CE/CZ 3001:
Advanced Computer Architecture

(Module 3: Data-path and Control Design)

Dr Smitha K. G.
School of Computer Science
And Engineering

Topics covered till now

- LEGv8 ISA:
  - Instruction format
    - R-format, D-format, I- format, CB format and B-format
  - Addressing modes
    - Register addressing
    - Base addressing
    - Immediate addressing
    - PC-relative addressing
  - Functionality
    - ALU instructions
    - Data transfer instructions
    - Conditional and unconditional instructions

Outline

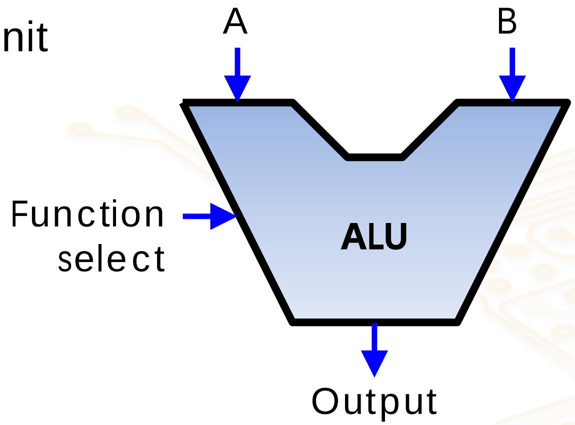
- Review of basic logic devices and register file

- Single-cycle datapath design

Basic datapath components

Arithmetic and logic unit (ALU)

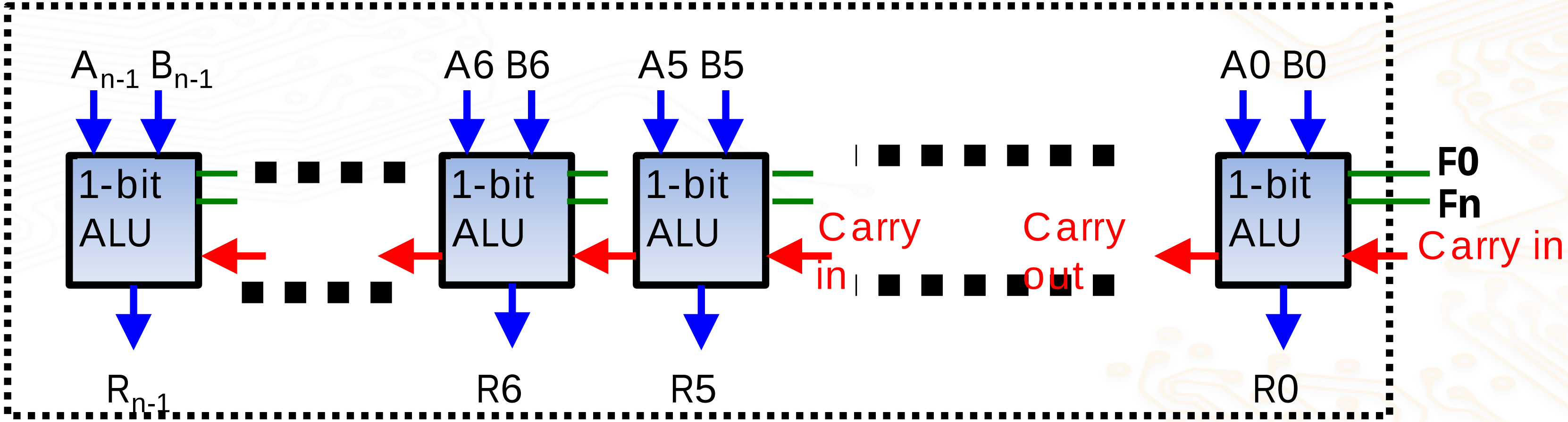
- ALU - is the CPU's data processing or execution unit
  - implements fixed point operations.
- The floating point and complex numerical functions are performed by arithmetic co-processors
- Typical arithmetic functions: Add, subtract, multiply, divide.
- Logic functions: AND, OR, XOR, and NOT.
- Other data manipulation functions: Arithmetic and logical shifts, incrementing and decrementing the operands etc.



Arithmetic and logic unit

- *n*-bit ALU can be made by placing *n*-1-bit ALU slices in parallel

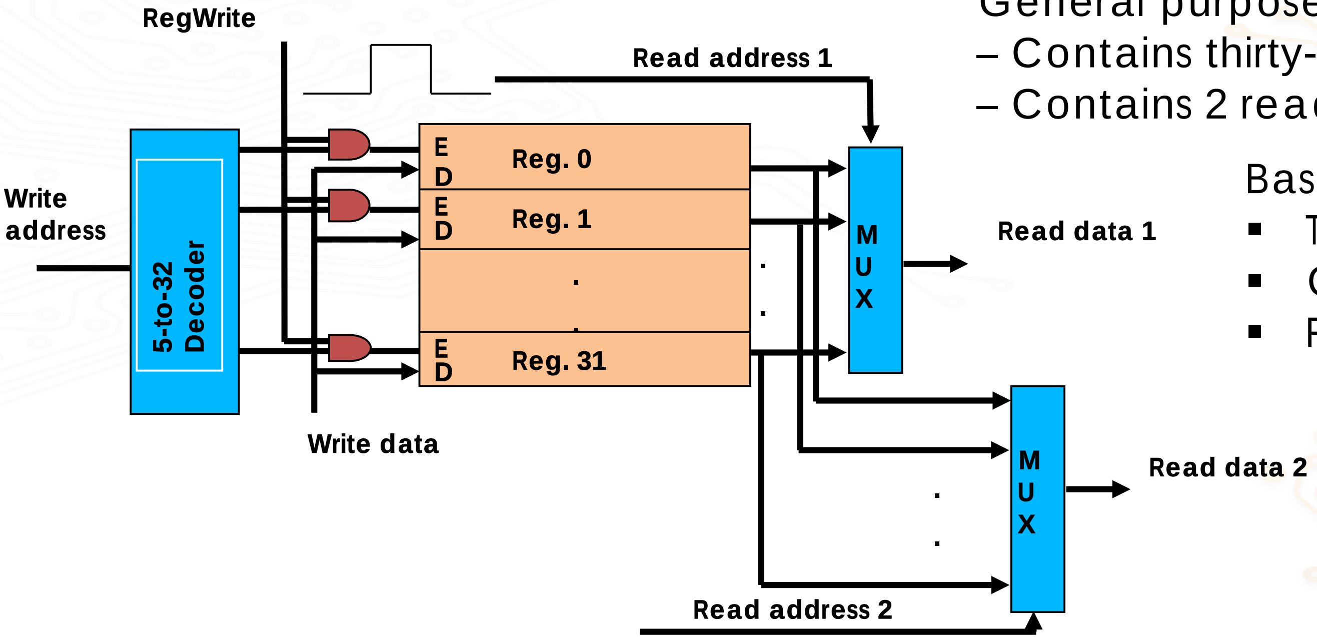
- The function select (F0..Fn) for each 1-bit ALU is driven by the instruction being executed.



R=A op B

Register File

A register file is a collection of registers in which any register can be read or written by specifying the address. They form the temporary storage inside the CPU.



General purpose register (GPR) file

– Contains thirty-two 64-bit registers

– Contains 2 read ports and 1 write port

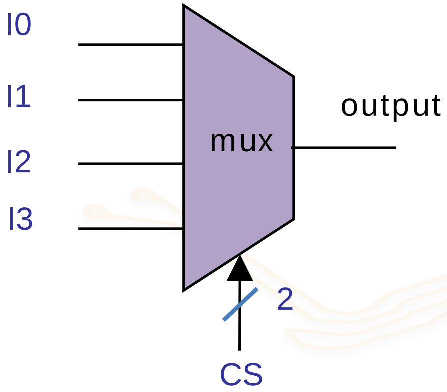
Basic building blocks

- Two 32:1 Multiplexor
- One 5 to 32 Decoder
- Register using Flipflop

Building blocks of Register File (Part 1/3)

Multiplexer:

The multiplexer used is 32:1 MUX in MIPS register file. But for explanation we are using 4:1 MUX. A 4: 1 MUX selects one of the 4 inputs according
to value of the select line (CS)



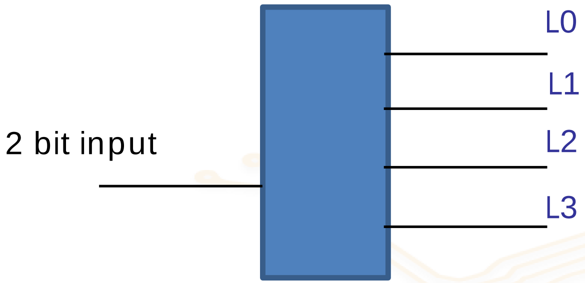
CS	Output
00	I0
01	I1
10	I2
11	I3

Building blocks of Register File (Part 2/3)

Decoder:

The decoder used is 5 to 32 decoder in register file. But for explanation of a decoder we are using 2 to 4 decoder.

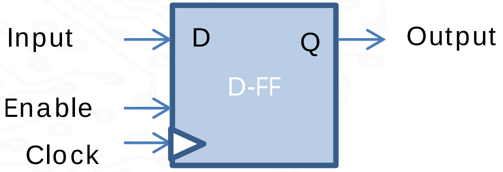
- A decoder having n input bits will have 2^n output bits.
- Only one output of a decoder is high at any given time depending on input.
- Output of the decoder “L0, L1, L2 and L3” can be used with **RegWrite signal** of a register file to write to the register file.



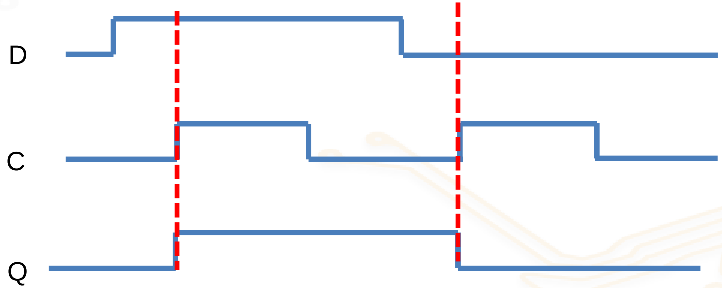
input	L0	L1	L2	L3
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

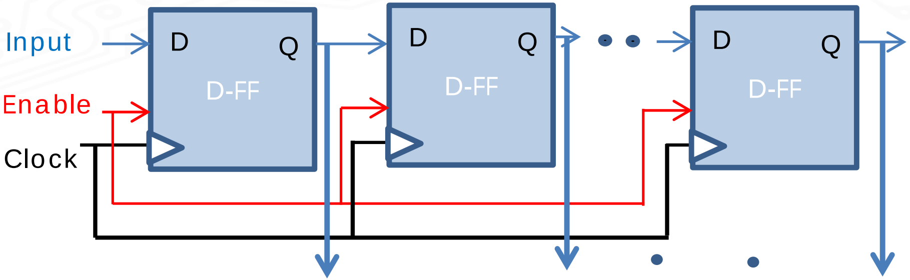
Building blocks of Register File (Part 3/3)

Register



One bit flipflop: Q← D , on the rising edge of the clock (+ve edge triggered Flipflop)

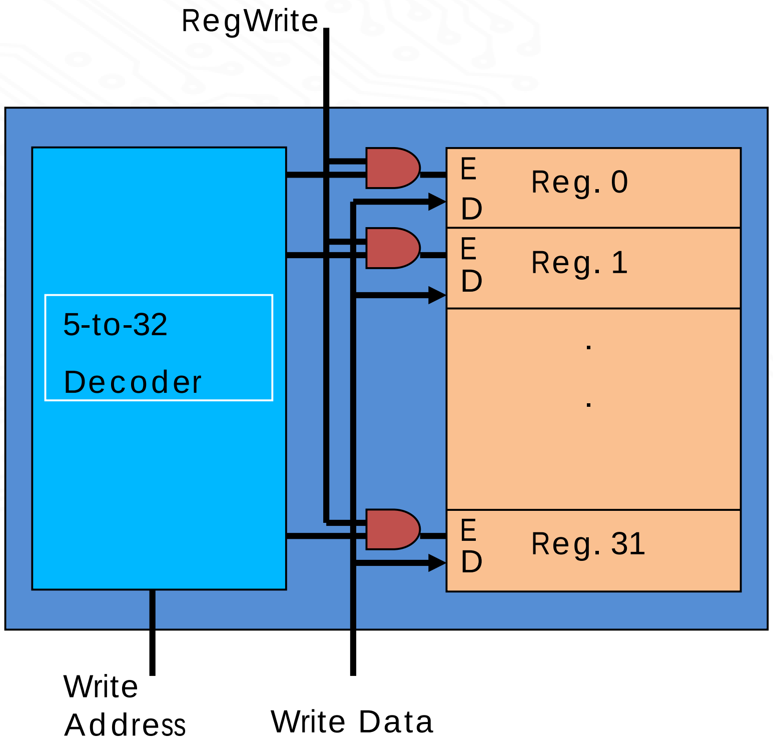


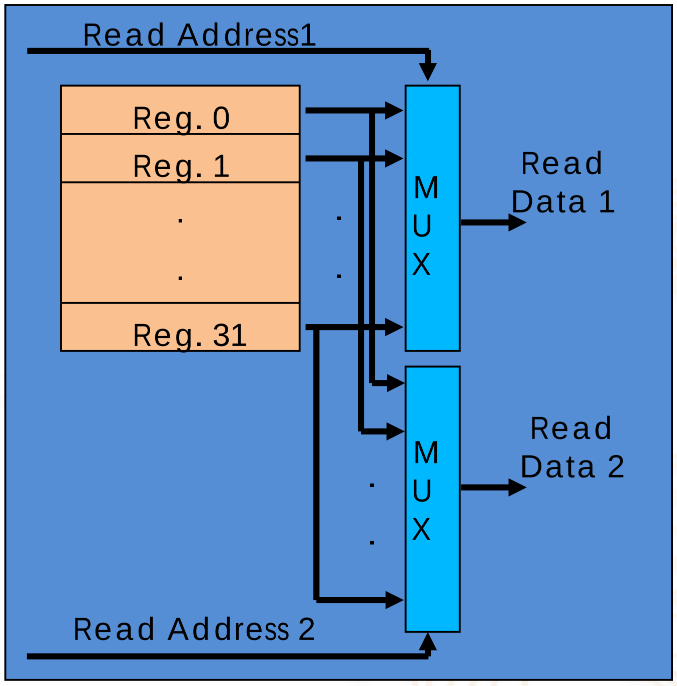


32 bit Output

- We can use an array of D- flipflops to build a register to that can hold multi-bit data such as bytes or words.
- We use these registers to build the datapath

Write and read port of MIPS Register File

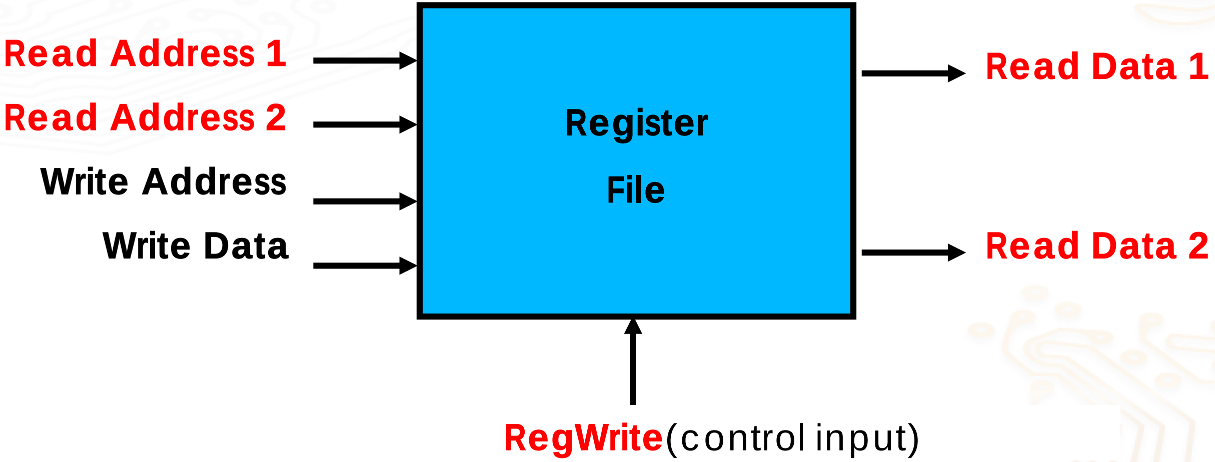




Read Ports for a Reg. File

Register File

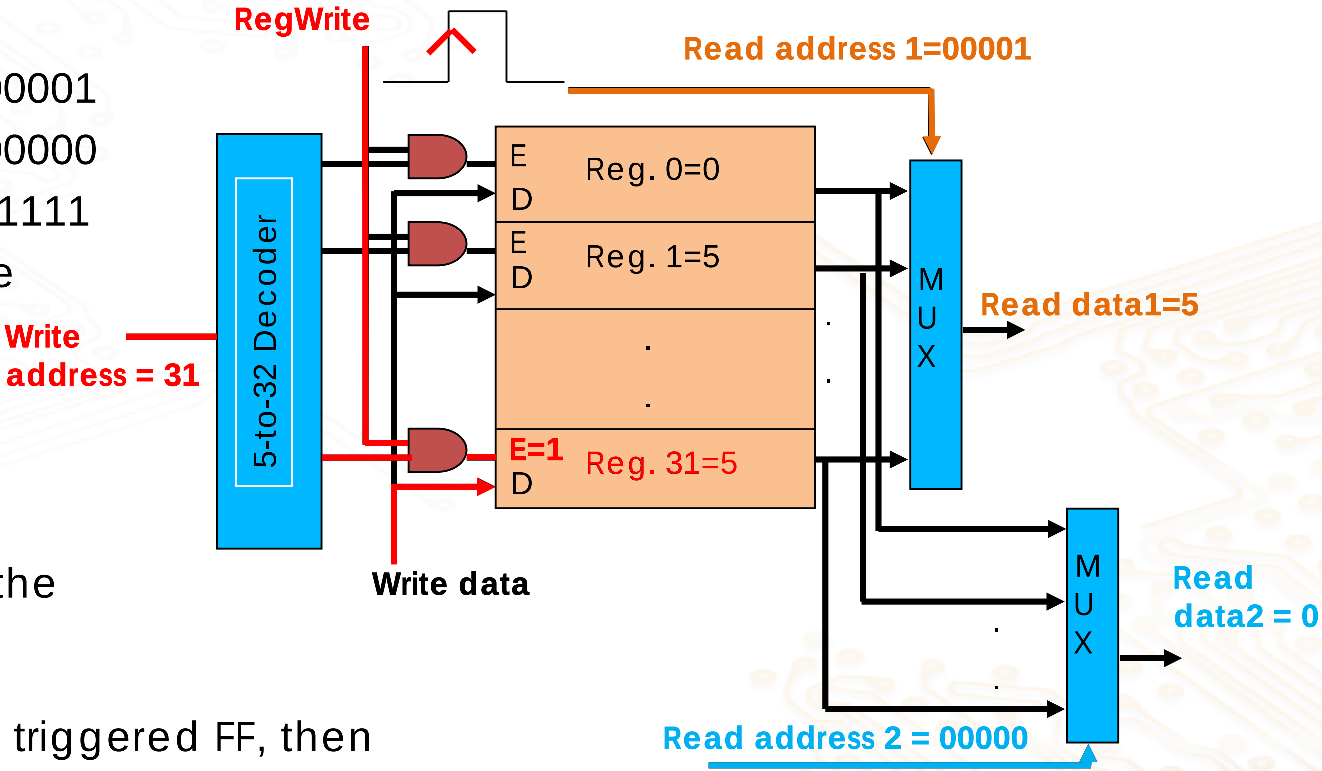
- Consists of a set of registers that can be read and written by supplying a register number to be accessed
- Can be implemented with a multiplexer for each read and a decoder to write and array of registers built from D-FF



Working of Register File

- Ex: ADD X31, X1, X0
- Read address1=1=00001
- Read address2=0=00000
- Write address=31=11111
- Assume Value inside

Reg. 0=0 and
Reg. 1 =5



- Do note that write happens only when RegWrite=1 and at the edge of the clock.

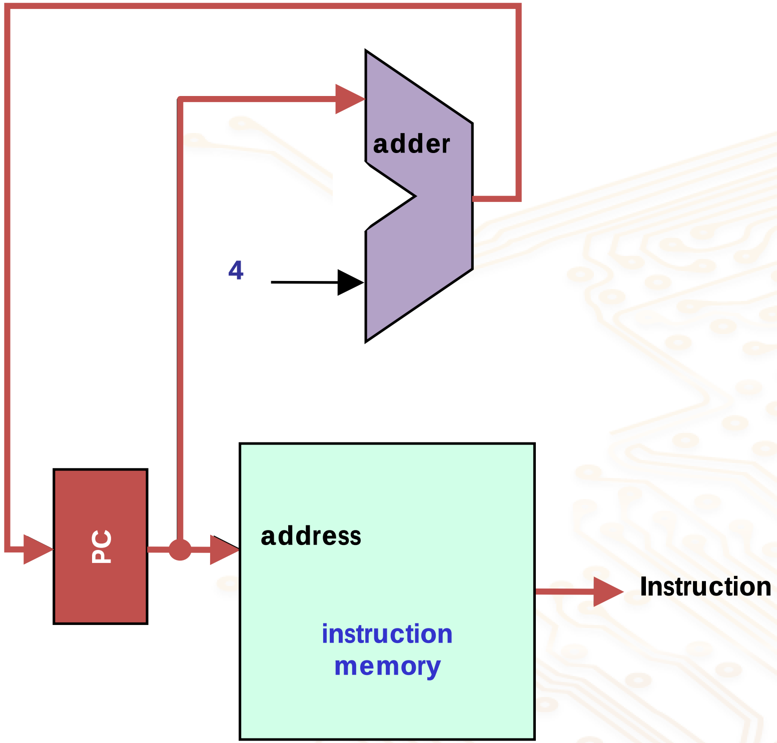
- So if it is a +ve edge triggered FF, then write occurs only at the positive edge of clock.

What happens when same register is read and written during same clock cycle?

- The register read happens combinational (no clock).
- The register will be valid during the time it is read.
- The value returned will be the value written in the earlier clock cycle.
- The write of the register file occurs on the clock edge.
- If we want to read to return the value currently being written- we need additional logic.

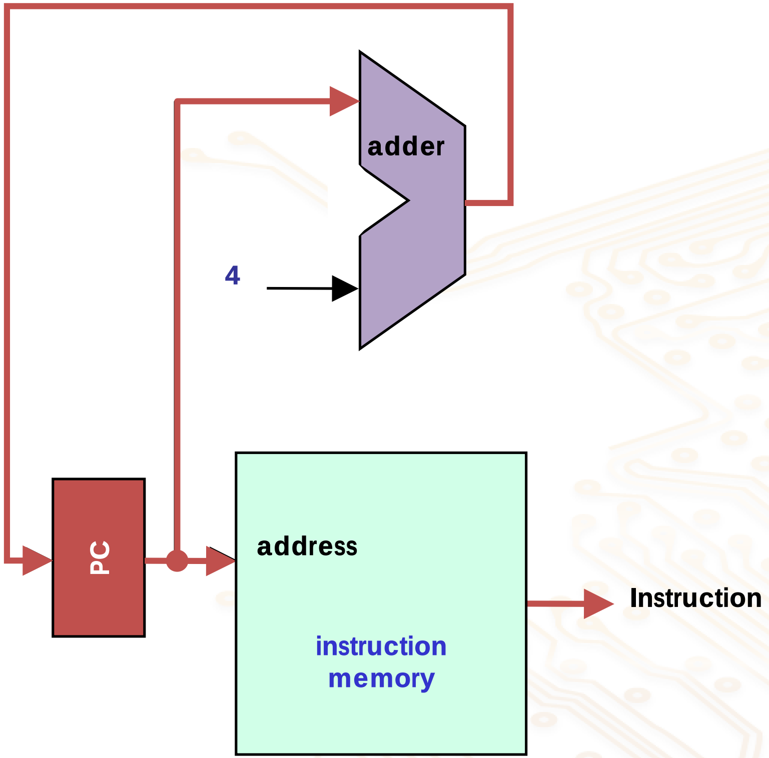
Introduction to datapath – Instruction fetch (Part 1/3)

- Instruction Memory: Memory to store the instructions of program and supply instructions given an address.
  - Needs only a read access
  - Treated as a combinational logic (output at any time reflects the contents of the location specified by the address input)
  - No read control signal is needed



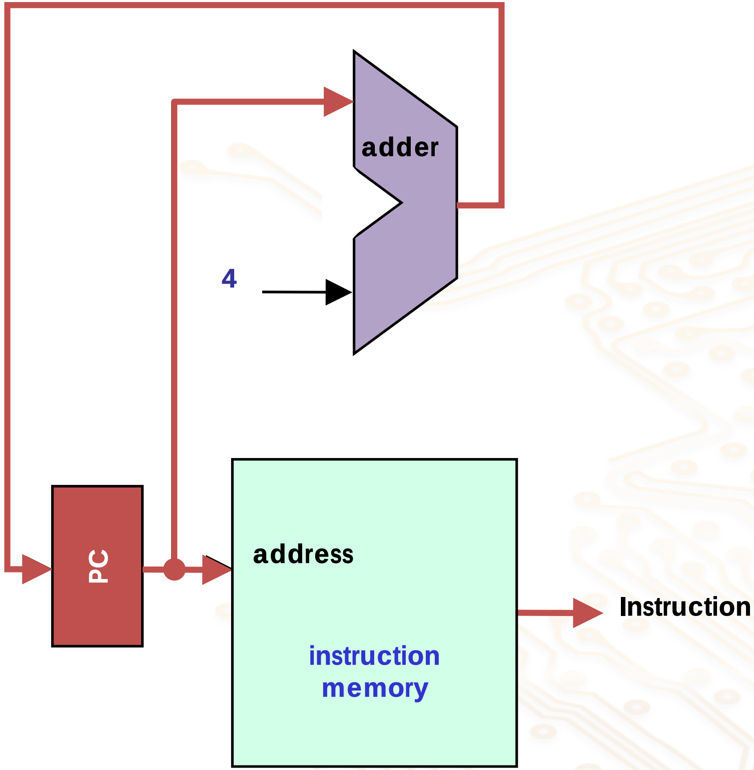
Introduction to datapath – Instruction fetch (Part 2/3)

- Program Counter (PC): Register that holds the address of the address of current instruction.
  - Register that is written at the end of every clock cycle



Introduction to datapath – Instruction fetch (Part 3/3)

- First, fetch the current instruction from instruction memory using PC.
- Second, prepare for the next instruction.
  - By incrementing PC by 4 to point to next instruction 4 bytes later



Datapath for R type instructions: (register addressing)

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

- **R-type** or (Register format)- All data values (operands/results) are located in registers and addressing mode is **Register addressing**

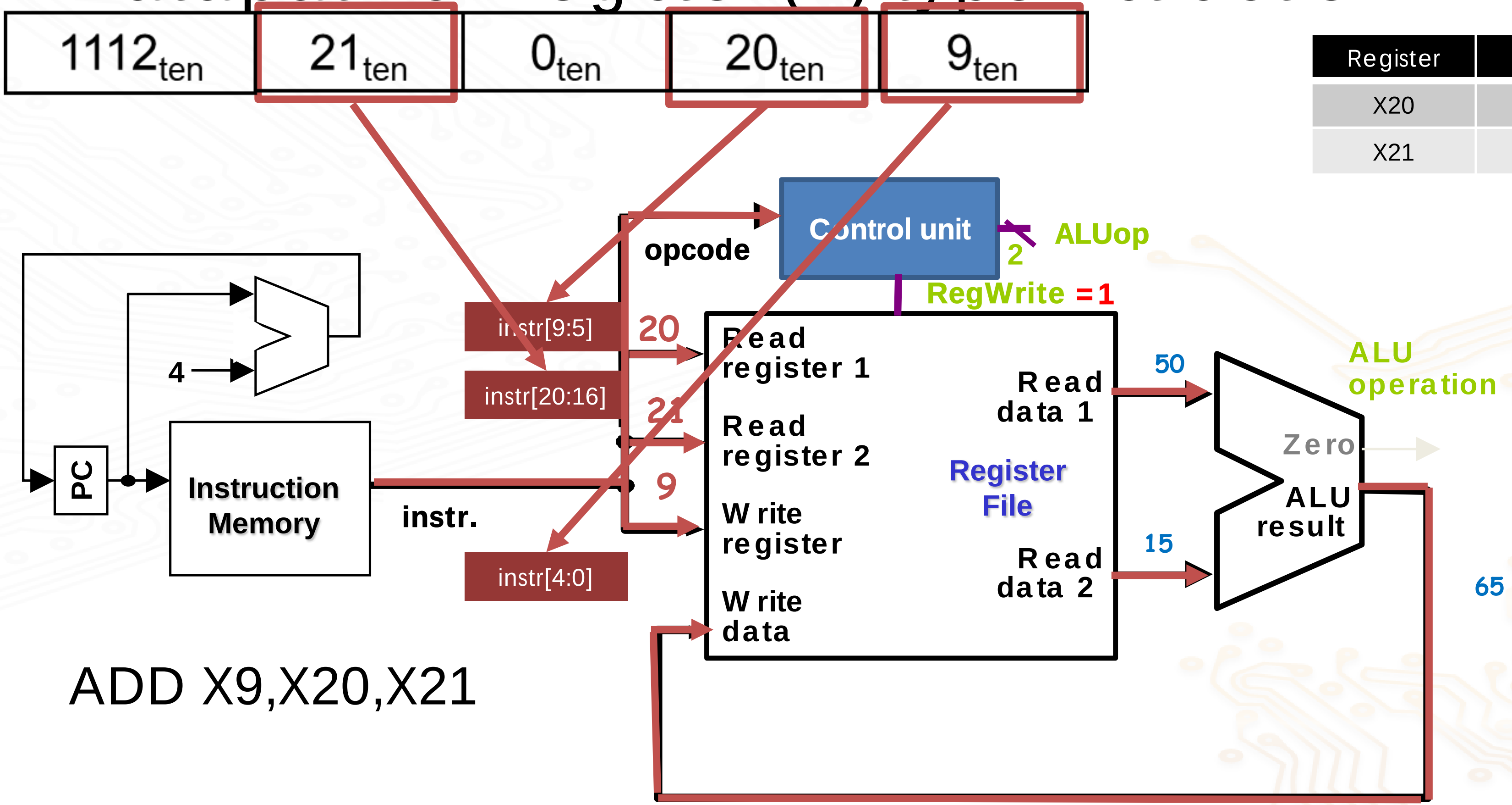
- The instruction is read from instruction memory (IF)

- The source registers specified by fields **Rn** and **Rm** of the instruction are read from the register file, and fed to the ALU (ID)

- ALU performs the operation specified in the opcode (EXE)

- The result is stored in the destination register, specified by field **Rd** of the instruction (WB)

Datapath of Register (R) type instruction



Register	Value
X20	50
X21	15

ADD X9,X20,X21

Adapted from Computer organization and design:
the hardware/software interface ARM edition , by D. A. Patterson, J. L. Hennessy & P. Alexander, 2017, Morgan Kaufmann.

Data transfer (D)type – Memory related instructions (Part 1/4)

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

- Memory access instructions: **LDUR and STUR**

- **LDUR Rt, [Rn, #address]**

  - Source register specified by field **“Rn”** is read from register file, and fed to ALU with “**address**” after sign-extension to 64-bits
  - ALU calculates the memory address of word to be loaded (EXE)
  - Content of the memory at location specified by the calculated address is read (MEM)
  - Word read from memory is loaded to the register specified by the address in “**Rt”** (WB)

Data transfer (D)type – Memory related instructions(Part 2/4)

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

- **STUR Rt, [Rn, #address]**
    - The field “**Rt**” in this case contains the address of the register that has the value to be stored: (while in case load instructions “**Rt**” was used as write address)
    - “**Rt**” is used as read address-2 for STU
    - The source register specified by “**Rn**” is read from register file, and fed to ALU with the sign extended operand “**address**” to calculate the address of memory location
    - ALU calculates the address of memory location where the data is to be stored (EXE)
    - Result is stored at the memory location whose address is calculated by the ALU (MEM)

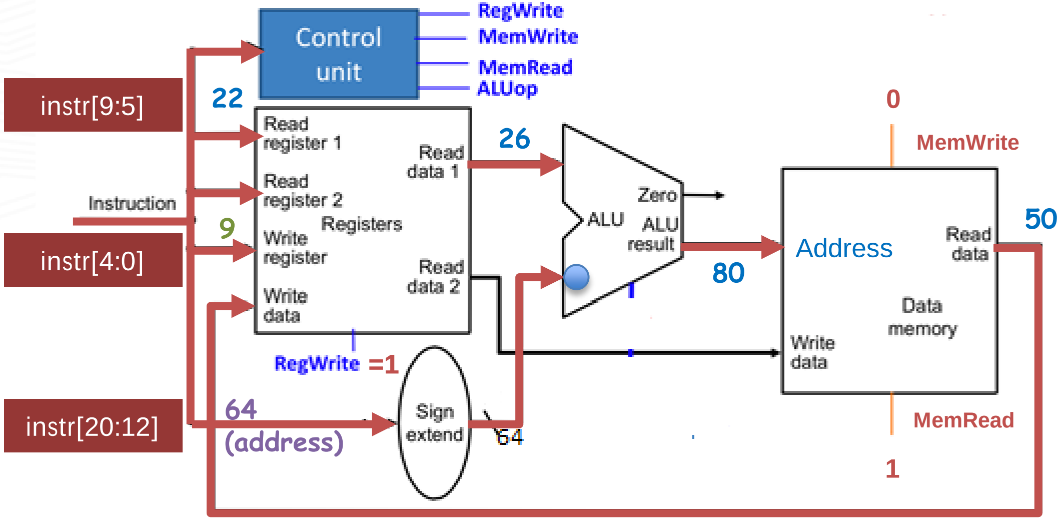
Data transfer (D)type – Base addressing Part 3/4)

- Operation **LDUR Rt, [Rn, #address]**
- LDUR X9, [X22, #64] //[X9] ←mem[[X22] +64]

1986	64	0	22	9
Opcode 11 bits	Address 9 bits	Op2 2 bits	Rn 5 bits	Rt 5 bits

Register	Value
X22	26

Memory[80]
50



Adapted from Computer organization and design:
the hardware/software interface ARM edition , by D. A. Patterson, J. L. Hennessy & P. Alexander, 2017, Morgan Kaufmann.

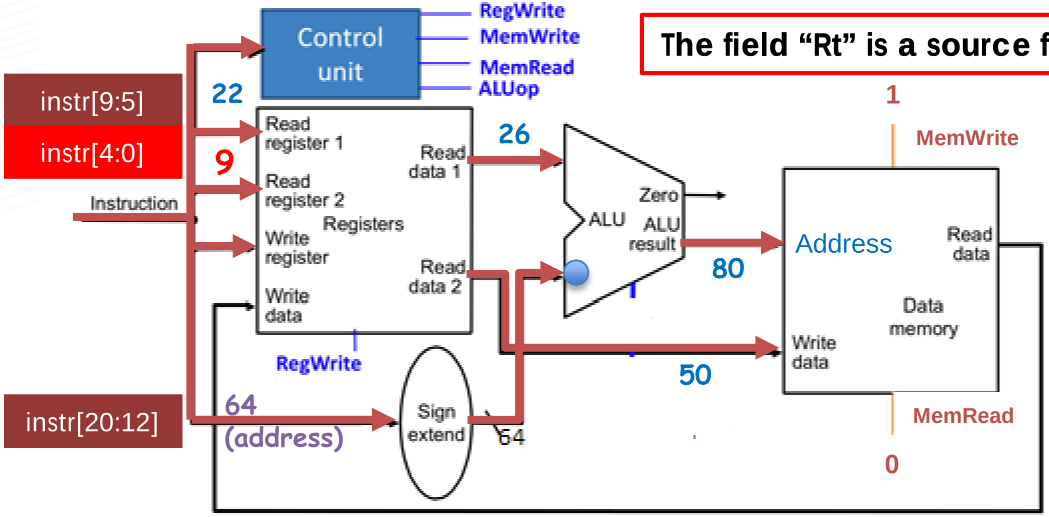
Data transfer (D)type – Base addressing Part 4/4)

- Operation **STUR Rt, [Rn, #address]**
- STUR X9, [X22, #64]] //[X9] →mem[[X22] +64]

1984	64	0	22	9
Opcode 11 bits	Address 9 bits	Op2 2 bits	Rn 5 bits	Rt 5 bits

Register	Value
X22	26
X9	50

Memory[80]
50



The field "Rt" is a source for SW

Adapted from Computer organization and design:
the hardware/software interface ARM edition , by D. A. Patterson, J. L. Hennessy & P. Alexander, 2017, Morgan Kaufmann.

Datapath for R and D-type Instructions

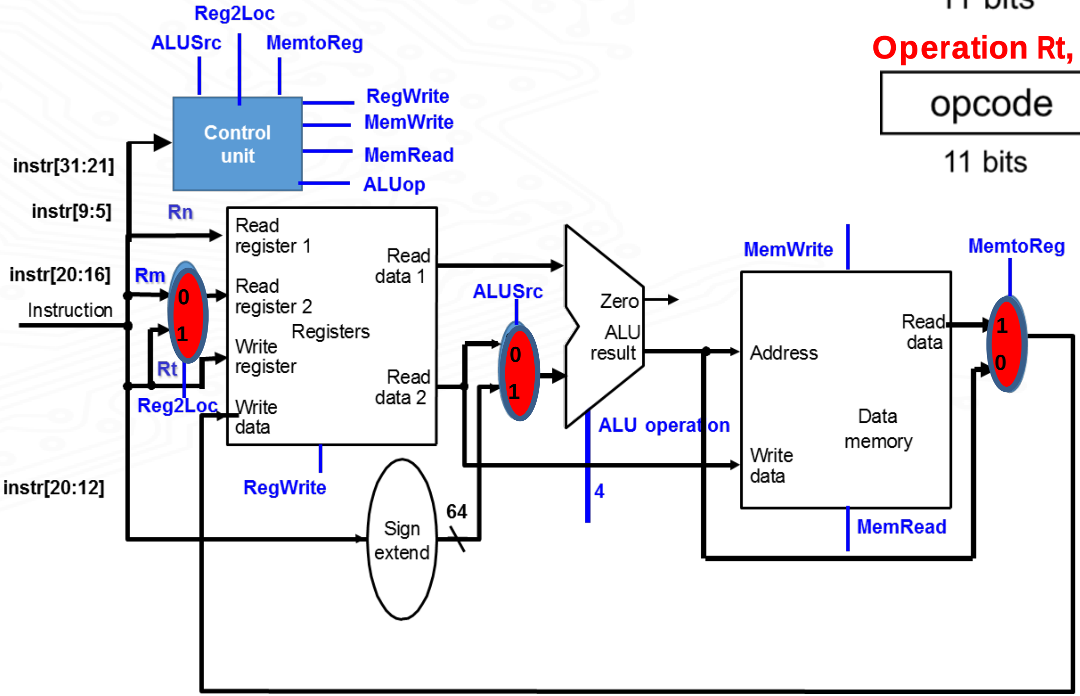
How can we combine the data path for R and D type instructions?

Operation Rd, Rn, Rm

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

Operation Rt, [Rn, address]

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits



"Rn" -Source for R and D type
"Rd" -Destination for R-type
"Rt" -destination/source for D-type

Extra mux needed

"Reg2Loc" -Selects between "Rt" and "Rm" as the source register address
"ALUSrc" -Selects between "read data2" and "address" as the source to ALU
"MemtoReg"-select the result from memory or from ALU

Adapted from Computer organization and design:
the hardware/software interface ARM edition , by D. A. Patterson, J. L. Hennessy & P. Alexander, 2017, Morgan Kaufmann.

Datapath for I type instructions: (Immediate addressing)

- **I-type** or (Immediate format)- one data in immediate field
- **immediate addressing**: Read from the register and immediate field and write to register

opcode	immediate	Rn	Rd
10 bits	12 bits	5 bits	5 bits

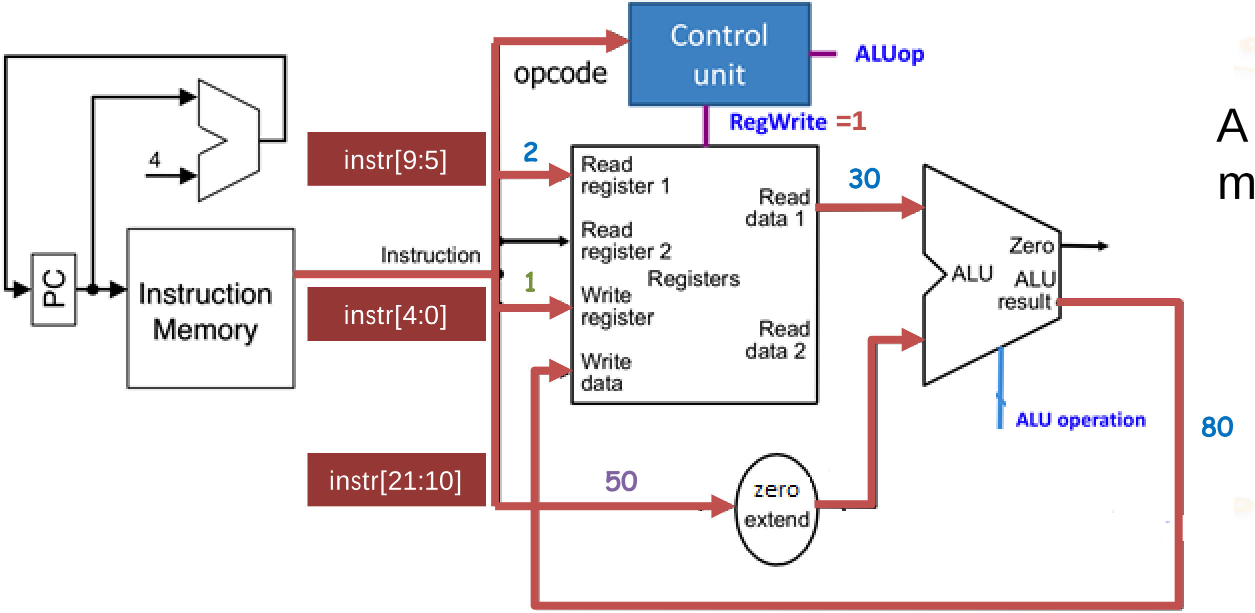
- Immediate field is zero-extended
- Opcode is reduced to 10 bits to have more range for immediate field.
- Example: ADDI X1, X2, #50 ([X1] ← [X2]+ 50])

580	50	2	1
Opcode 10 bits	immediate 12 bits	Rn 5 bits	Rd 5 bits

Datapath for I type – Immediate addressing

580	50	2	1
Opcode 10 bits	immediate 12 bits	Rn 5 bits	Rd 5 bits

Register	Value
X2	30



ADDI X1, X2, #50
meaning([X1] ← [X2]+ 50])

Adapted from *Computer organization and design: the hardware/software interface*, by D. A. Patterson, J. L. Hennessy & P. Alexander, 2015, Amsterdam: Morgan Kaufmann.

Conditional Branch Instructions

opcode	address	Rt
8 bits	19 bits	5 bits

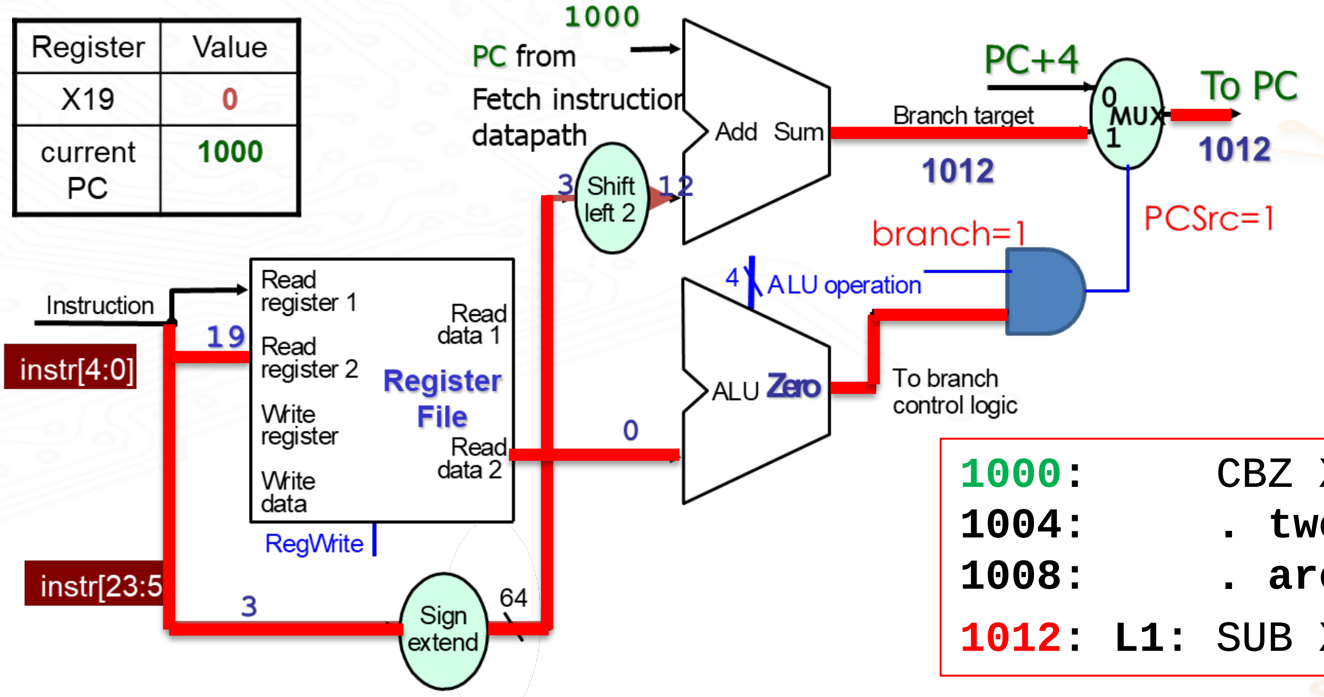
- Read register (Rt) operands
- Compare operands according to condition
  - For CBZ (Use ALU, and check Zero output)
- Calculate target address
  - Sign-extend displacement for “address”
  - Shift left 2 places (word displacement)
  - Add to PC (PC+ sign extend(address)<<2)

Datapath for CB type – Branch instruction

180	3	19
8 bits	19 bits	5 bits

Branch address in words

Register	Value
X19	0
current PC	1000



```
1000:       CBZ X19, L1 # L1=3
1004:       . two instructions
1008:       . are skipped
1012: L1: SUB X9,X22,x23
```

Adapted from *Computer organization and design: the hardware/software interface* , ARM edition, by D. A. Patterson, J. L. Hennessy & P. Alexander, 2015, Amsterdam: Morgan Kaufmann

Datapath for R , D, I and CB type Instructions

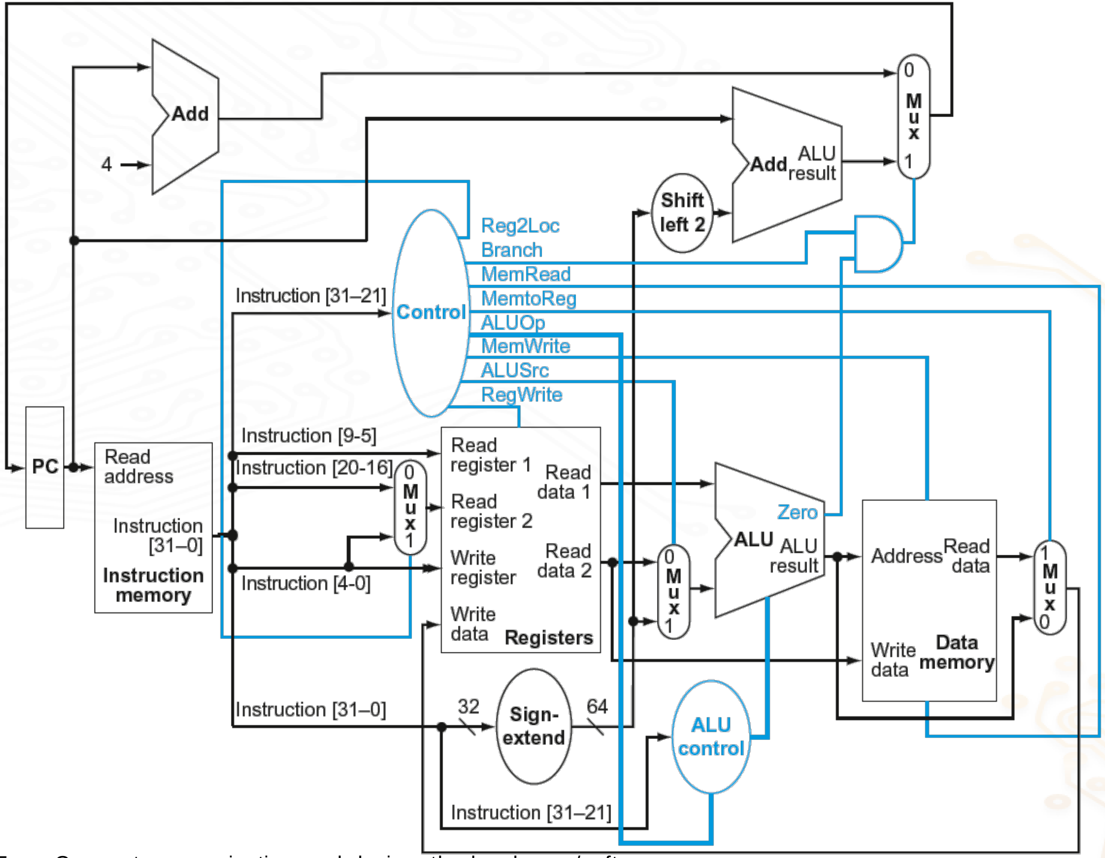
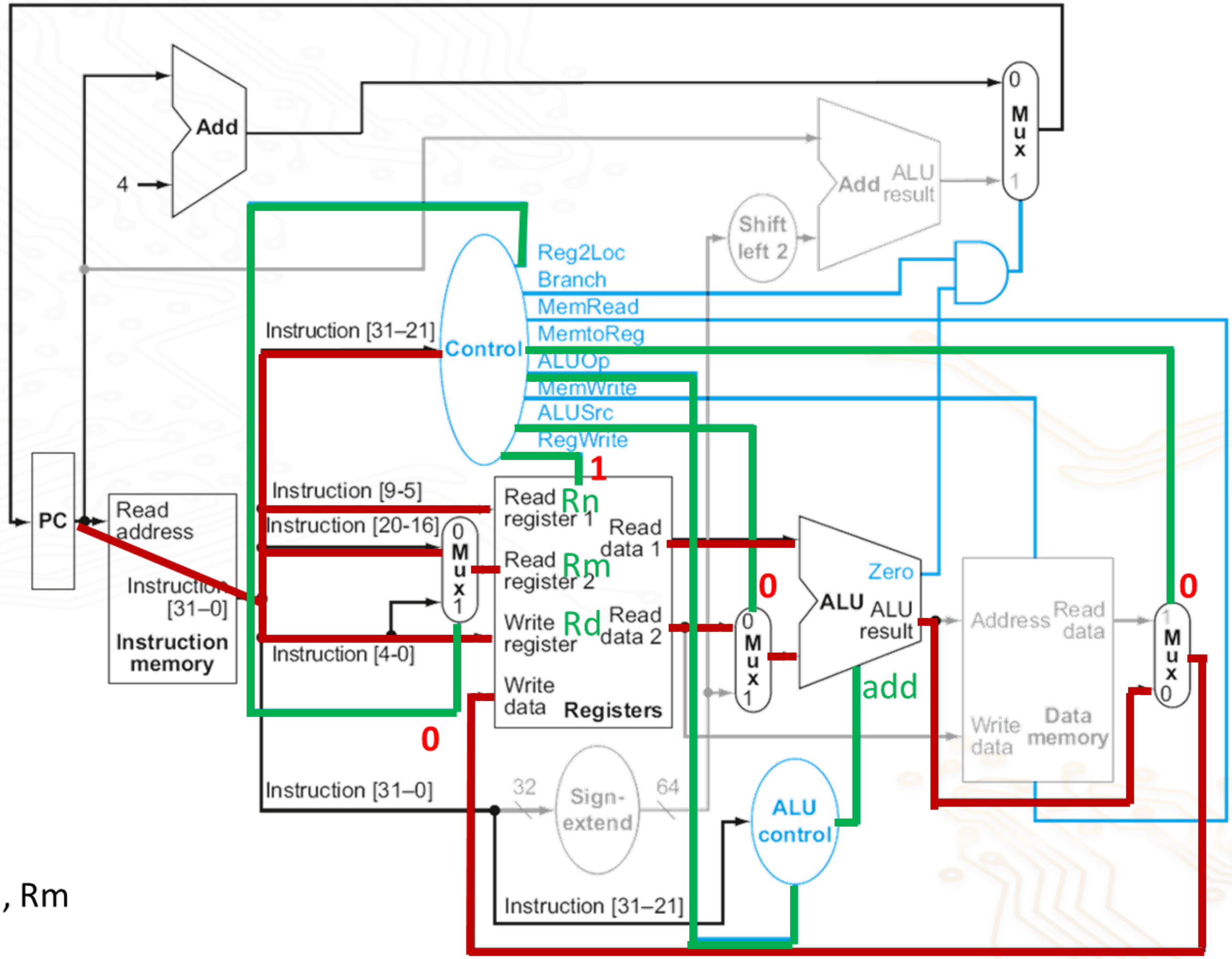


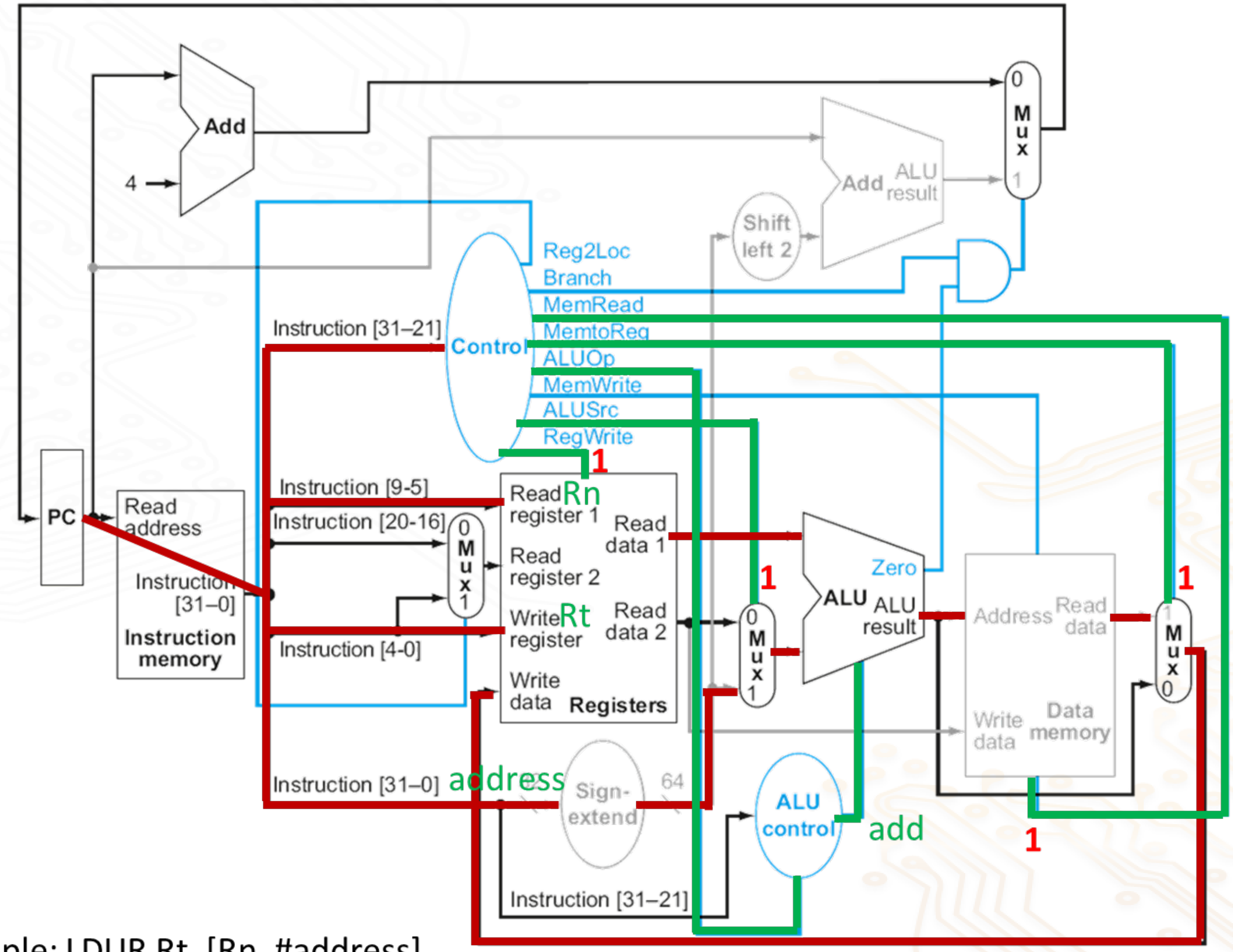
Figure 4.15, page 359 From Computer organization and design: the hardware/software interface ARM edition, by D. A. Patterson, J. L. Hennessy & P. Alexander, 2017 Amsterdam: Morgan Kaufmann.

R-type datapath



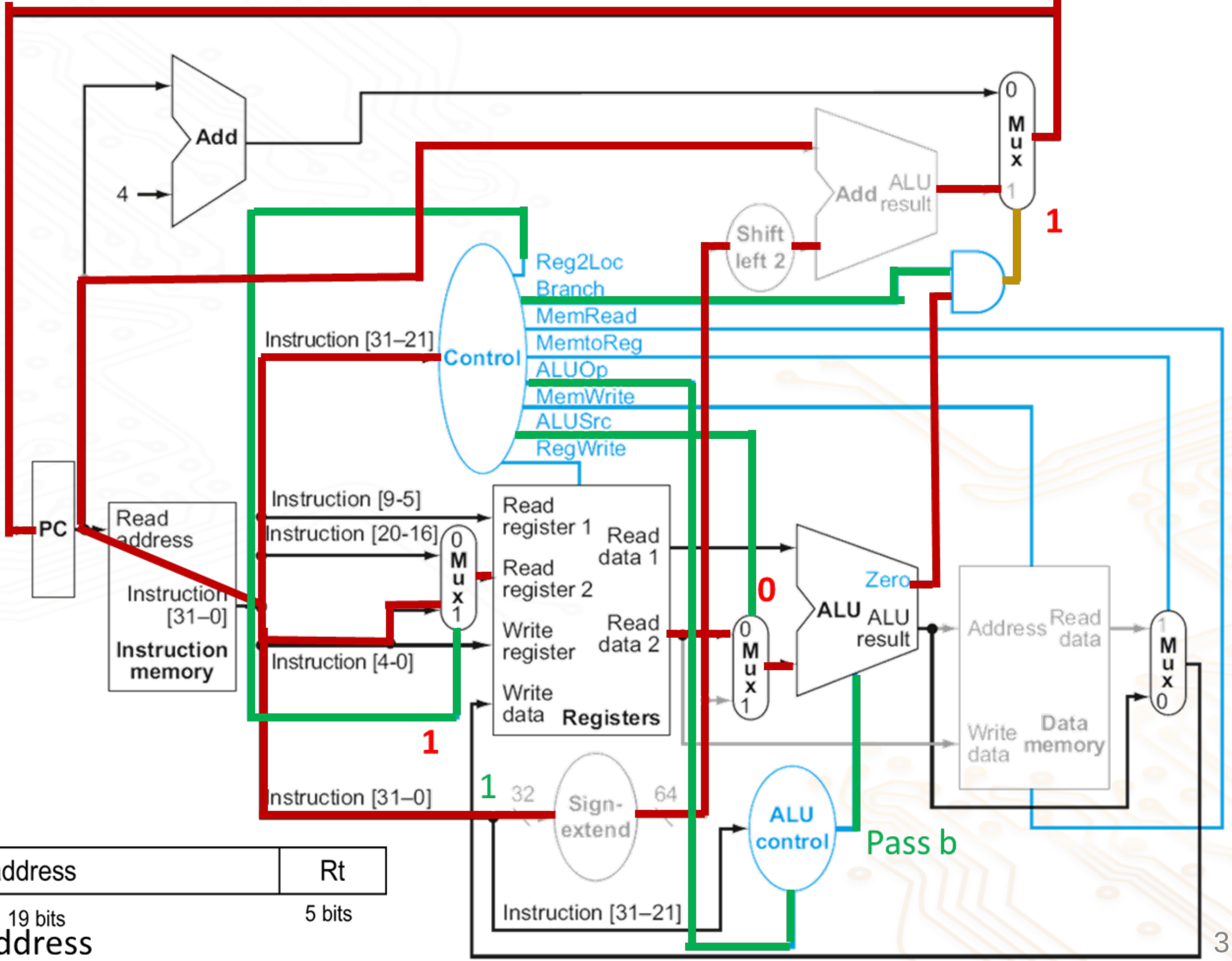
example: ADD Rd, Rn, Rm

Load (D-type) datapath



Example: LDUR Rt, [Rn, #address]

CBZ (CB-type) datapath



opcode	address	Rt
8 bits	19 bits	5 bits

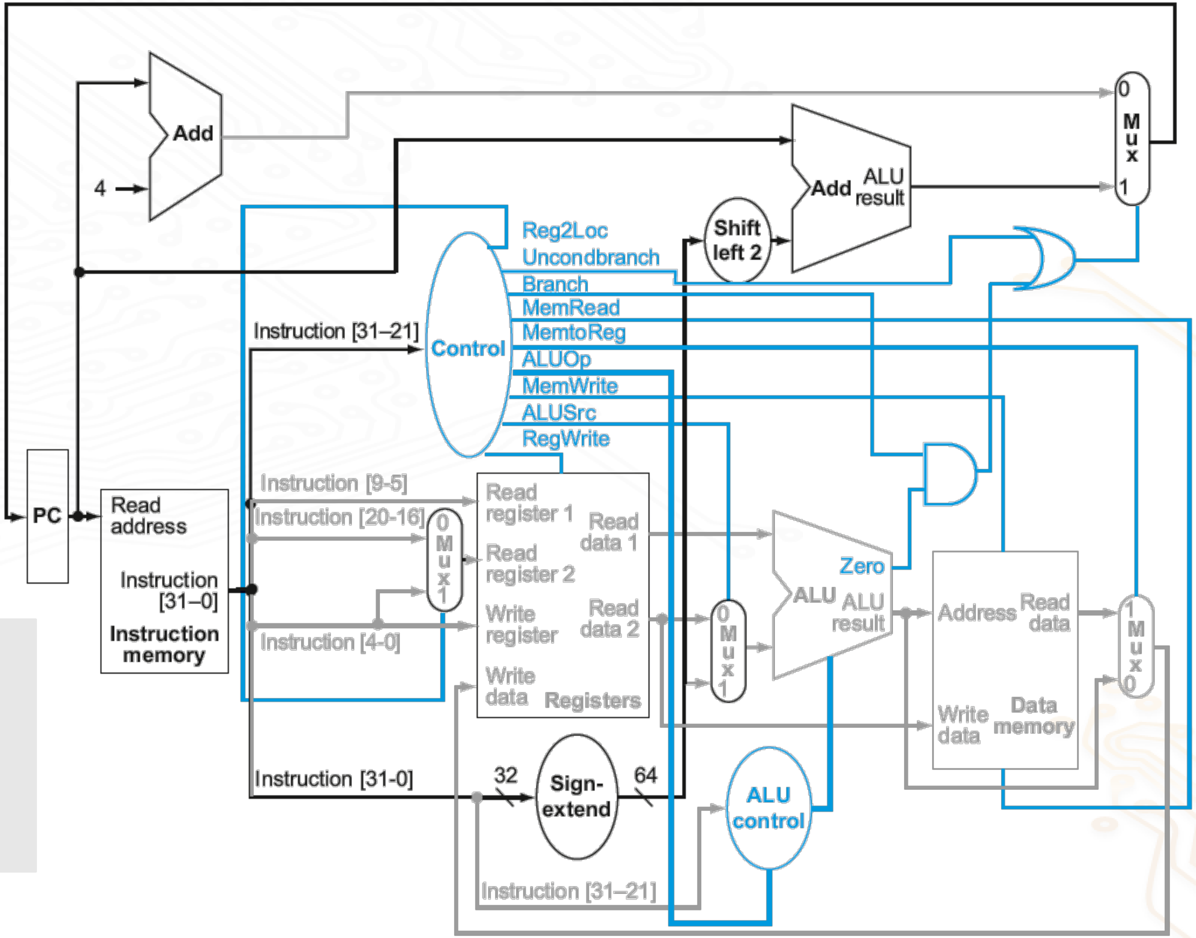
example: CBZ Rt, #address

Implementing Un-Conditional Branch Instructions

opcode	address
6 bits	26 bits

- Branch uses word address
  - 26-bit unconditional Branch address
  - Left shift by 2 (or add “00” to LSB)
  - Add with PC
- Need an extra control signal decoded from opcode

Complete Datapath including B Instructions



B

(B-type)

datapath

Summary of creating an architecture

- Combine all into a single architecture
  - Using mux with control signals
- Five steps finishes in a single clock cycle
  - Step 1: Instruction Fetch
  - Step 2: Instruction Decode and Register Fetch
  - Step 3: Execution, Memory Address Computation, or Branch Completion
  - Step 4: Memory Access or R-type instruction completion
  - Step 5: Write-back step

- No functional unit can be used twice in one clock cycle
- The clock period is decided by the slowest instruction

Performance issue of single cycle architecture

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- **improve performance by pipelining**