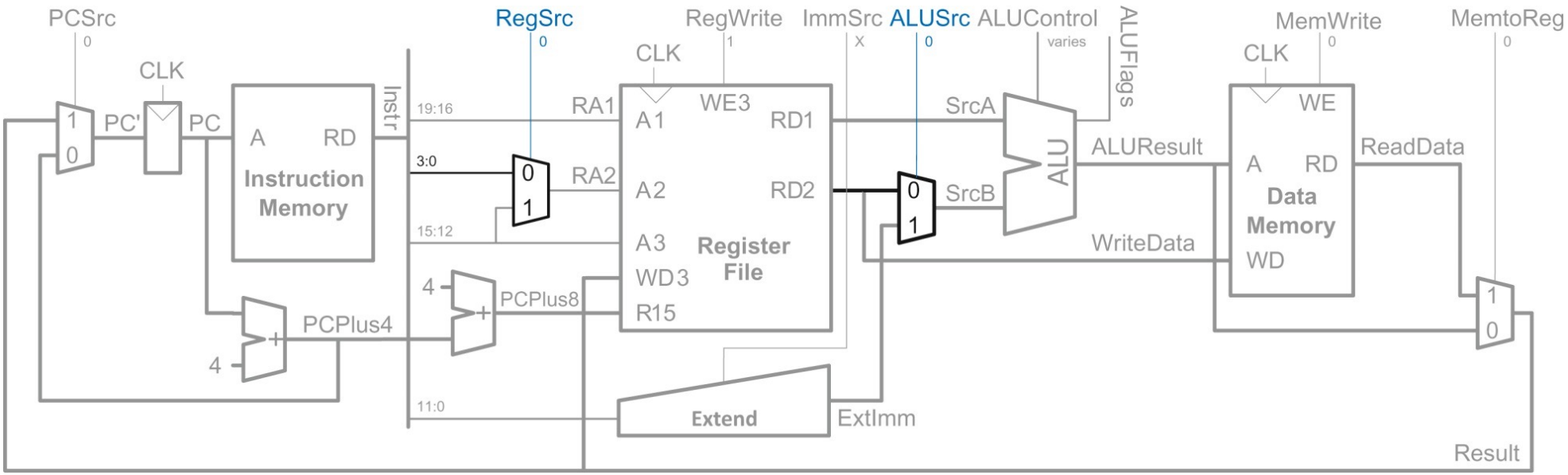
↓ ↓ ↓

ADD Rd, Rn, Rm

31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

DP-Register Datapath

Step 10: Read 2nd register (Rm) from Reg File and send RD2 to ALU



31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

Branch Instruction: Unconditional

- The second source operand is Rm instead of an immediate
- Place Rm on the A2 port of the register file for DP instructions with register as the second operand

- Format

  B TARGET

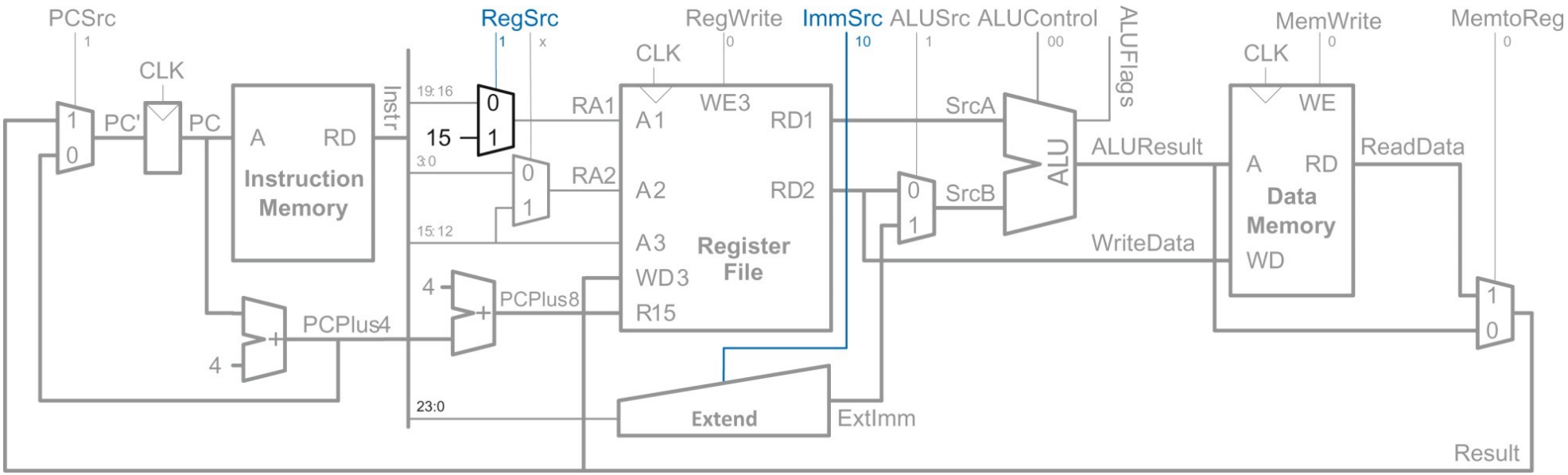
  ↓

  B imm24

31:28	27:26	25:24	23:0
1110	10	10	imm24

Branch Datapath

Step 11: Change extend block, and add a bit to RegSrc for branch



31:28	27:26	25:24	23:0
1110	10	10	imm24

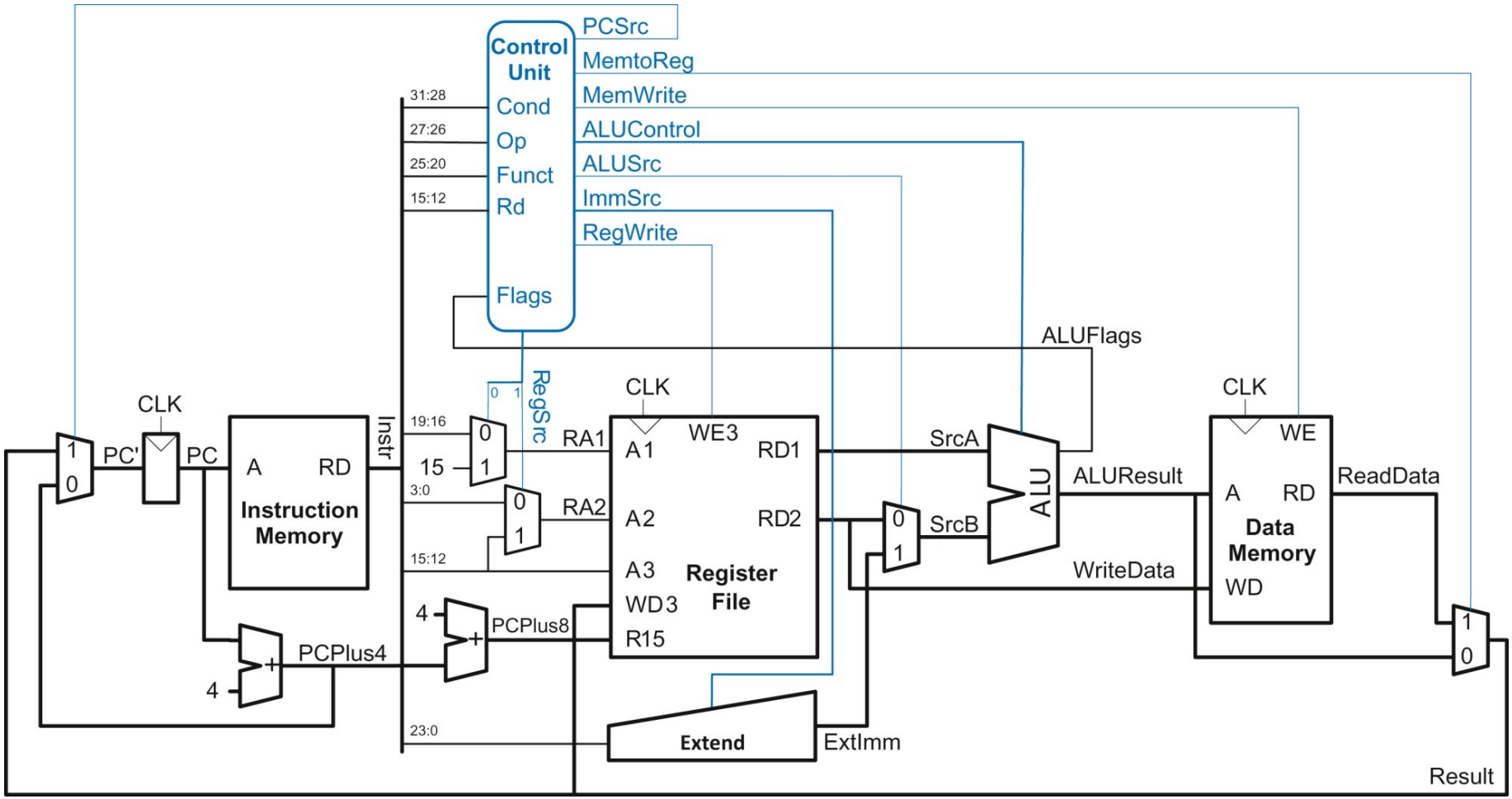
Operation of the Extend Block

Each of the three instruction formats interpret the immediate field differently

- $ImmSrc_{1:0}$ is the 2-bit control signal input to the extend block

$ImmSrc_{1:0}$	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*

Datapath w/t Control



Control Unit

- Generate control signals based on instruction fields
  - $Instr_{31:20}$ (cond)
  - $Instr_{27:26}$ (op)
  - $Instr_{25:20}$ (funct)
  - Flags
  - Destination register (*why does the controller needs this?*)
- Controller for "*our microarchitecture*" is purely combinational
- Things to ponder
  - Ensure correct PC update (branch or Rd == R15)
  - Update state (Reg file, Data memory) based on conditional execution

Decoder Truth Table

- Only selected signals are shown in the truth table

Op	$Funct_5$	$Funct_0$	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1
00	1	X	DP Imm	0	0	0	1	00	1	X0	1
01	X	0	STR	0	X	1	1	01	0	10	0
01	X	1	LDR	0	1	0	1	01	1	X0	0
11	X	X	B	1	0	0	1	10	0	X1	0