The Processor: Datapath

CSCM601252 – Pengantar Organisasi Komputer

Instructor: Erdefi Rakun + Tim Dosen POK

Fasilkom UI

Outline

- Datapath Elements
  - Classification
  - Instruction Memory
  - Adder
  - Register File
  - ALU
  - Data Memory
  - Multiplexer

- Building Datapath
  - Fetch
  - R-Format
  - Load/Store
  - ALU + Memory
  - Branch

Note: These slides are taken from Aaron Tan's slide

So You Want To Build A Computer

- A processor executes instructions
- Datapath
  - Performs the arithmetic, logical and memory operations
- Control
  - Tells the datapath, memory, and I/O devices what to do according to the wishes of the instructions of the program
- Best learnt through an example

Ack: Slides 3 – 31 are taken from Dr Tulika Mitra's CS1104 notes.

Datapath + Control

Datapath can perform many operations.
Control tells datapath which operation to perform.

MIPS Implementation

- Simplest possible implementation of a subset of the core MIPS ISA
  - Arithmetic and logical operations (**add**, **sub**, **and**, **ori**, **slt**)
  - Data transfer instructions (**lw**, **sw**)
  - Branches (**beq**, **bne**)
- We will not consider J-type instructions (jump) here; however it requires simple modification of the datapath + control
- Similarly, handling of shift instructions will be left as an exercise

Execution Steps (1/2)

- Program counter (PC) contains the address of the current instruction
- Instruction Fetch: Send the PC to the memory that contains the code and fetch the instruction
- Register Read: Read zero, one or two registers depending on the instruction

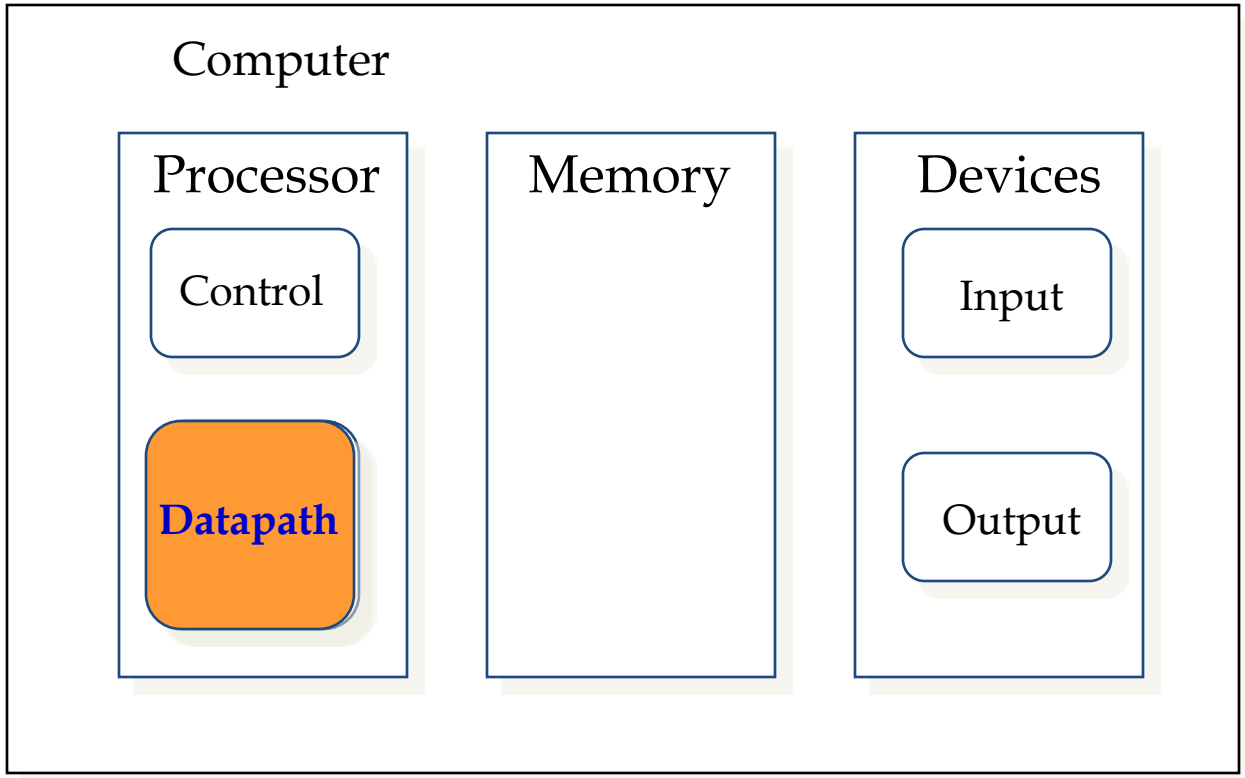
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		

Identify number of register reads for add, sub, and, or, slt, lw, sw, beq

Execution Steps (2/2)

- Computation
  - Arithmetic/logical instructions use ALU (Arithmetic-Logical unit) for the operation
  - Load/store instructions use ALU to calculate the address from base register and offset
  - Branch instructions use ALU for comparison
- Little similarity among instruction classes
  - Register Write: Arithmetic/logical instructions write back the result to destination/target register
  - Memory read/write: Load/store instructions read data from memory or write data to memory
  - Modify PC: Branch instructions may change the PC; otherwise PC is incremented by 4 to get the address of the next instruction

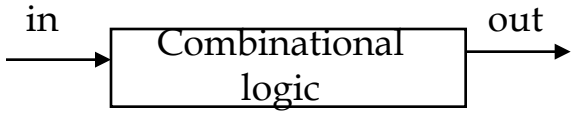
Today's Focus



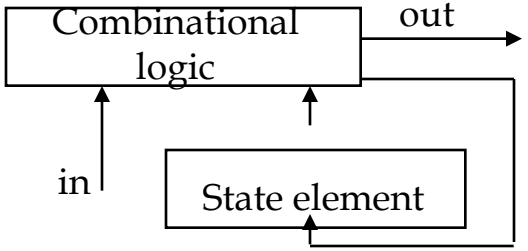
Datapath: Definition

- Simple: "The components of the processor that performs arithmetic operations"
- More comprehensive: "The collection of different elements that together provide a conduit for the flow and transformation of data in the processor during execution"

Datapath Elements: Classification

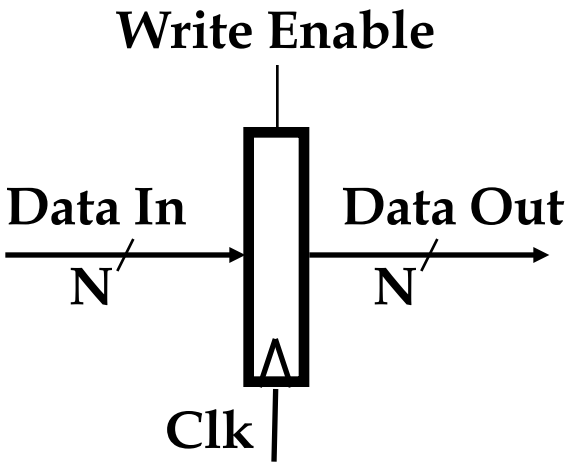


- Computation elements
  - Elements that operate on data values
  - Combinational logic: output depends only on the current inputs



- Storage elements
  - Elements that have some internal storage, i.e., they contain state
  - Also called sequential logic because their output depends on both the inputs and the internal state
  - Example: registers, instruction and data memories

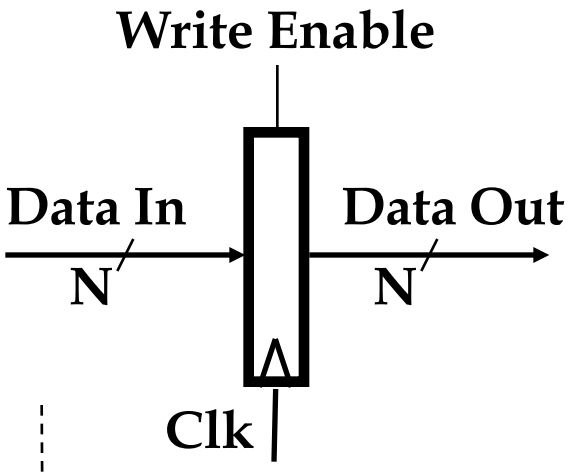
Register: A Basic Storage Element

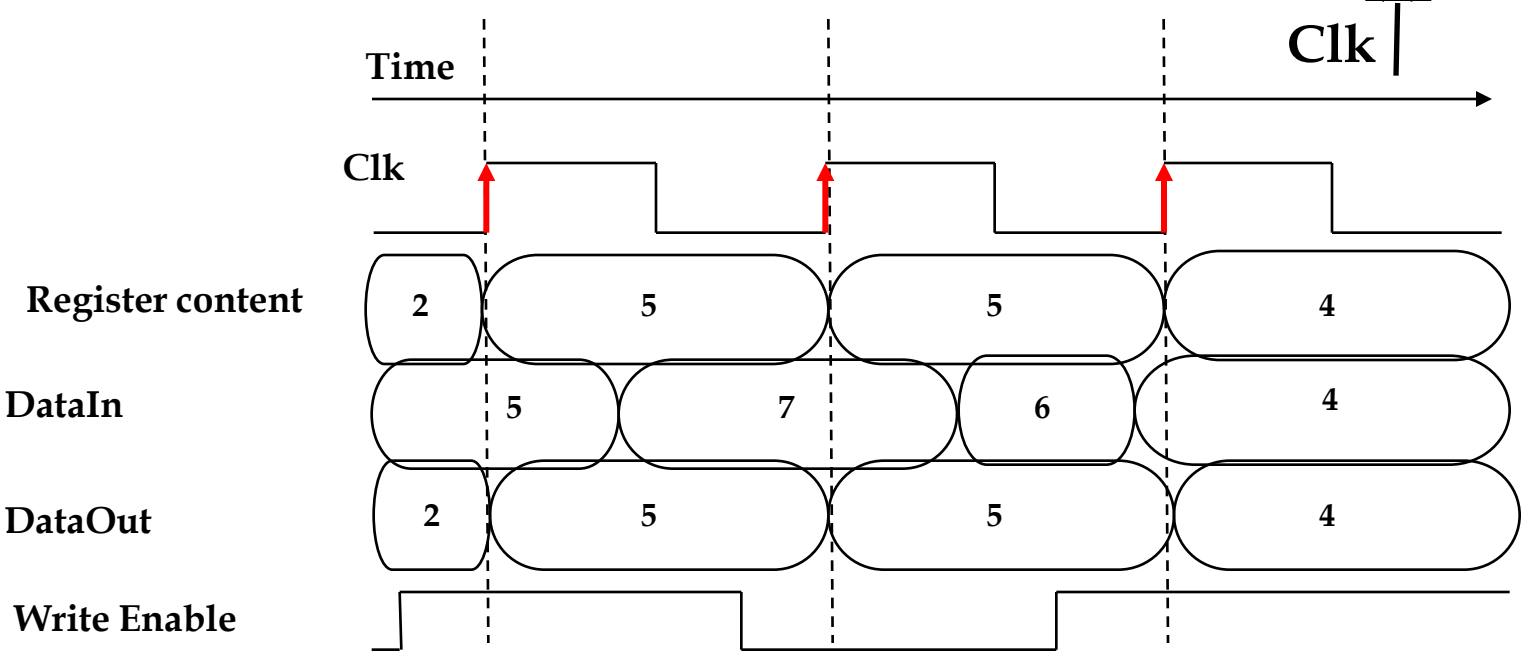


- State/storage: N-bit data
- Output
  - Data Out: value stored inside the register
  - The value can be read any time
- Input
  - Data In: data value to be written
  - Clk: Clock determines when the data will be written
- Control
  - Write Enable:
    - negated (0): Data In will not be written
    - asserted (1): Data In will be written

Register Operations

1. DataOut = Register content at all times
2. Register content changes to DataIn at rising clock edge
3. Register content changes at rising clock edge only if Write Enable=1 (see DataIn = 7)
4. The change in DataIn value within a clock period has no effect on the register content (see DataIn = 6)





Lecture Roadmap

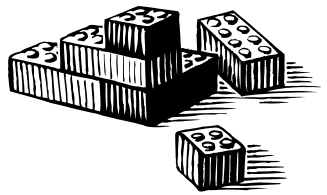
- Identify datapath elements for each execution step starting with instruction fetch
- Combine datapath elements to build datapath for an instruction type
- Combine datapath for different instruction types to build the complete MIPS datapath

"Now, this is just a simulation of what the blocks will look like once they're assembled."

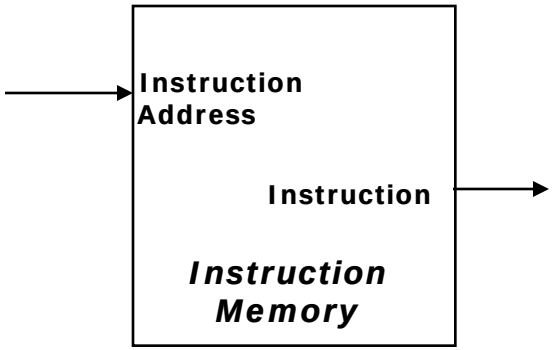
Building Datapath: FETCH

- Instruction Fetch:
  - Send the PC (implemented as register) to the memory that contains the code and fetch the instruction
  - Increment the PC by 4 to get the address of the next instruction (exception: branch) – Why 4?

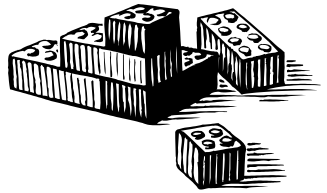
Datapath Elements: Instruction Memory



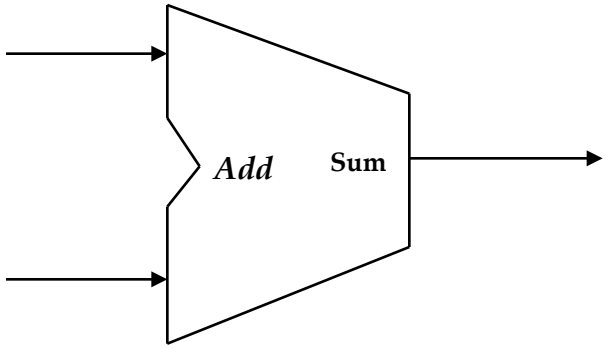
- Storage element for the instructions
- Supply instructions given the address
  - Given instruction address M as input, the memory outputs the content at address M
- Think of it logically as an array of registers
- Actual implementation is different – more about this later …
- Clock will not be shown explicitly for storage elements; however, it is assumed to be there



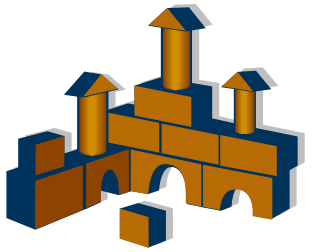
Datapath Elements: Adder



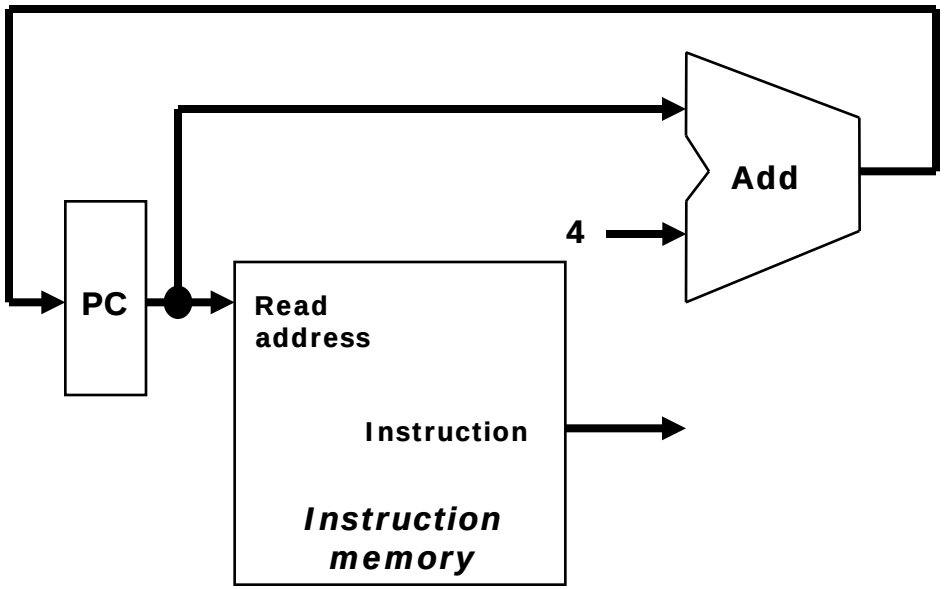
- Combinational logic to implement addition
  - Not an ALU
- Inputs: Two 32-bit numbers
- Output: sum of the input numbers
- You know how to implement this!



Building Datapath: FETCH



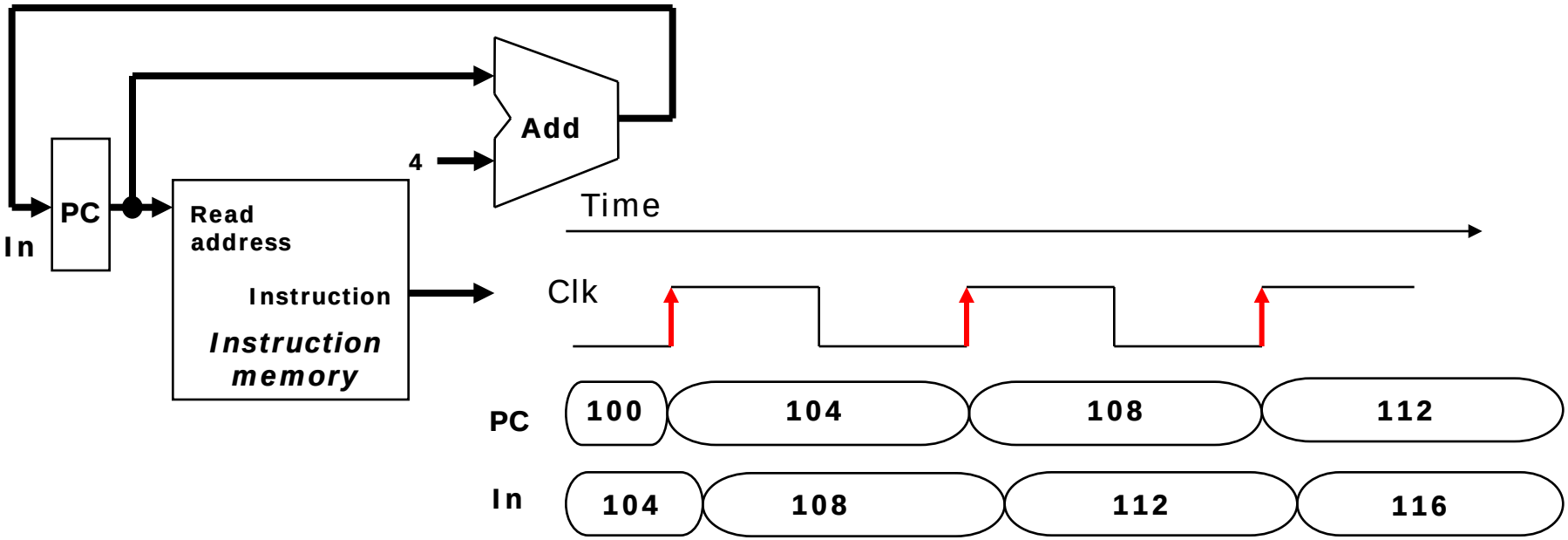
- Instruction Fetch:
  - Send the PC (implemented as register) to the memory that contains the code and fetch the instruction
  - Increment the PC by 4 to get the address of the next instruction (exception: branch)



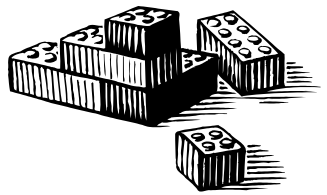
When do you think PC is updated?

Does This Really Work?

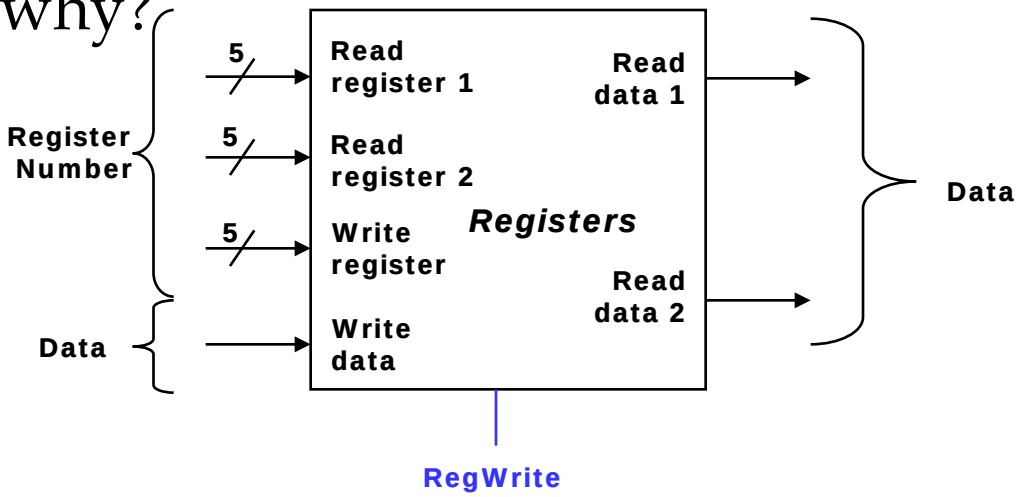
- How can you read and update the PC?
- Magic of clock: PC is read during the first half of the clock period and it is updated with PC+4 at the next rising clock edge



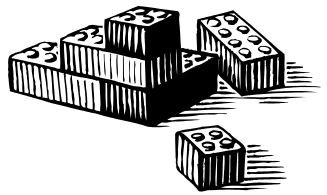
Datapath Elements: Register File



- A collection of 32 registers (each 32-bit wide) where any register can be read/written by specifying register number
- Read at most two data words per instruction
- Write at most one data word per instruction
- RegWrite must be asserted (=1) for the data to be written at rising clock edge – why?

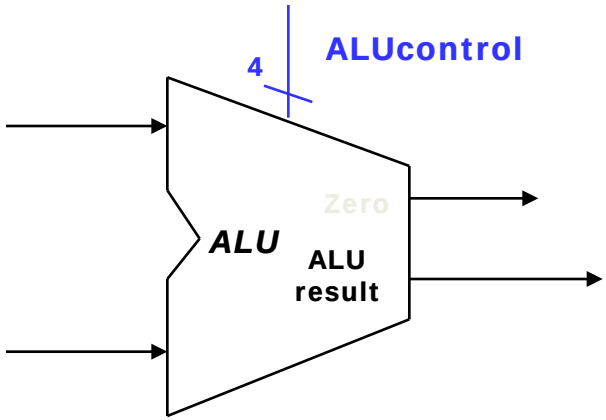


Datapath Elements: ALU



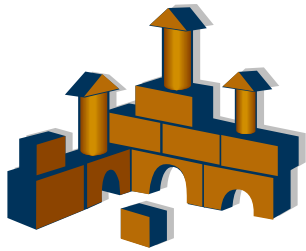
- ALU (Arithmetic-logical unit)
  - Combinational logic to implement arithmetic and logical operations
- Inputs: Two 32-bit numbers
- Outputs: result of arithmetic/logical operation

  Zero detection for result – why?
- Control: 4-bit to decide the particular operation

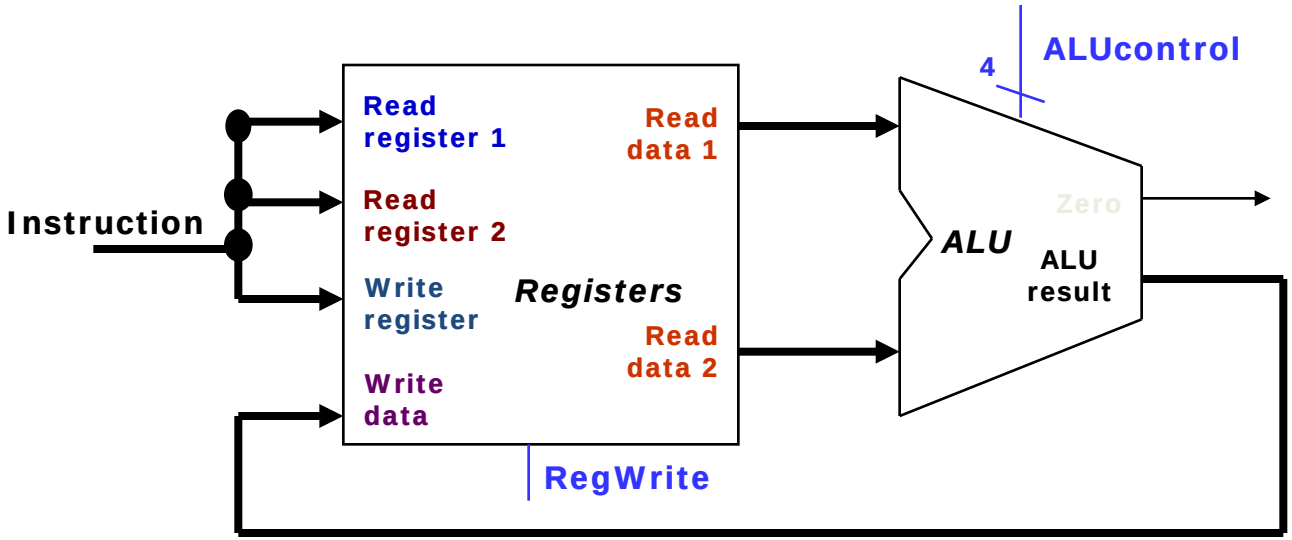


ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

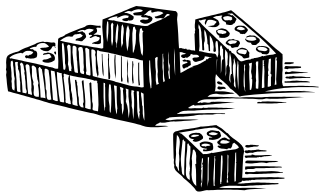
Building Datapath: R-Format



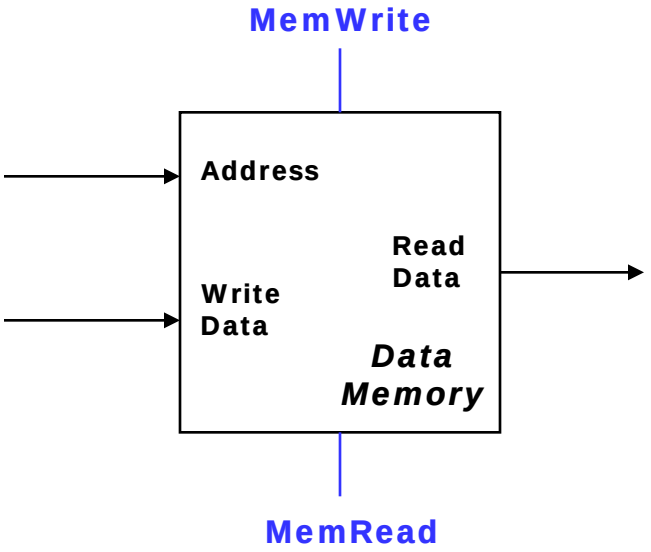
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
opcode	rs	rt	rd	shamt	funct



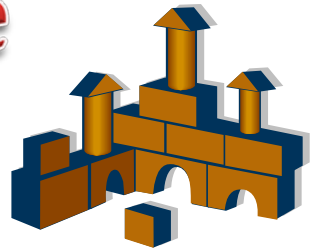
Datapath Elements: Data Memory



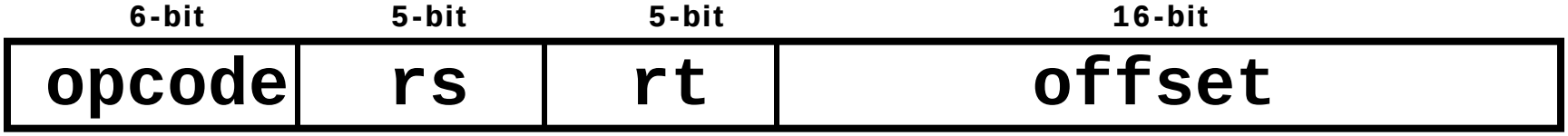
- Storage element for the data of a program
- Inputs: Address and write data
- Output: Read Data
- Control: Read and write controls; only one can be asserted at any point of time

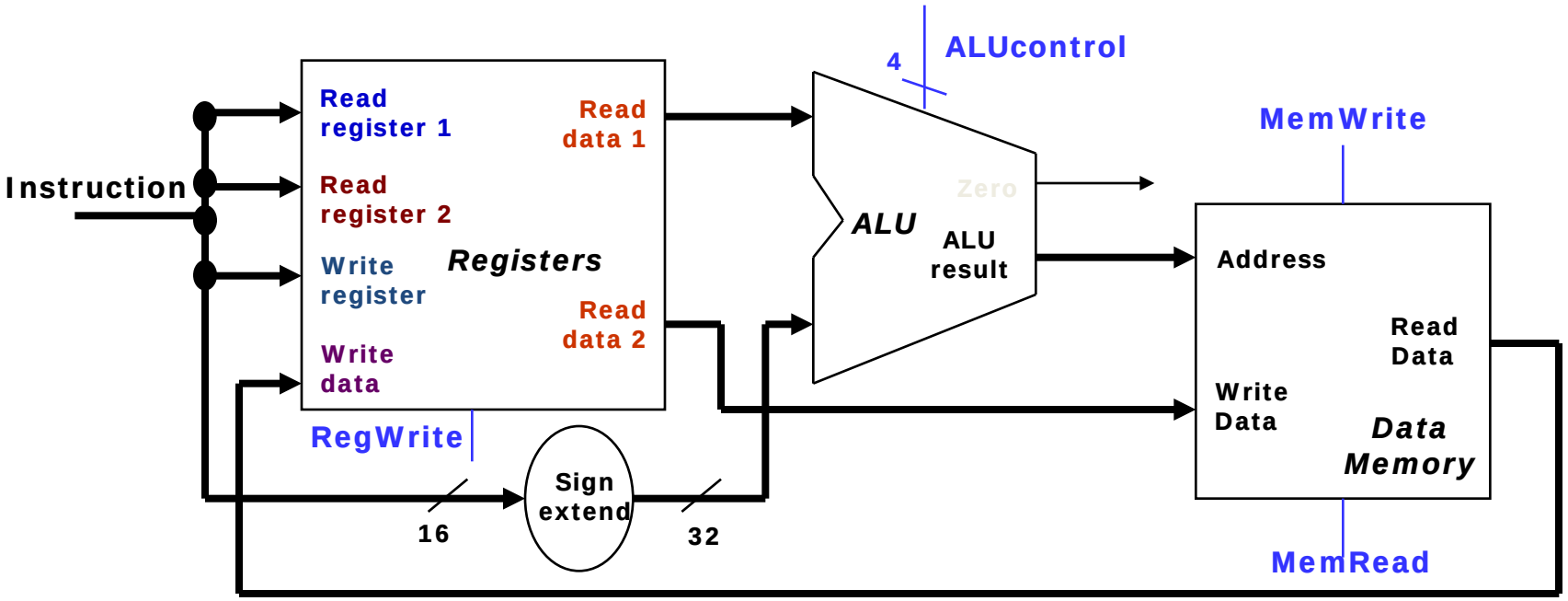


Building Datapath: Load/Store

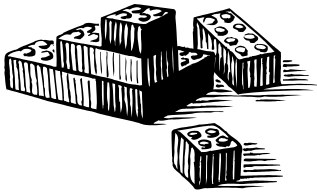


- Address calculation: add the content of the base register to 16-bit signed offset using ALU
- Requires sign-extend unit to sign extend 16-bit offset field to 32-bit signed value – why?

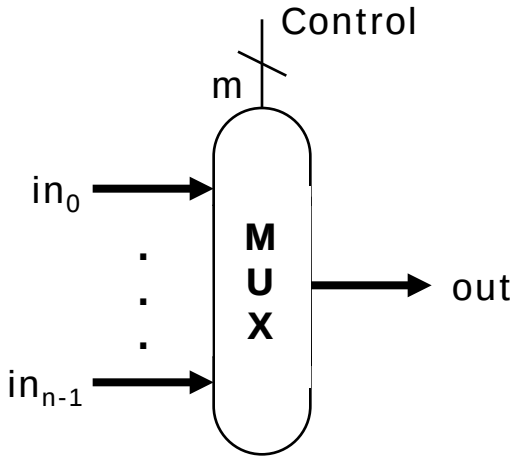




Last Piece: Multiplexer



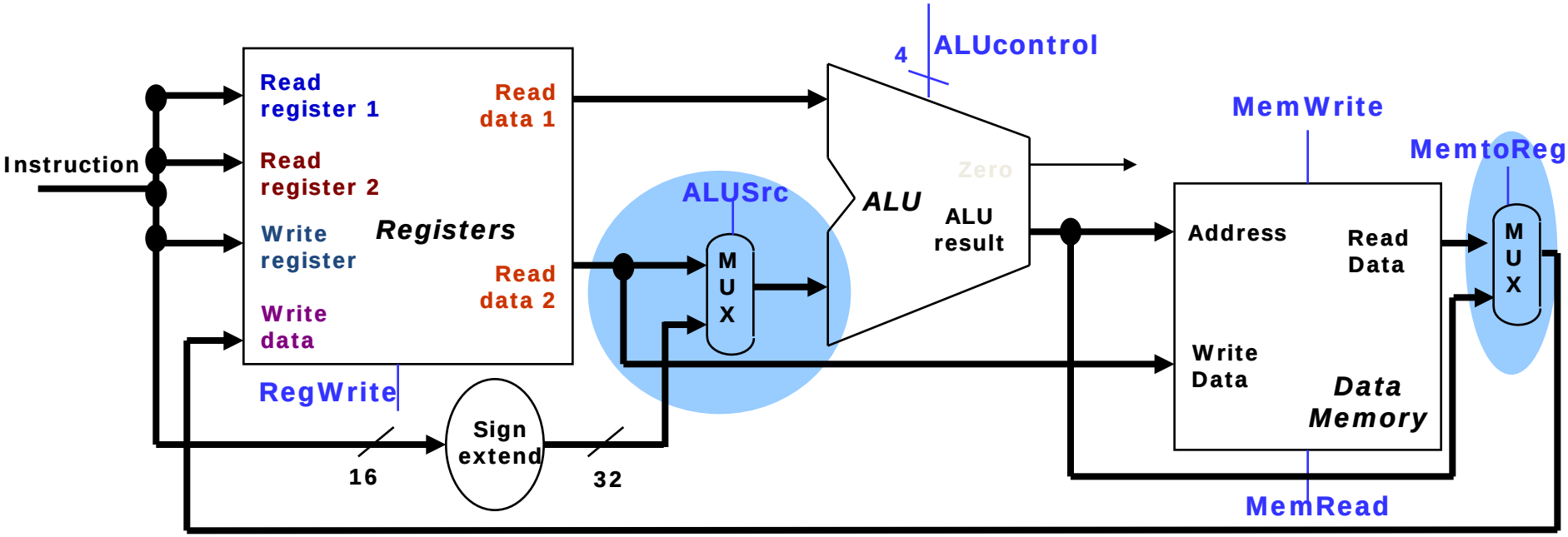
- Selects one input from multiple input lines
- Inputs: n lines (same width each)
- Control: m bits where $n = 2^m$
- Output: select i^{th} input line if control=i



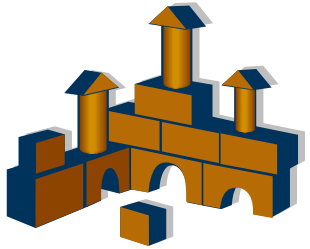
Control=0 → select in_0
Control=3 → select in_3

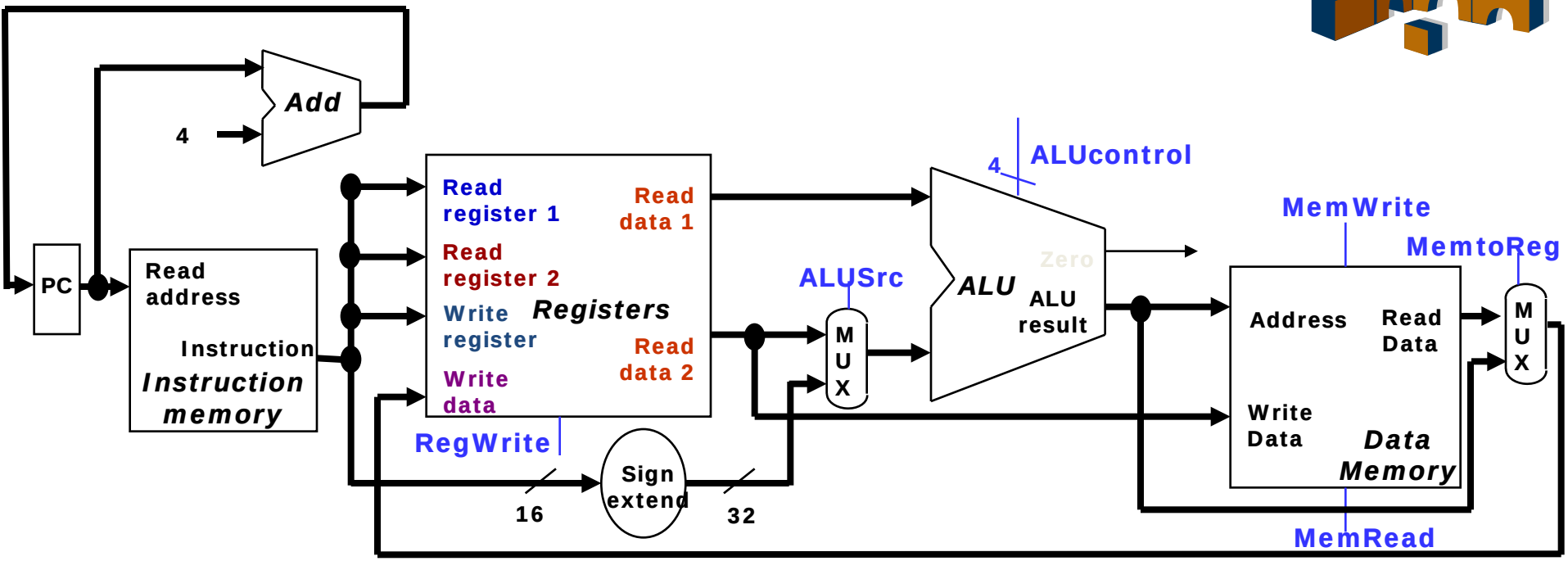
Building Datapath: ALU + Memory

- R-type instructions use two registers as ALU operands whereas load/store uses one register (base address) and 16-bit offset as ALU operands
- Value stored into a destination register comes from ALU (for R-type instructions) or from memory (for load)



Throw in Instruction Fetch

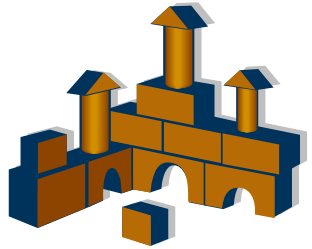


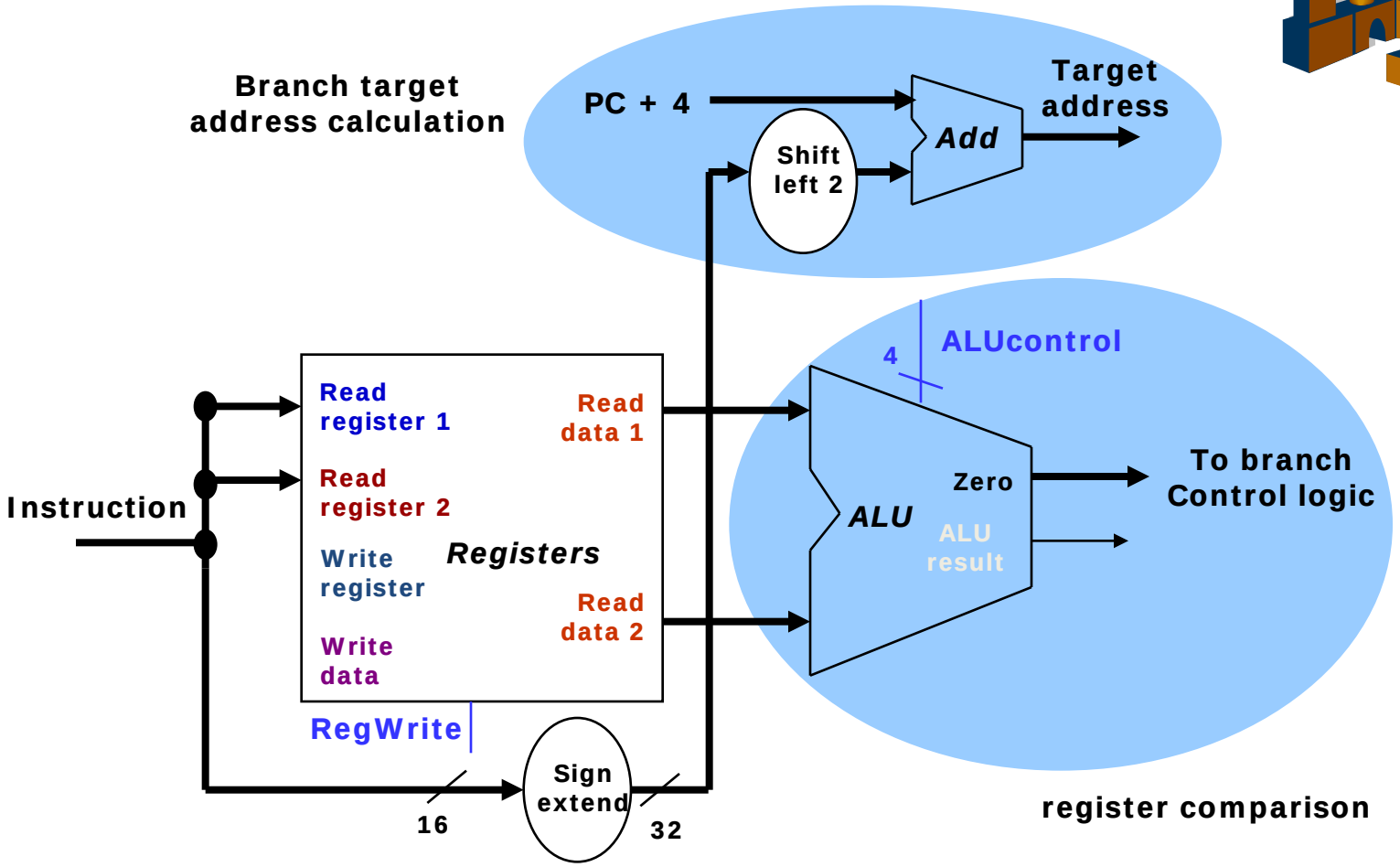


Conditional Branching

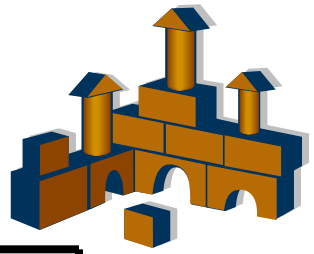
- Compare register contents (say $t0, $t1) to see if the branch will be taken
  - Achieved via subtracting $t1 from $t0 in ALU and checking the Zero output
    - Zero output =1 ➔ $t0 = $t1
- Branch target address: add sign-extended offset field (shifted left by 2 bits due to word offset) to PC+4
  - Can you see why we use PC+4 now?
  - Observation: Need extra adder to compute branch target address as ALU is doing the comparison

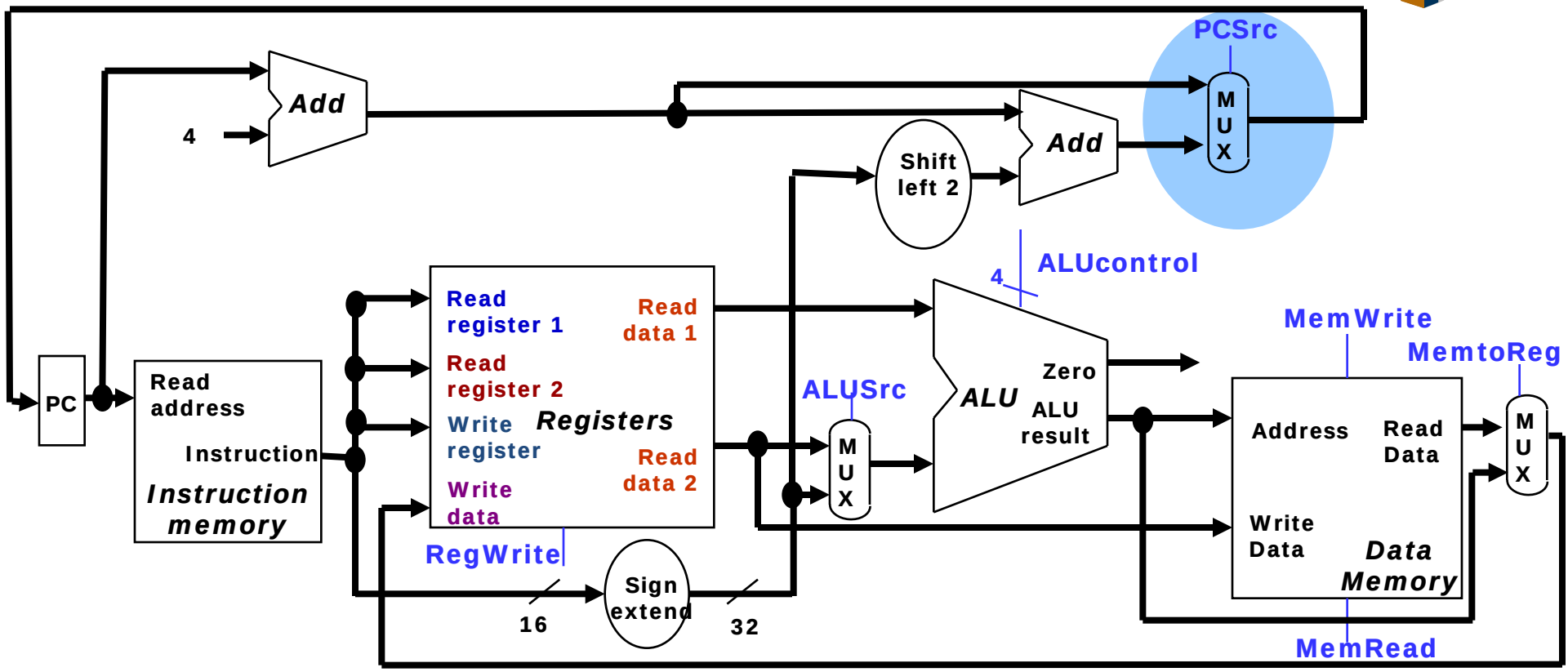
Building Datapath: Branch





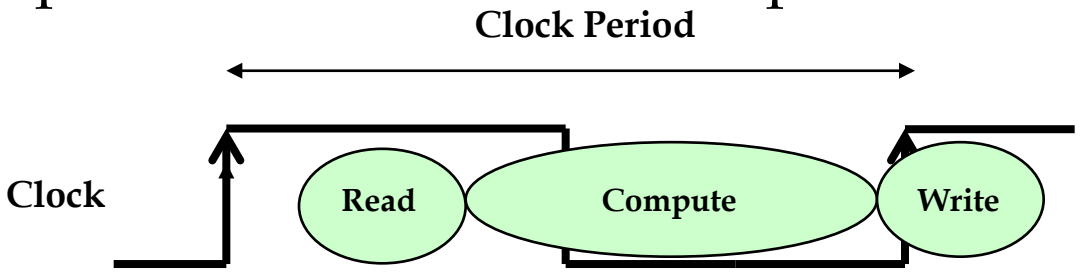
Complete Datapath FINALLY!





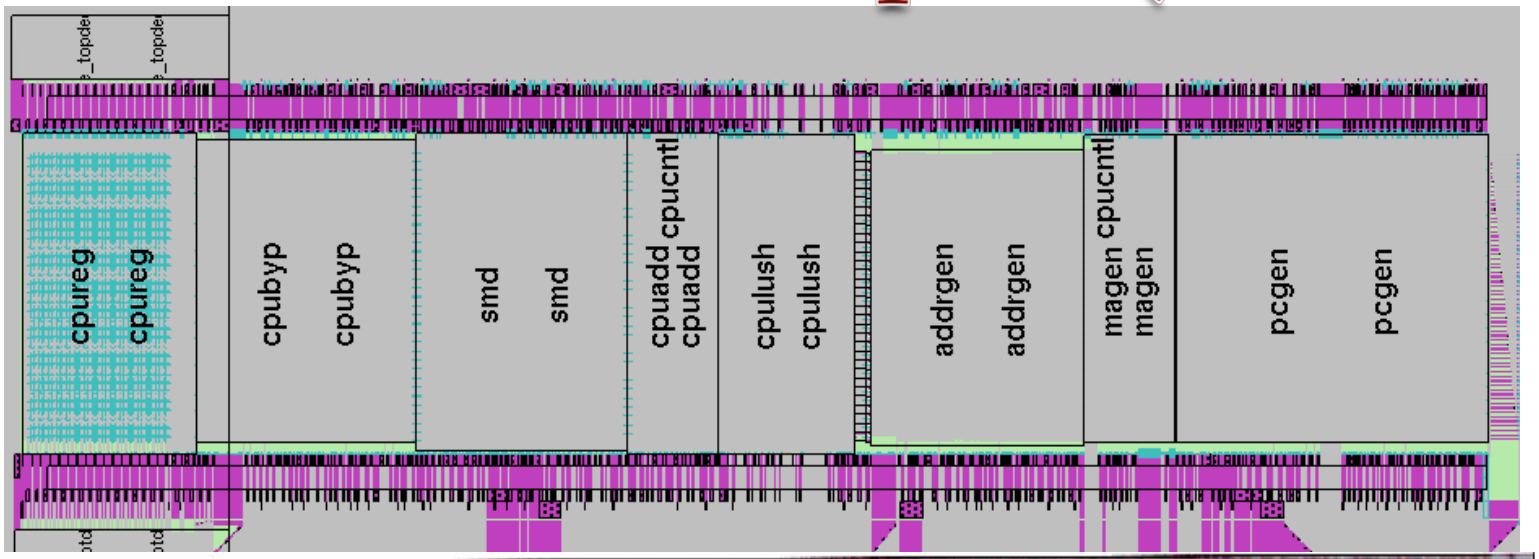
Big Picture: Instruction Execution

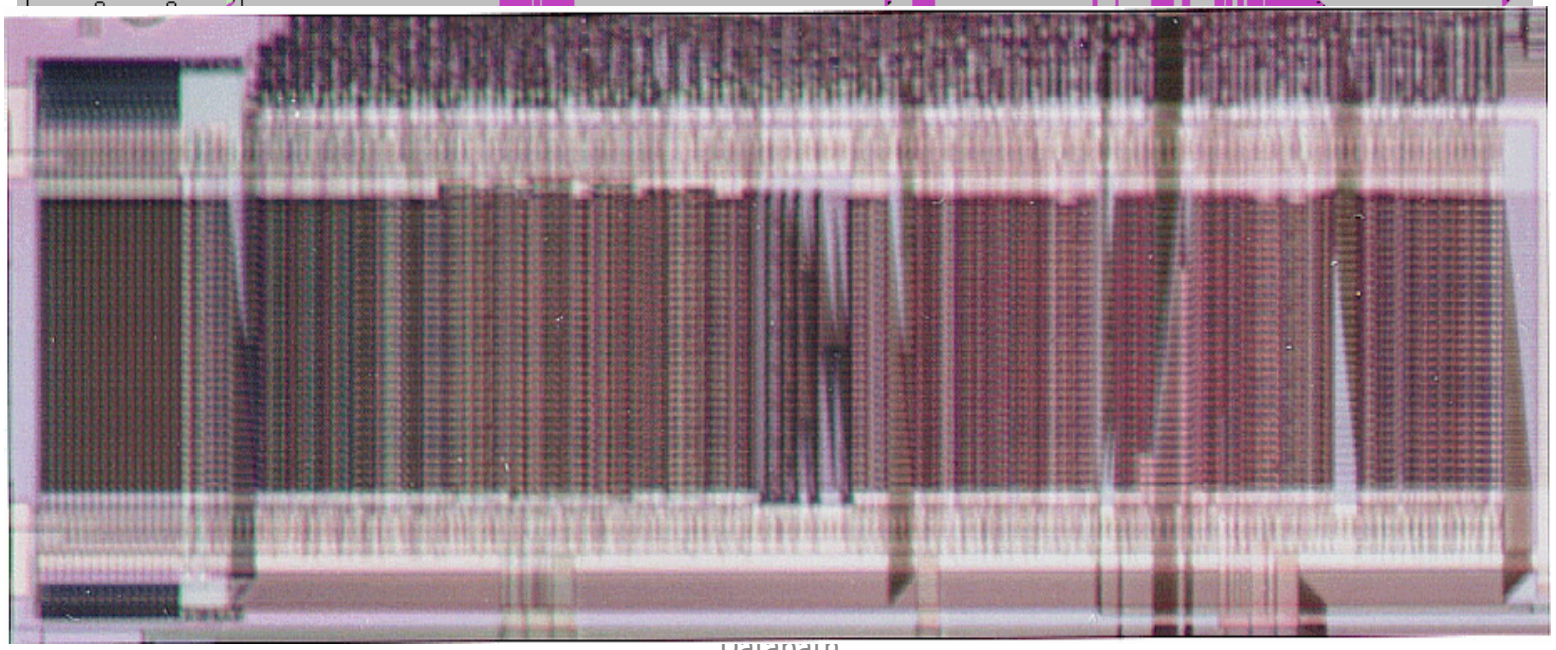
- Instruction execution is equivalent to
  - Read contents of one or more storage elements (register/memory)
  - Perform computation through some combinational logic
  - Write results to one or more storage elements (register/memory)
- All these performed within a clock period



Don't want to read a storage element when it is being written

A Real MIPS Datapath (CNS T0)





Stages Of A Datapath (1/7)

- Problem: a single, atomic block which "executes an instruction" (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient.
- Solution: break up the process of "executing an instruction" into stages, and then connect the stages to create the whole datapath.
  - Smaller stages are easier to design.
  - Easy to optimize (change) one stage without touching the others.

Stages Of A Datapath (2/7)

- There is a *wide* variety of MIPS instructions: so what general steps do they have in common?
- Stages
  1. Instruction Fetch
  2. Instruction Decode
  3. ALU
  4. Memory Access
  5. Register Write

Stages Of A Datapath (3/7)

- Stage 1: **Instruction Fetch**.
  - No matter what the instruction is, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy).
  - Also, this is where we **increment PC** (that is, PC = PC + 4, to point to the next instruction; byte addressing so + 4).

Stages Of A Datapath (4/7)

- Stage 2: **Instruction Decode**
  - Upon fetching the instruction, we next gather data from the fields (*decode* all necessary instruction data).
  - First, read the **opcode** to determine instruction type and field lengths.
  - Second, read in data from all necessary registers.
    - For **add**, read two registers.
    - For **addi**, read one register.
    - For **jal**, no read necessary.

Stages Of A Datapath (5/7)

- Stage 3: **ALU** (Arithmetic-Logic Unit)
  - The real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (**slt**).
  - What about loads and stores?
    - **lw $t0, 40($t1)**
    - The address we are accessing in memory = the value in **$t1** plus the value 40.
    - We do this addition at this stage.

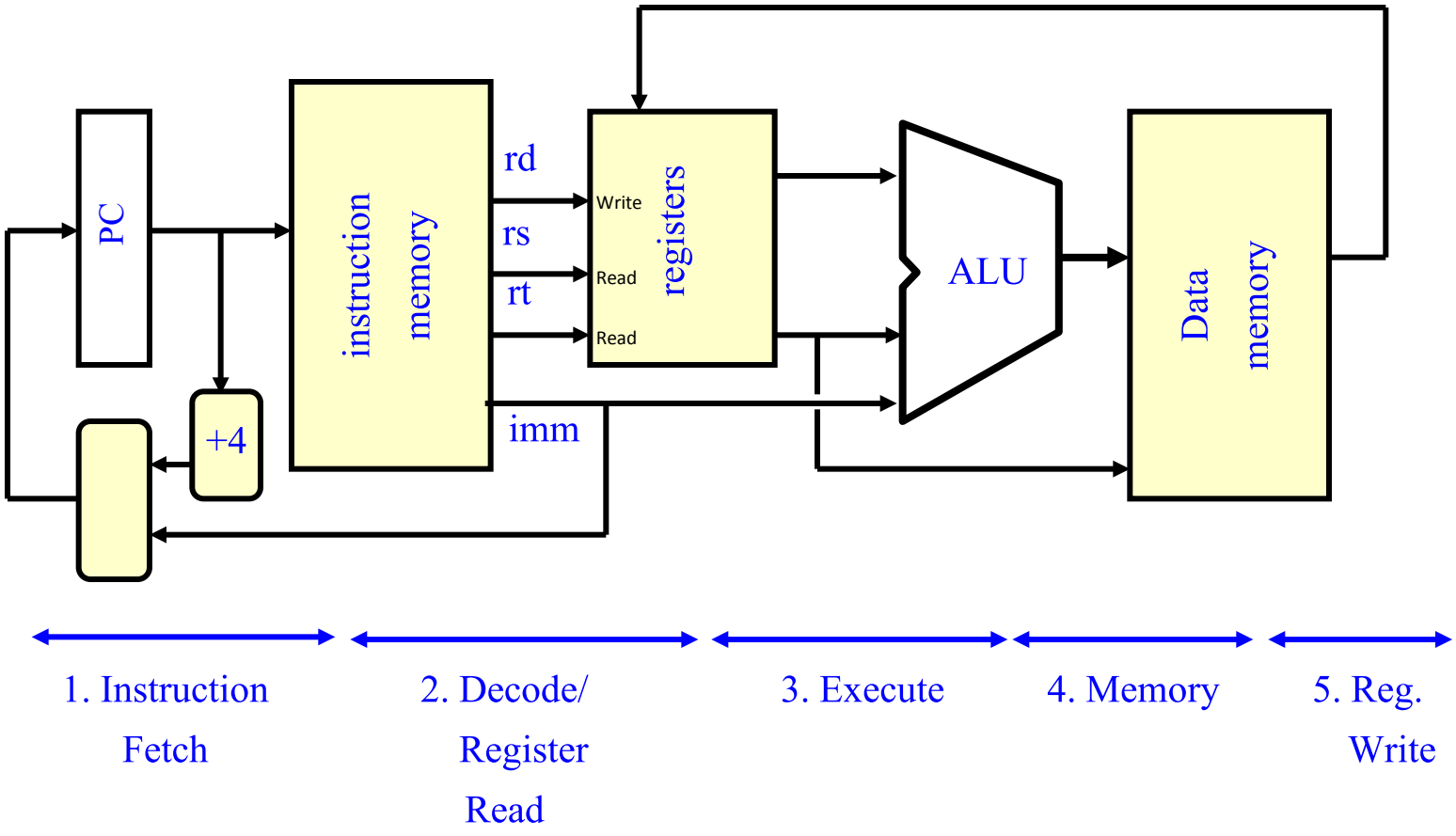
Stages Of A Datapath (6/7)

- Stage 4: **Memory Access**
  - Actually only the load and store instructions do anything during this stage; for the other instructions, they remain idle during this stage.
  - Since these instructions have a unique step, we need this extra stage to account for them.
  - As a result of the cache system, this stage is expected to be just as fast (on average) as the others.

Stages Of A Datapath (7/7)

- Stage 5: **Register Write**
  - Most instructions write the result of some computation into a register.
  - Examples: arithmetic, logical, shifts, loads, **slt**
  - What about stores, branches, jumps?
    - They do not write anything into a register at the end.
    - These remain idle during this fifth stage.

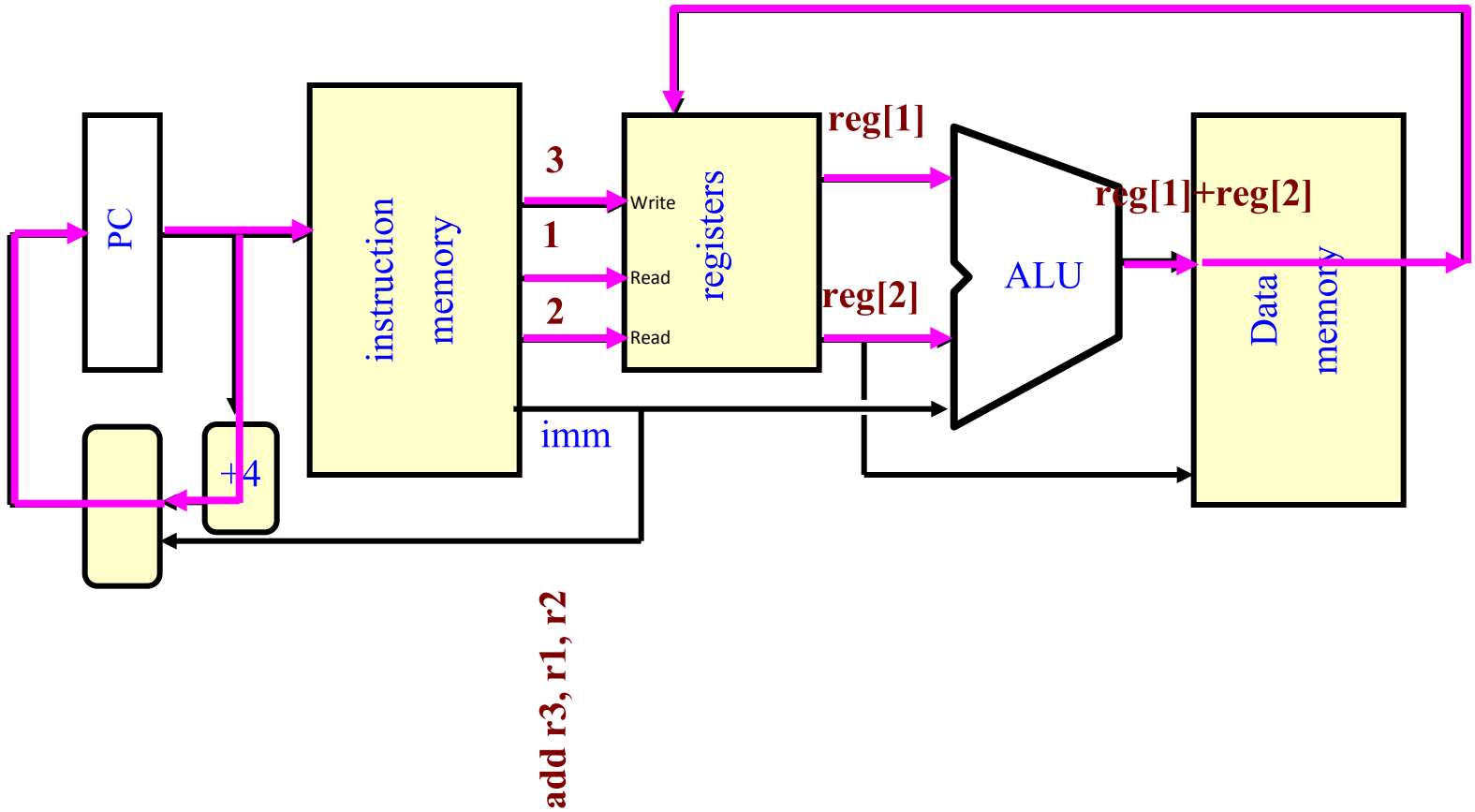
Datapath: Generic Steps



Datapath Walkthroughs: ADD (1/2)

- **add $r3,$r1,$r2 # r3 = r1+r2**
  - Stage 1: Fetch this instruction, increment PC.
  - Stage 2: Decode to find that it is an **add** instruction, then read registers **$r1** and **$r2**.
  - Stage 3: Add the two values retrieved in stage 2.
  - Stage 4: Idle (nothing to write to memory).
  - Stage 5: Write result of stage 3 into register **$r3**.

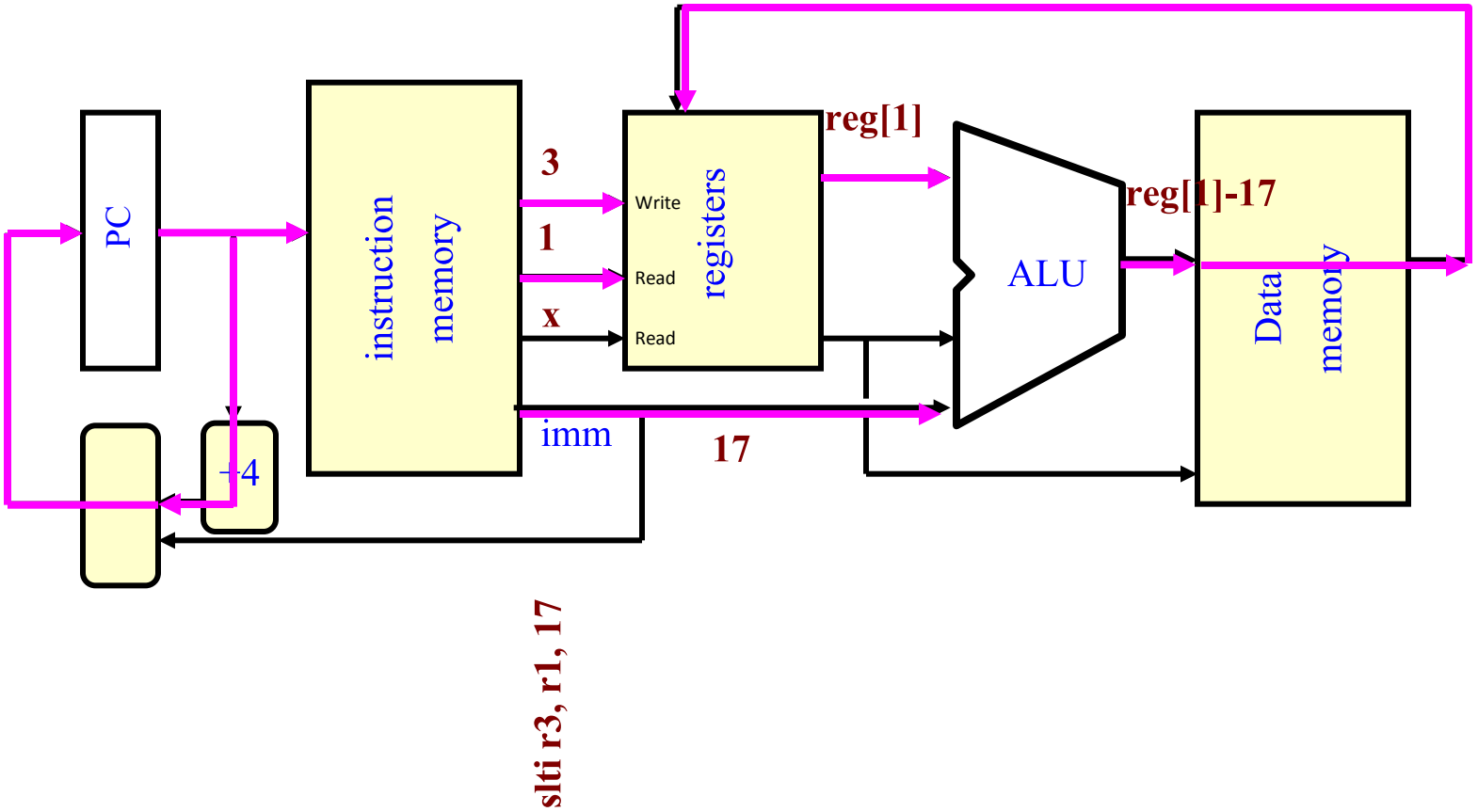
Datapath Walkthroughs: ADD (2/2)



Datapath Walkthroughs: SLTI (1/2)

- **slti $r3,$r1,17**
  - Stage 1: Fetch this instruction, increment PC.
  - Stage 2: Decode to find it is an **slti**, then read register **$r1**.
  - Stage 3: Compare value retrieved in stage 2 with the integer 17.
  - Stage 4: Go idle.
  - Stage 5: Write the result of stage 3 in register **$r3**.

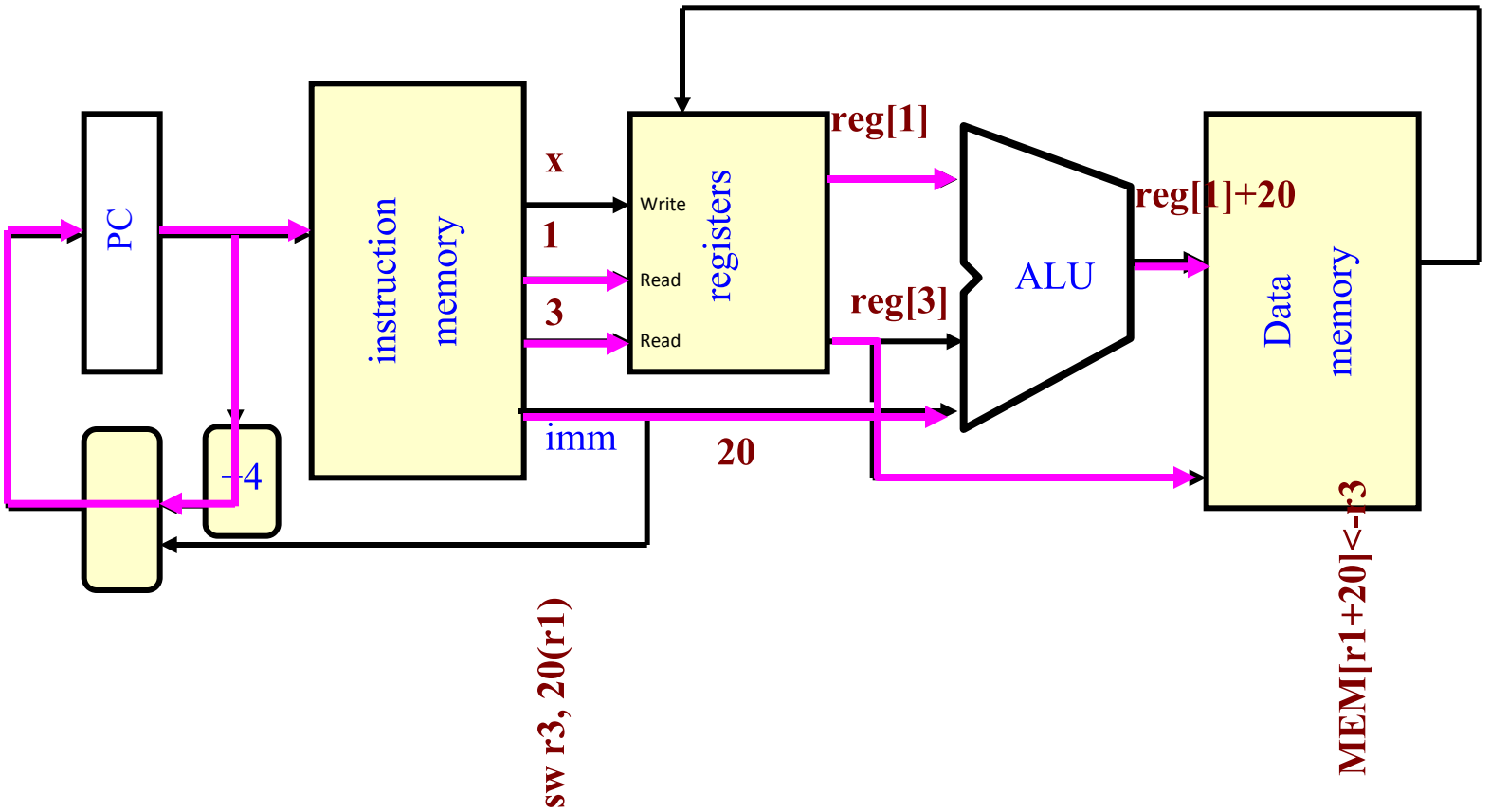
Datapath Walkthroughs: SLTI (2/2)



Datapath Walkthroughs: SW (1/2)

- **sw $r3, 20($r1)**
  - Stage 1: Fetch this instruction, increment PC.
  - Stage 2: Decode to find it is an **sw**, then read registers **$r1** and **$r3**.
  - Stage 3: Add 20 to value in register **$r1** (retrieved in stage 2).
  - Stage 4: Write value in register **$r3** (retrieved in stage 2) into memory address computed in stage 3.
  - Stage 5: Go idle (nothing to write into a register).

Datapath Walkthroughs: SW (2/2)



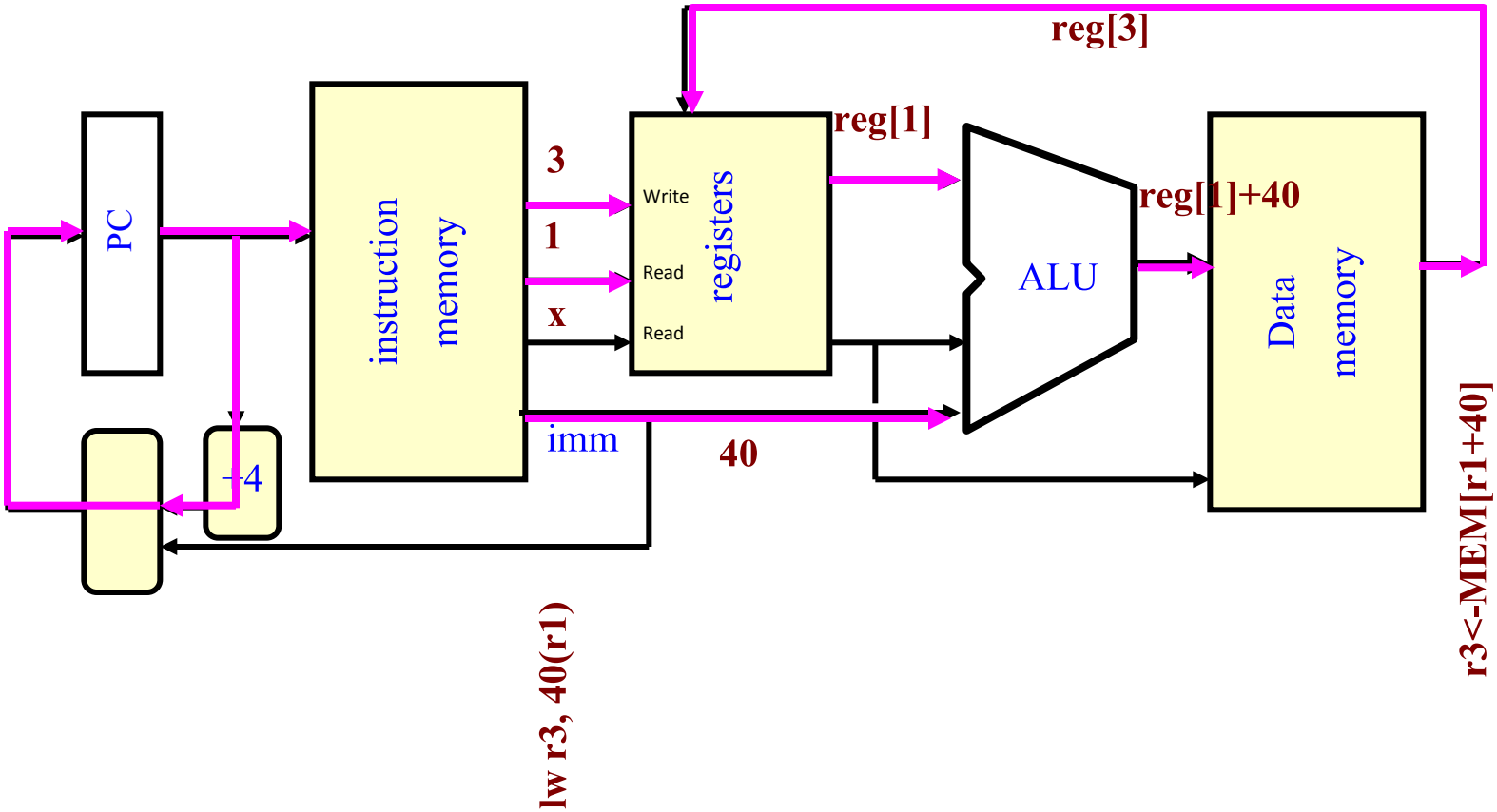
Why Five Stages?

- Could we have a different number of stages?
  - Yes, and other architectures do.
- So why does MIPS have five stages, if instructions tend to go idle for at least one stage?
  - There is one instruction that uses all five stages: the load.

Datapath Walkthroughs: LW (1/2)

- **lw $r3, 40($r1)**
  - Stage 1: Fetch this instruction, increment PC.
  - Stage 2: Decode to find it is a **lw**, then read register **$r1**.
  - Stage 3: Add 40 to value in register **$r1** (retrieved in stage 2).
  - Stage 4: Read value from memory address compute in stage 3.
  - Stage 5: Write value found in stage 4 into register **$r3**.

Datapath Walkthroughs: LW (2/2)

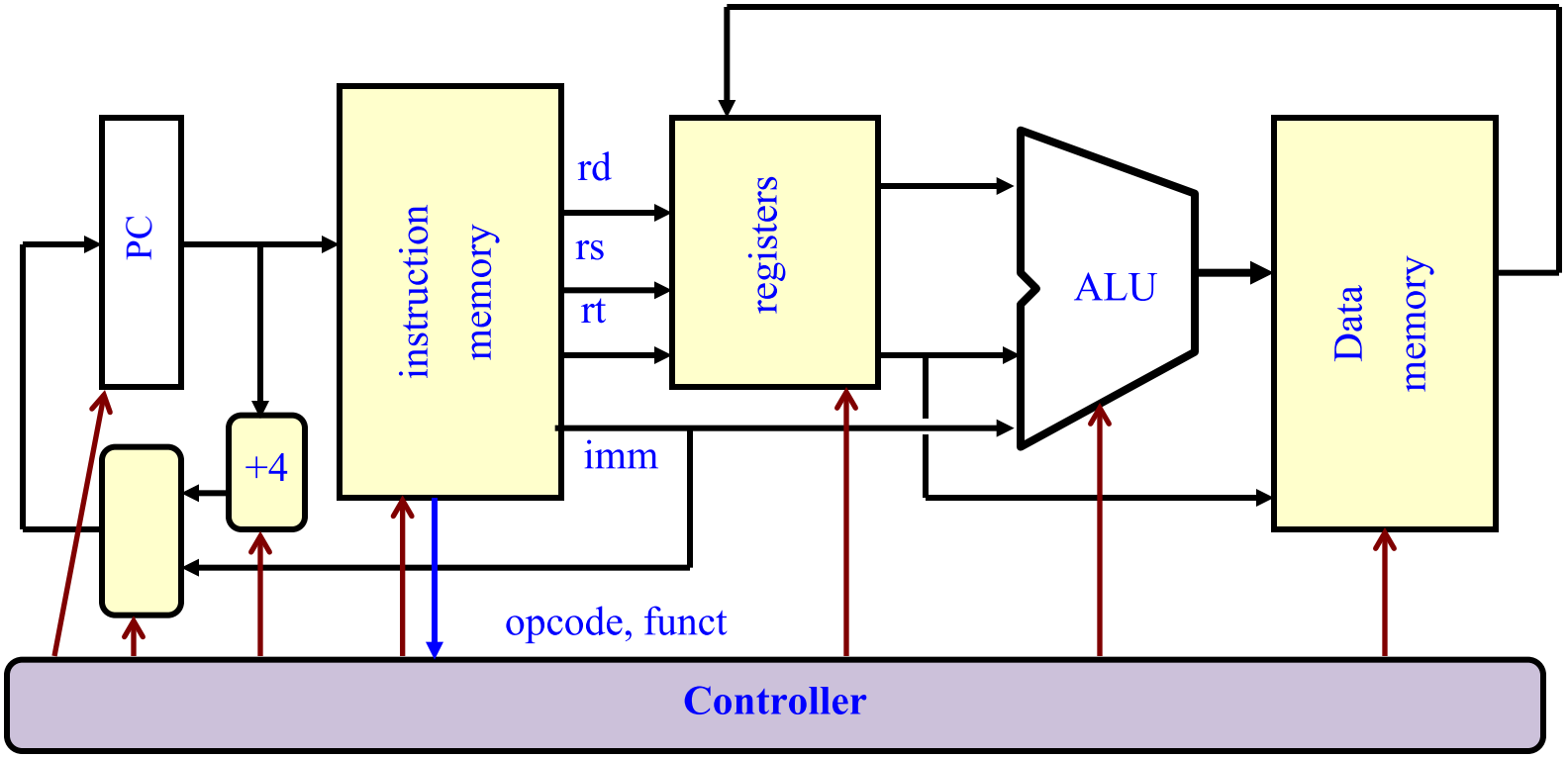


What Hardware Is Needed?

- PC: a register which keeps track of address of the next instruction.
- General Purpose Registers
  - Used in stage 2 (read) and stage 5 (write).
  - We are currently working with 32 of these.
- Memory
  - Used in stage 1 (fetch) and stage 4 (R/W).
  - Cache system makes these two stages as fast as the others, on average.

Datapath: Summary

- Construct datapath based on register transfers required to perform instructions.
- Control part causes the right transfers to happen.



READING ASSIGNMENT

- The Processor: Datapath and Control
  - 3rd edition: Chapter 5 Sections 5.1 – 5.3
  - 4th , 5th edition: Chapter 4 Sections 4.1 – 4.3