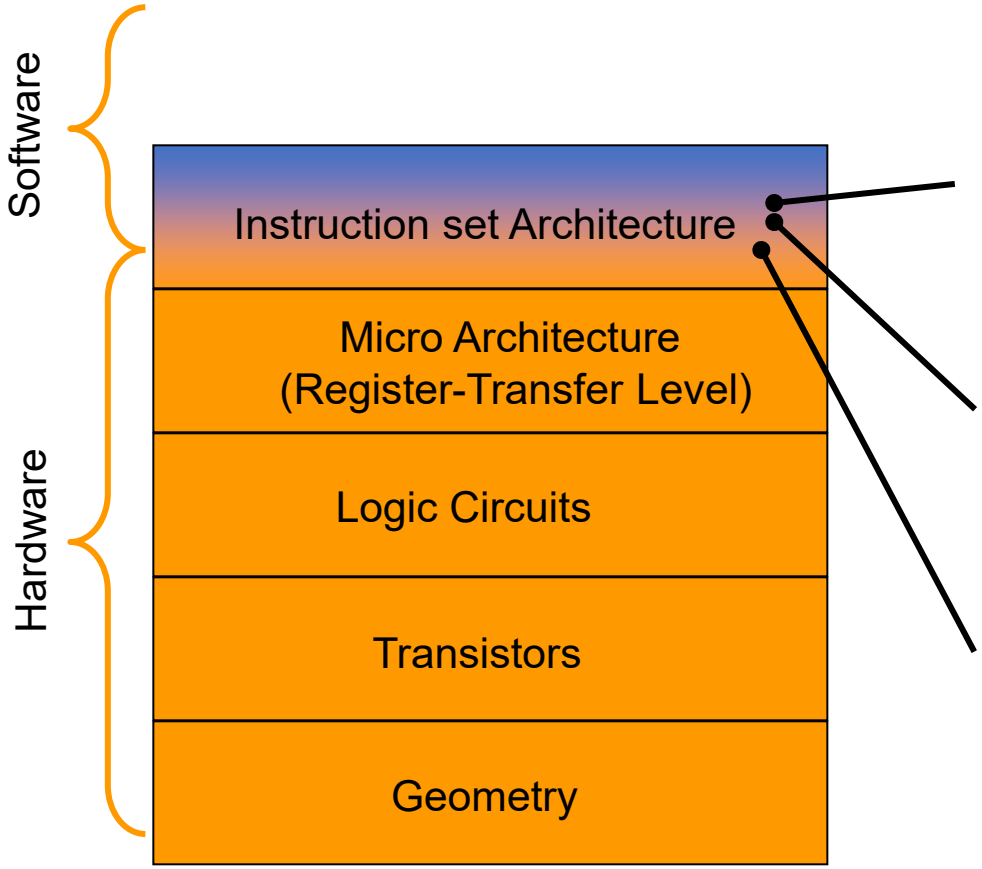
UF Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

Digital Logic And Computing Systems

Lecture 07 – Processor Design

Dr. Christophe Bobda
EEL3701C Fall 2025

Instruction Set Architecture (ISA)



HW/SW interface:

Define a set of instructions, needed to controll the hardware (CPU)

→ instruction set architecture

The number and type of instructions depend on the application field

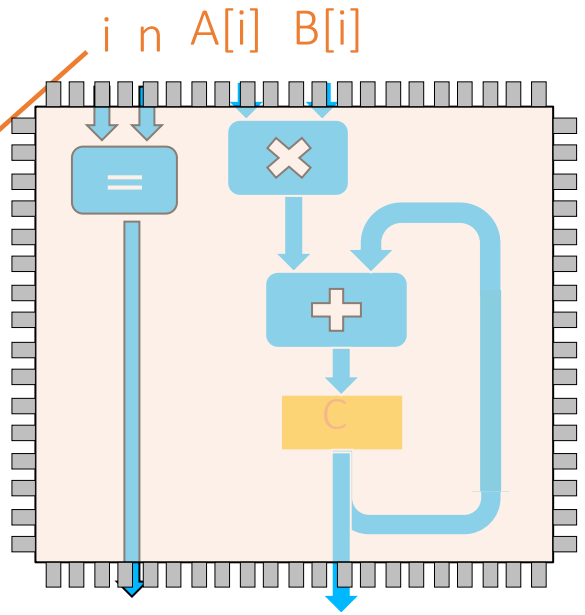
In general, instructions are designed

1. to optimize the construction and operation of the underlying hardware
2. to make the program readable and easy to understand

Computer Paradigms

- Domain Specific Computers
  - Application Specific Processor (ASIP)

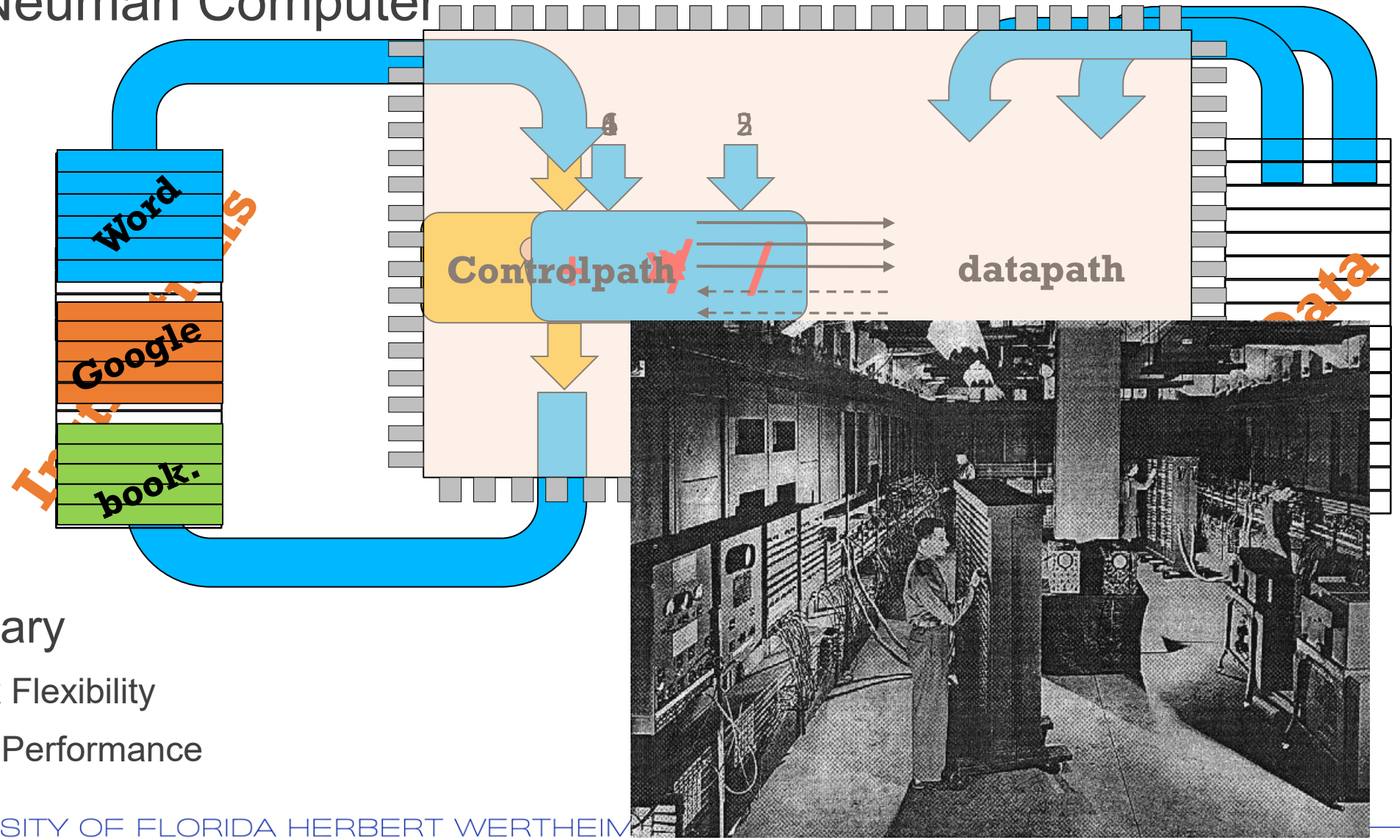
```
Array  A, B: [1:n]
Real c = A*B;


C = 0;
for i=1 to n do
    C = C + A[i]*B[i];
End for;
```



- Summary
  - Max. Performance
  - Min. Flexibility

Computer Paradigm

- Von Neuman Computer



- Summary
  - Max Flexibility
  - Min Performance

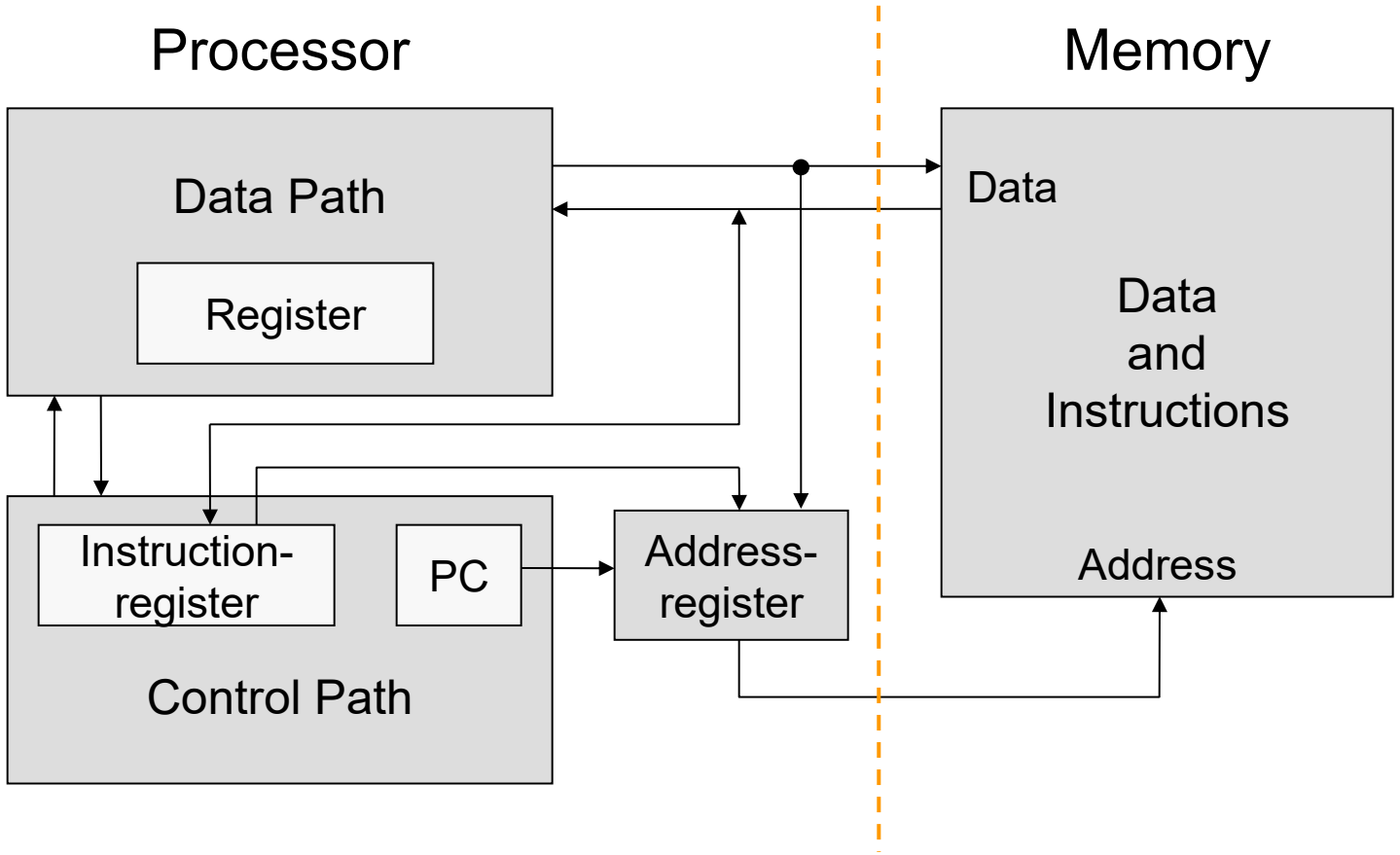
General Purpose Computing

- Examples
  - Microcontrollers
    - Motorolla 68000 familly
    - Intel MCS-51 (8056), MCS 96 (8xC196)
  - Embedded Processor (System on Chip)
    - Apple A series
    - Nvidia Tegra series
    - Qualcom Snapdragon series
    - IBM PowerPC series
  - Soft Cores
    - ARM Cortex
    - Altera Nios, Xilinx Microblazes,

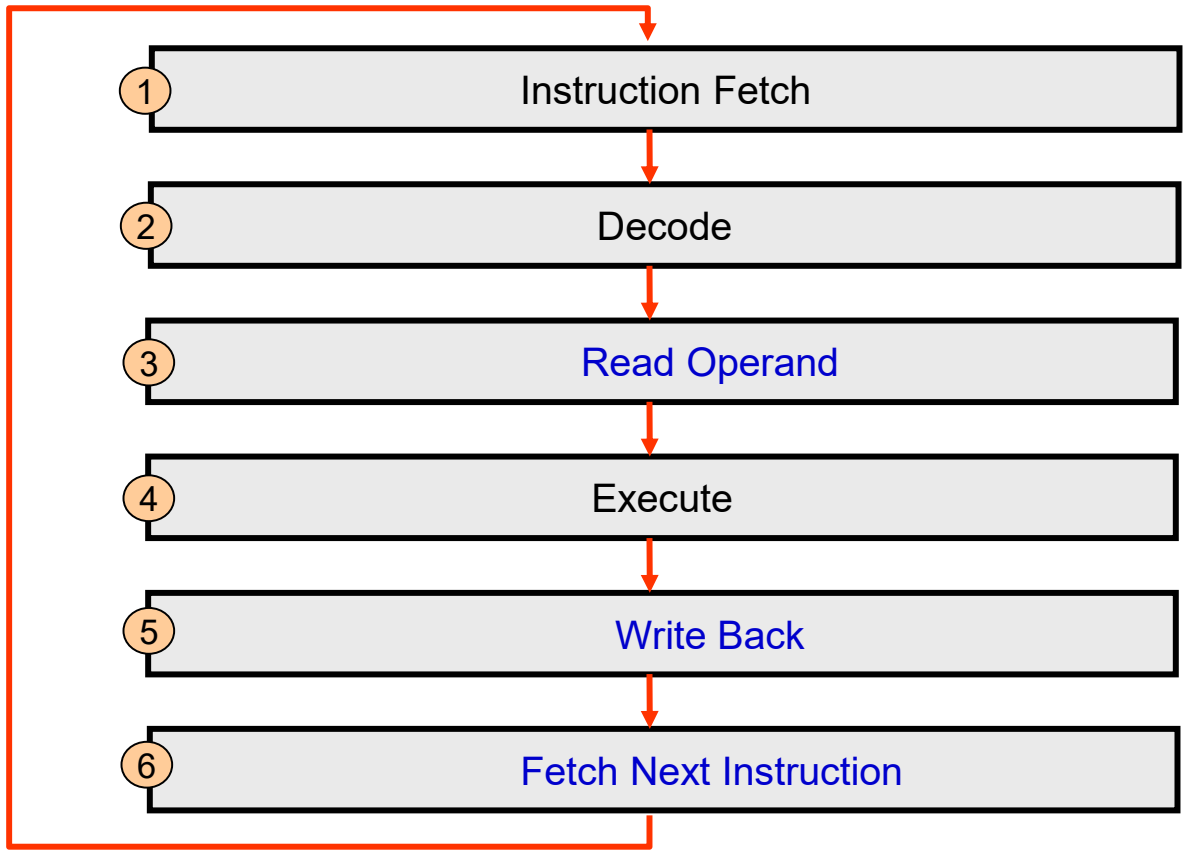
Von Neumann Computer Basic Structure

- A computer consists of
  - Processor + Memory + In/Output
- Memory consists of fixed-length words
  - Data and Instructions
- Processor consists of data path and control path
  - Data path + Control path = Central Processing Unit (CPU)
  - Program Counter (PC), store memory address of the next instruction
  - More Registers in data path
    - Operation on register much faster

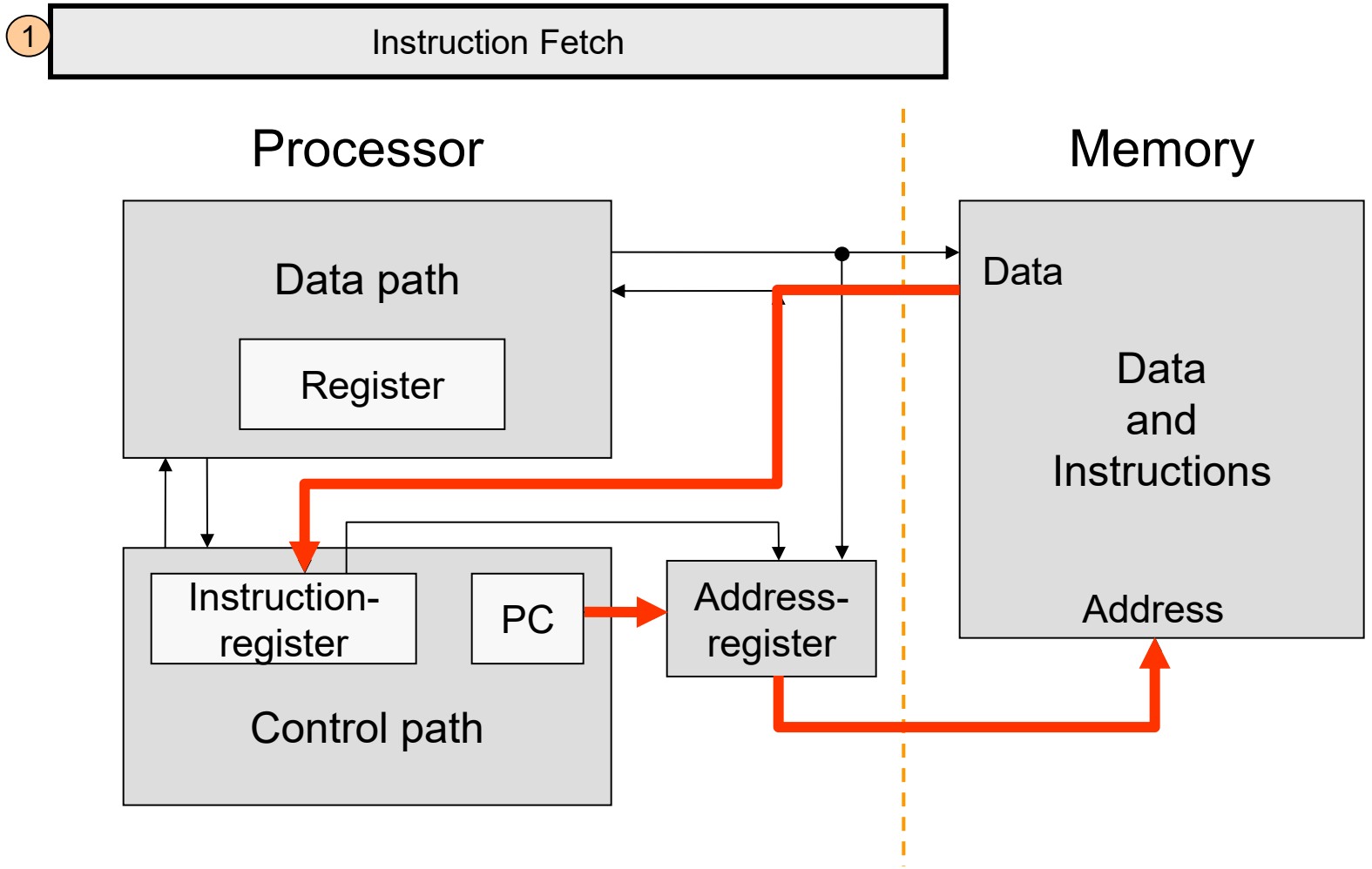
Processor/Central Processing Unit (CPU)



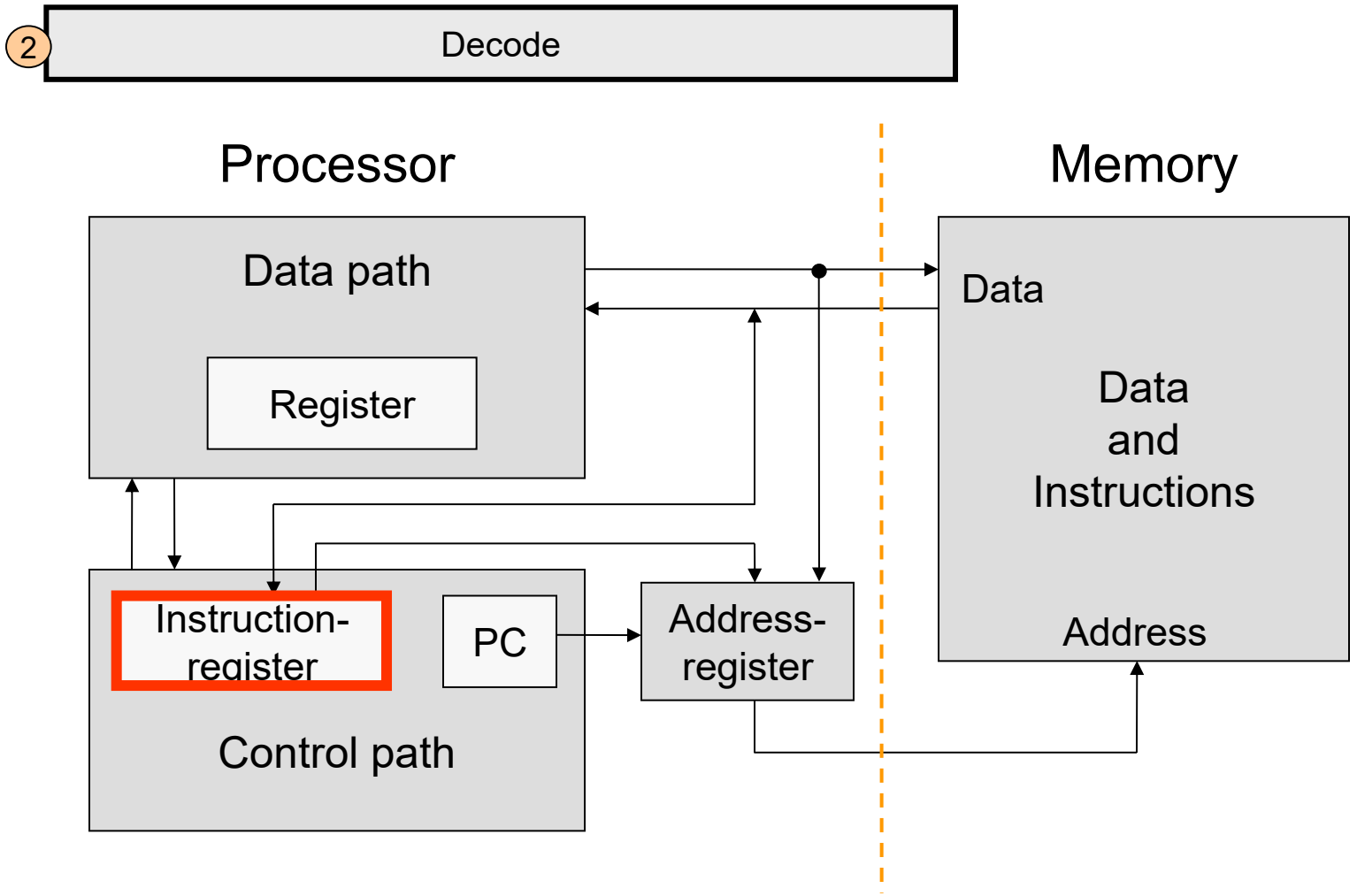
Execution Cycle



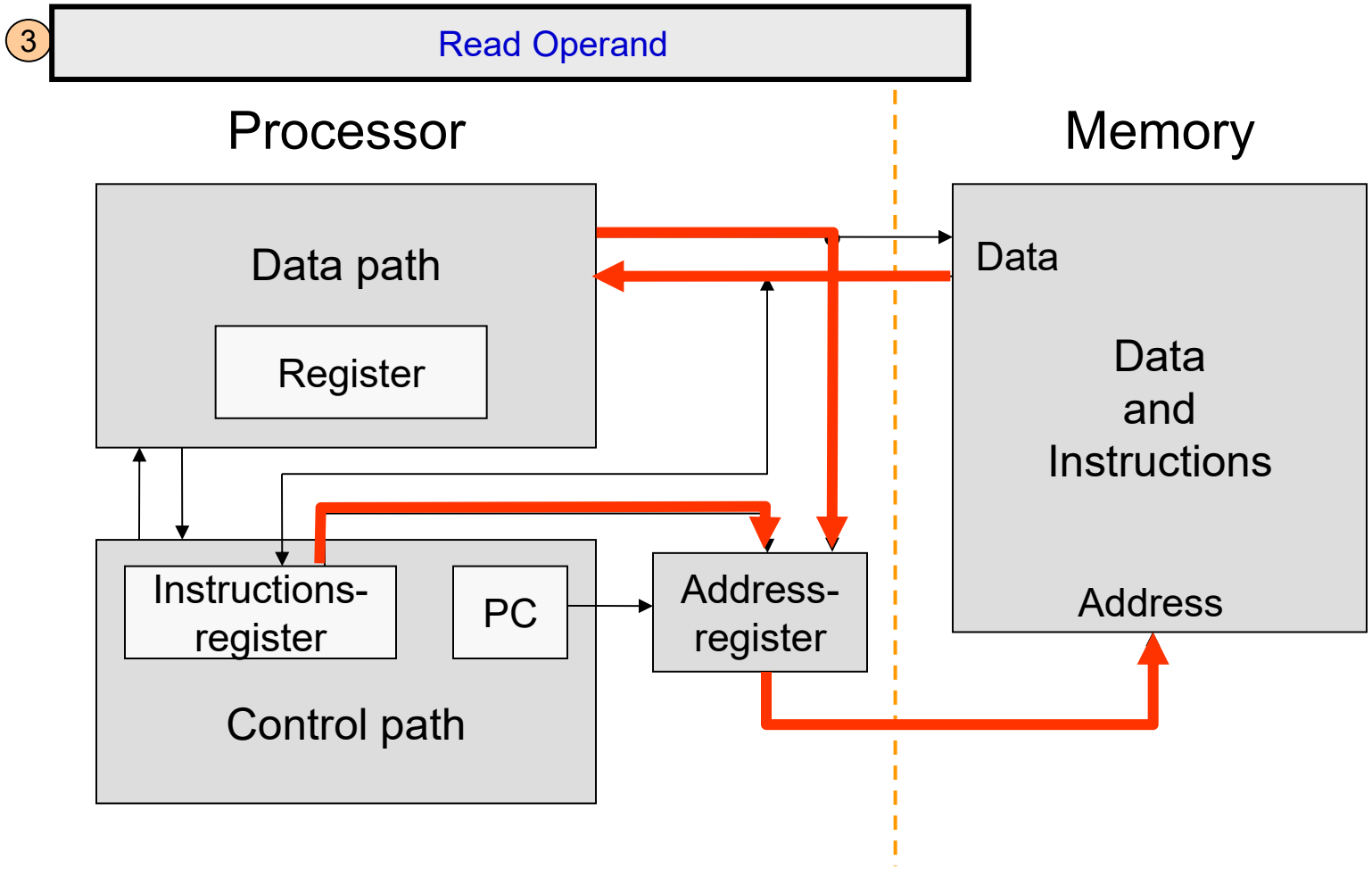
Instruction fetching



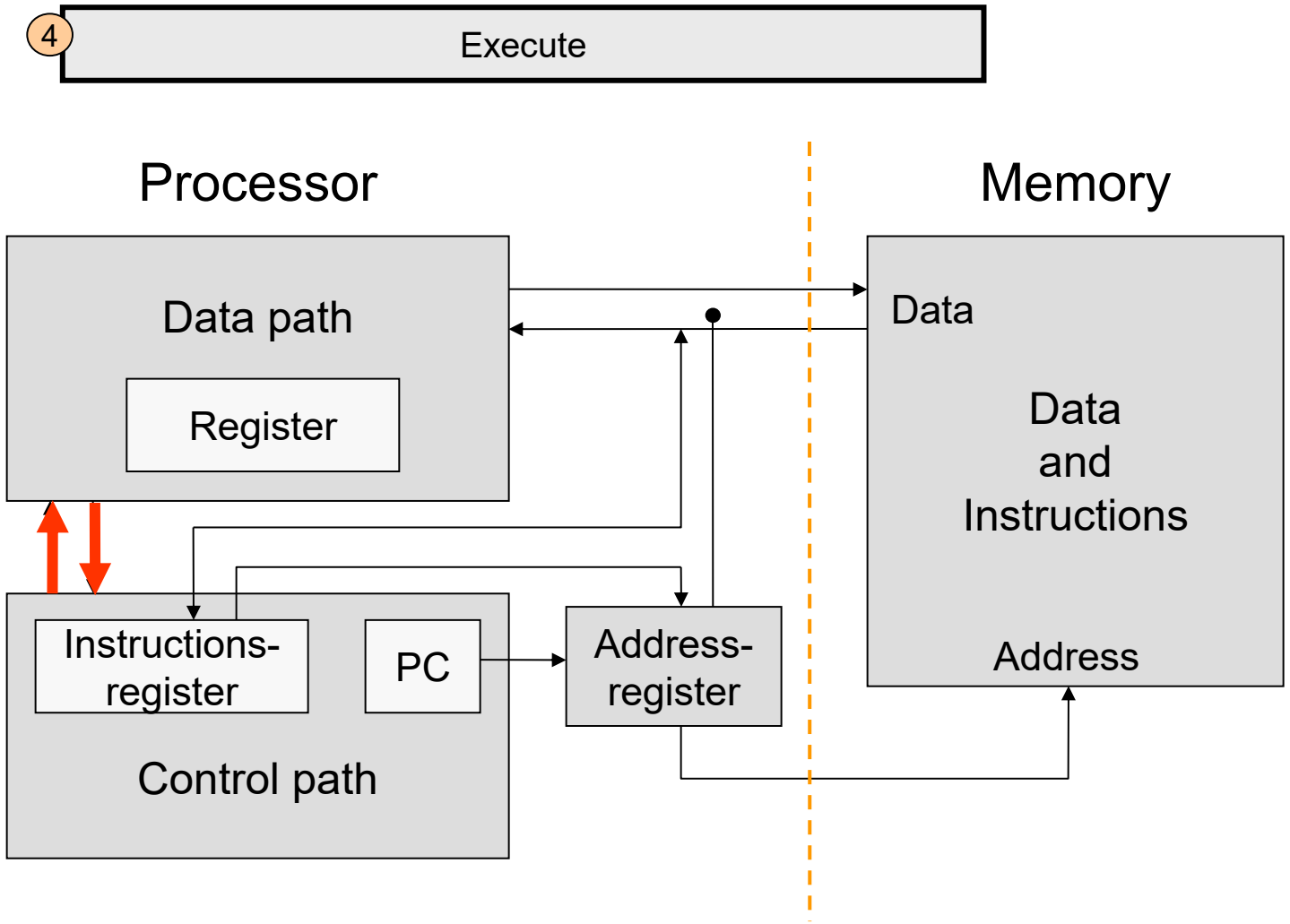
Instruction decode



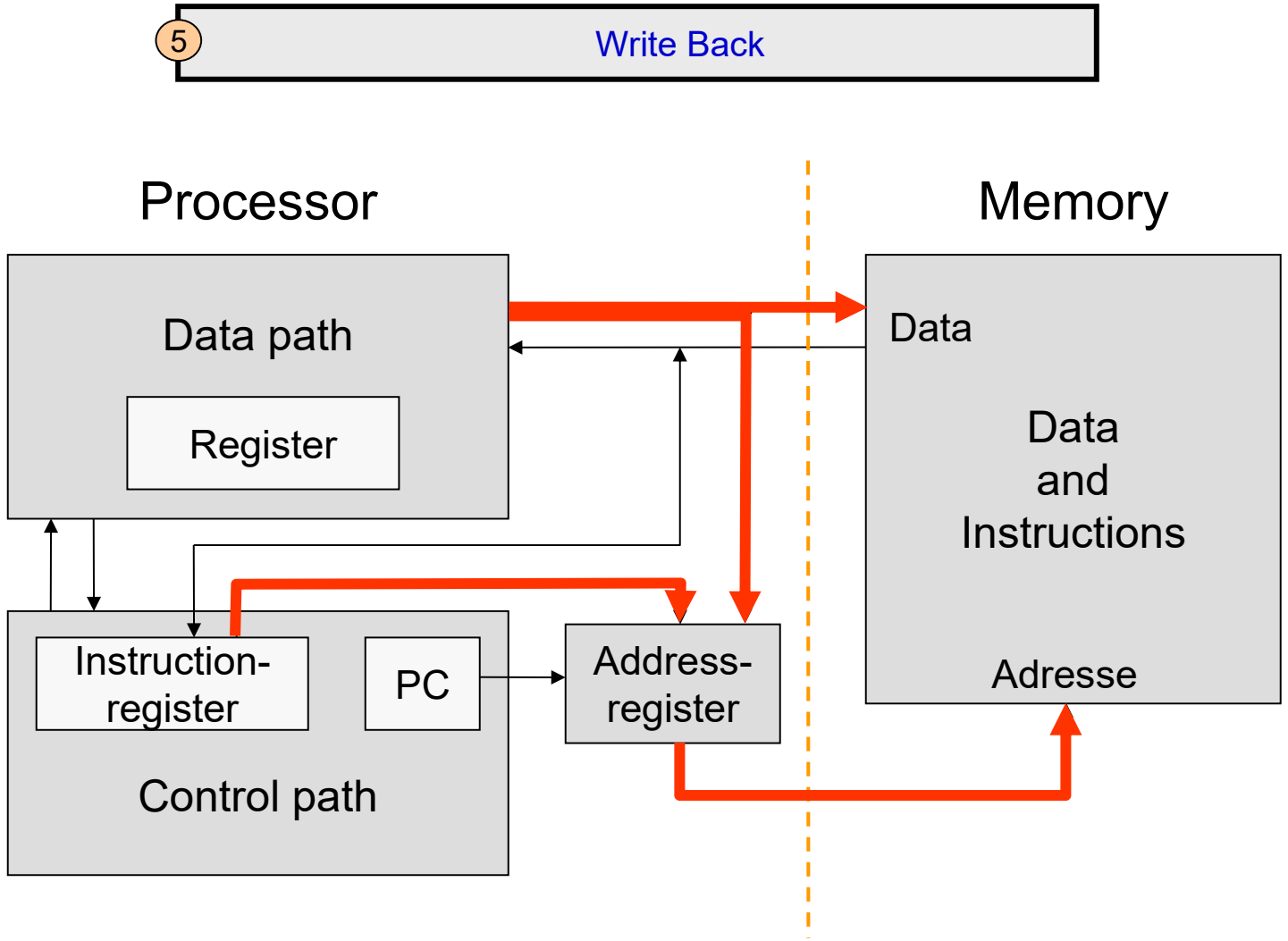
Read Operands



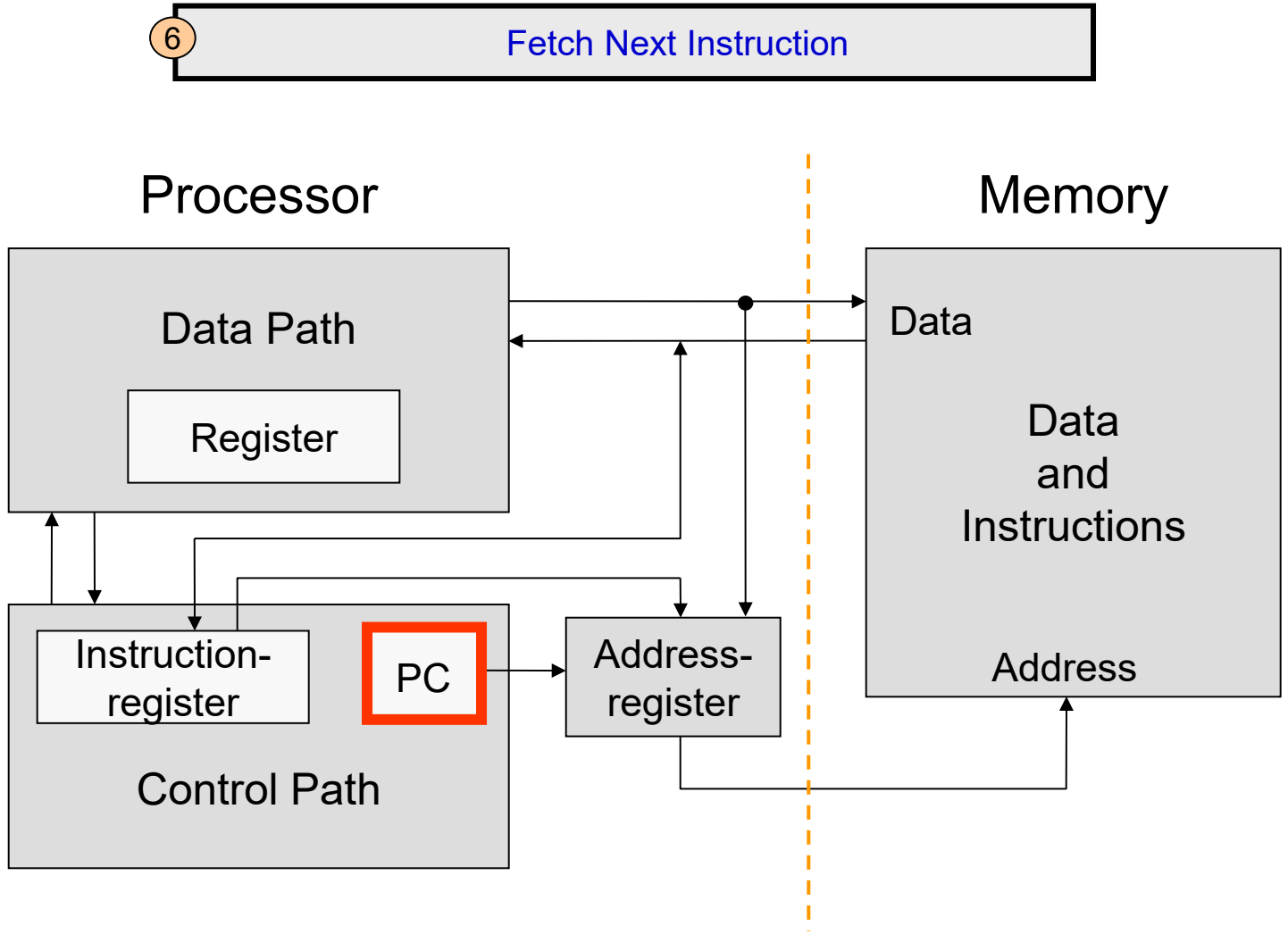
Execute


4
Execute
Processor
Memory
Data path
Register
Data
Data
and
Instructions
Address
Instructions-
register
PC
Address-
register
Control path

Result write back

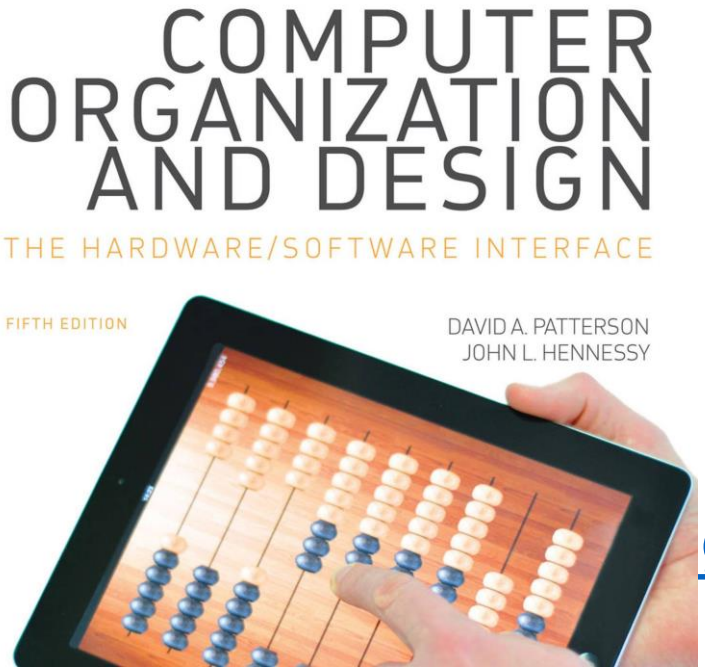


Next Instruction fetching



Literature

Computer Organization and Design MIPS Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Comput Architecture and Design)
by David A. Patterson, John L. Hennessy,
ISBN-13: 978-0124077263



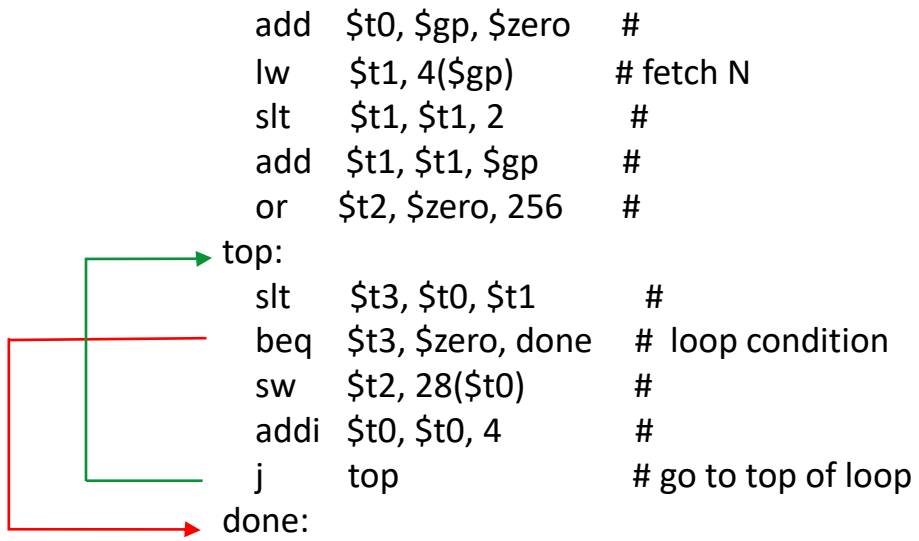
https://www.elsevier.com/books/computer-gn-mips-edition/patterson/978-0-12-407726-3

Overview

- Datapath's Components
- Once Cycle Implementation
- Multiple Cycle Implementation

Implementing a MIPS Subsets

- Instruction Set Architecture (ISA) is a simplified MIPS-ISA
  - Data transfer: `lw, sw`
  - Branching: `beq, j`
  - Arithmetic/Logic: `add, sub, and, or, slt`
- Example



```
        add   $t0, $gp, $zero     #
        lw    $t1, 4($gp)         # fetch N
        slt   $t1, $t1, 2         #
        add   $t1, $t1, $gp       #
        or    $t2, $zero, 256     #
top:
        slt   $t3, $t0, $t1       #
        beq   $t3, $zero, done    #  loop condition
        sw    $t2, 28($t0)        #
        addi  $t0, $t0, 4         #
        j     top                 # go to top of loop
done:
```

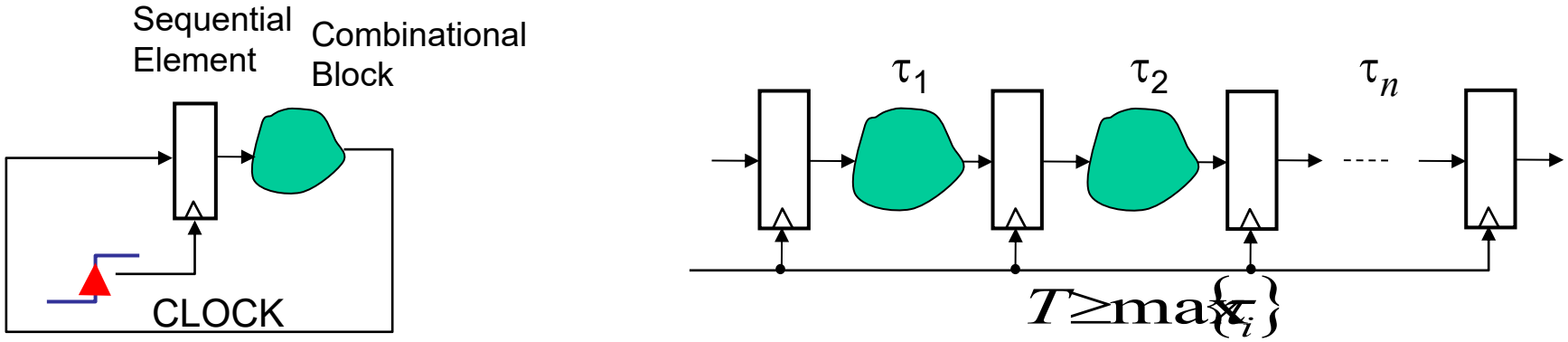
Implementing a MIPS Subset

- Datapath Design
  - Define components and their interconnects
  - Define control and status signals

- Controller Design
  - Define a state for each execution step
  - For each state, define the next states and the control signals
  - Implement the controller (hardwired or microprogrammed)

Clocking

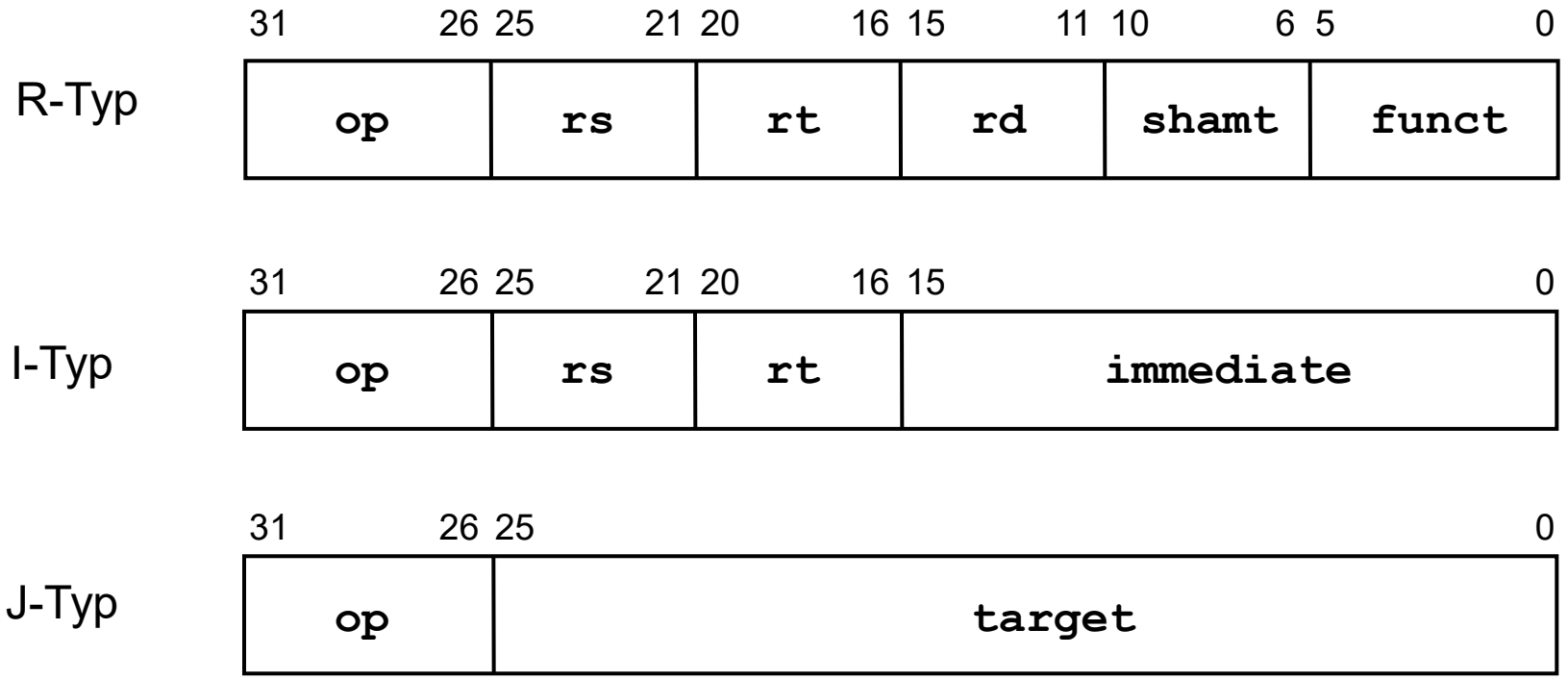
- The whole CPU consists of combinational and sequential parts
- We use a system clock (CLOCK). All registers are synchronous edge-triggered
  - Sequential element can be simultaneously read and be written in 1 clock



- The clock period is the longest path between two registers

Instruction Format

- Instructions: `add, sub, and, or, slt, beq, lw, sw`
- Instruction's format:



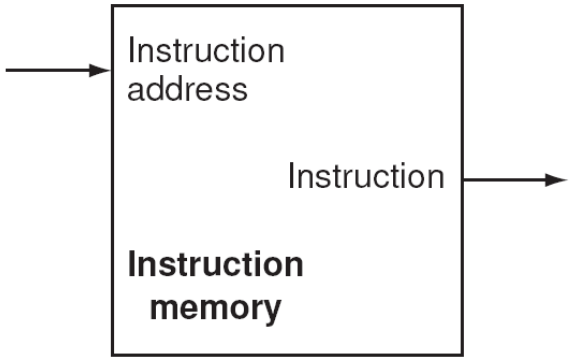
Datapath's Components

1. Instruction Load, Incrementing Program Counter (PC)
   - Each instruction cycle, an instruction is fetched from the memory
   - Register PC holds the address of the next instruction
   - In general, the address of the next instruction is PC ← PC + 4

   - Components needed for instruction fetching
     - Instruction memory
     - Register PC
     - Adder

Datapath's Components

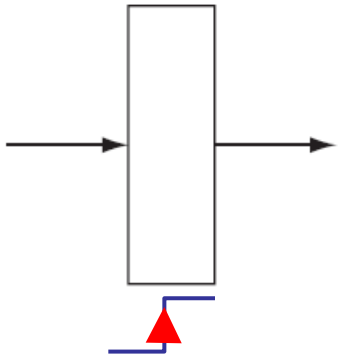
- Components



Instruction Memory

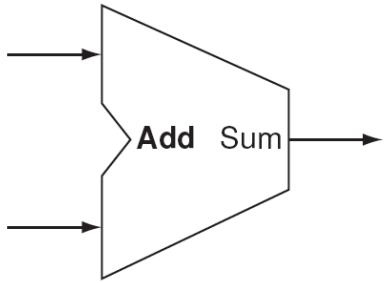
Input: Address (32 Bit)
Output: Instruction (32 Bit)

Because the memory can only be read, it can be considered as a combinational element



Program Counter (PC)

New value is loaded in PC with the clock

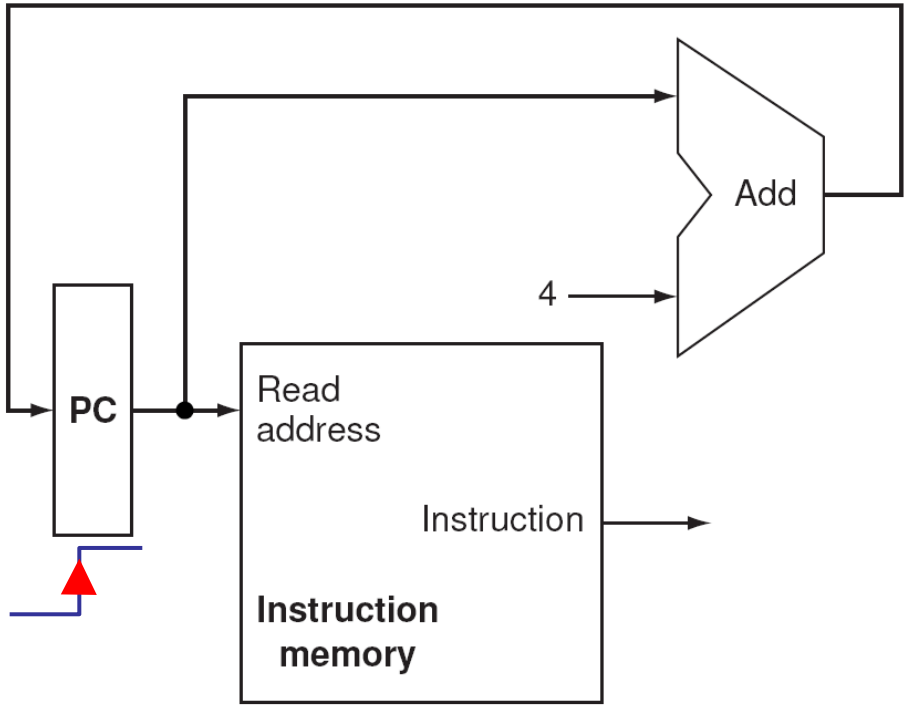


Adder

PC← PC + 4

Datapath's Components

- Segment of the Datapath for instruction fetch and PC increment

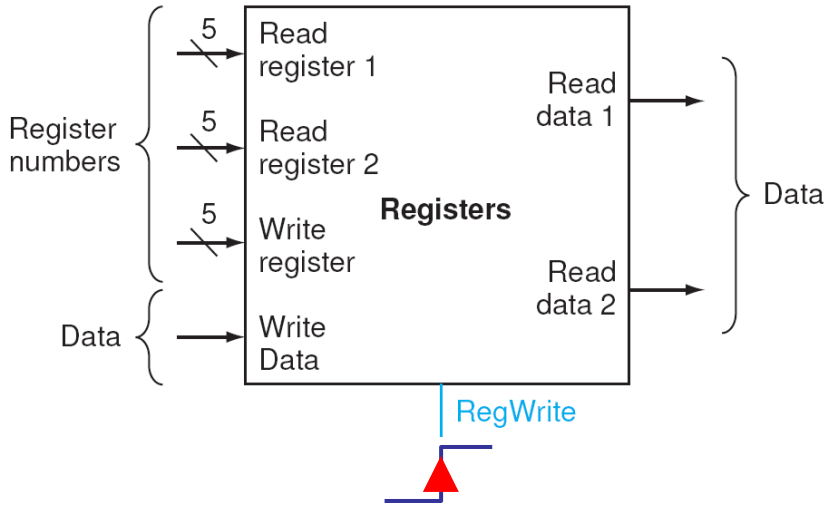


Datapath's Components

2. Arithmetic and Logic Instructions

- Read 2 registers, execute operation and store result in register

- Components
  - Register file
  - ALU



Register file

Inputs:
– 2 register addresses (5 Bit), which point to the registers to read (operand)

– 1 register address (5 Bit), which points to the register to write (result)

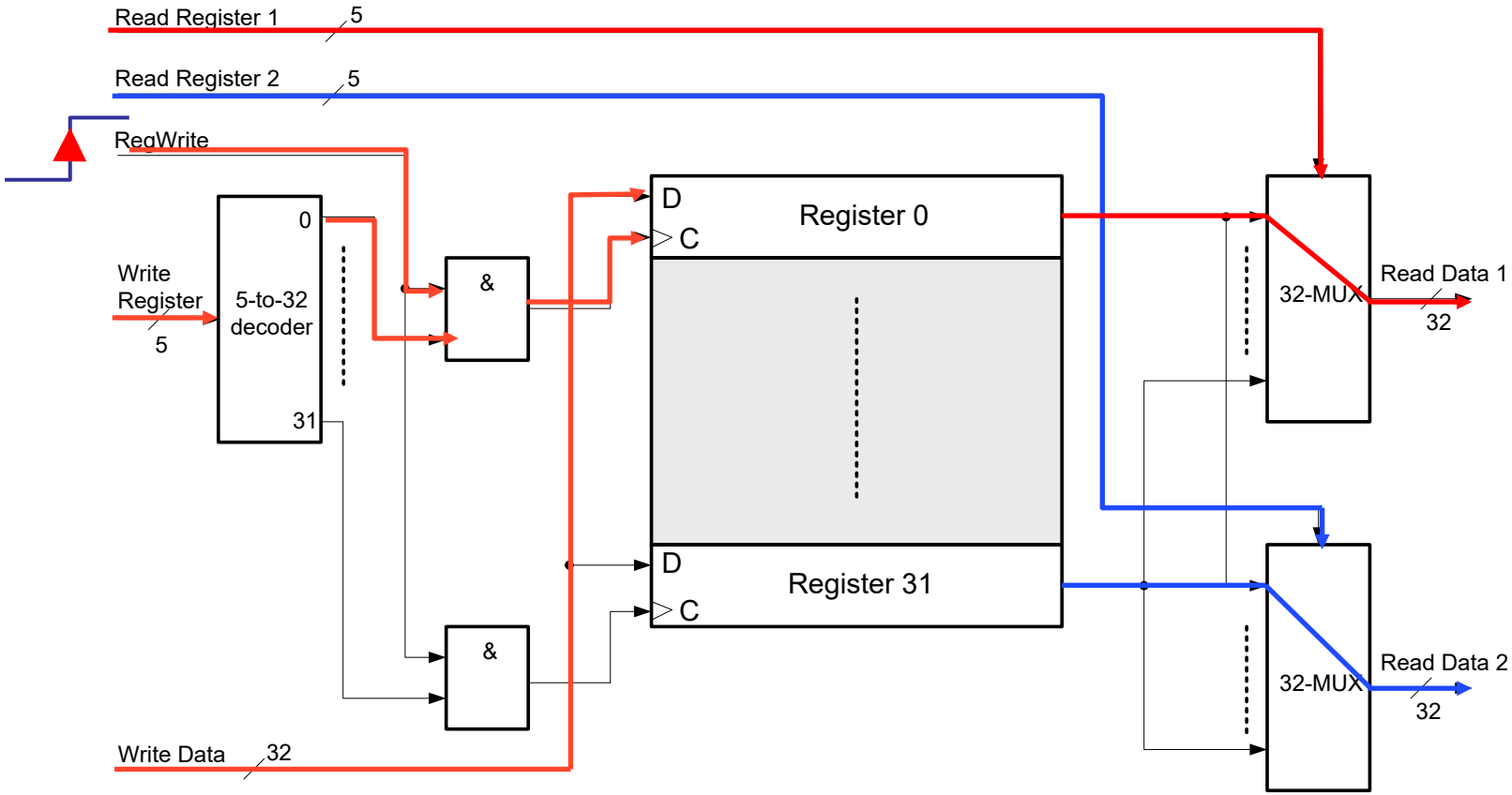
– Data input for writing (32 Bit)

– clocked signal RegWrite to control the writing

Outputs:
– 2 Data output (32 Bit) to read the two operand registers

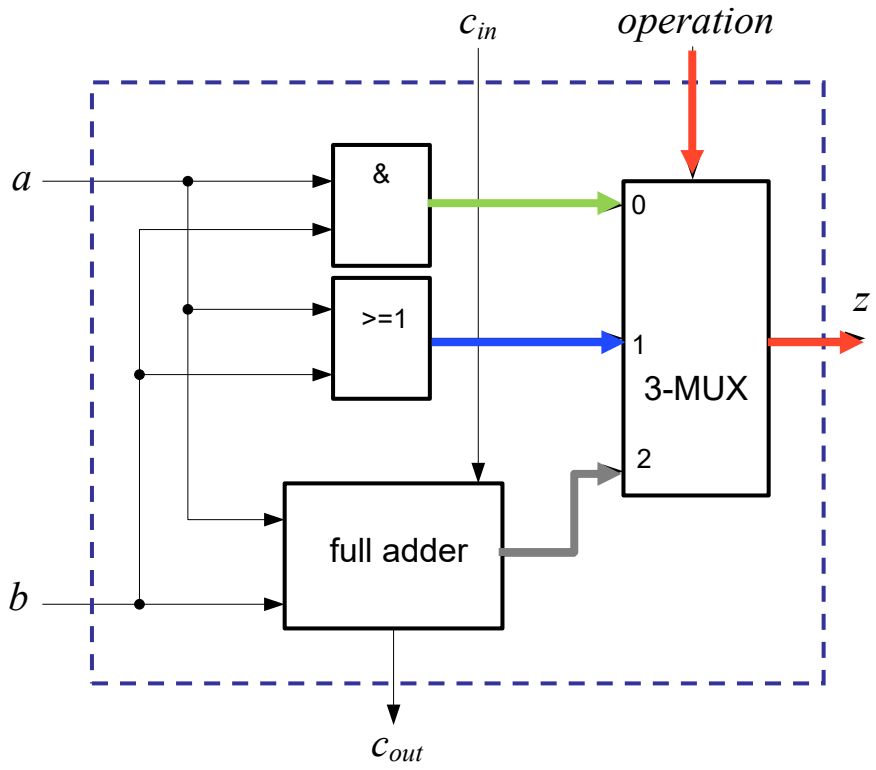
Datapath's Components

- Register file with 32 registers, 2 reads and one write ports



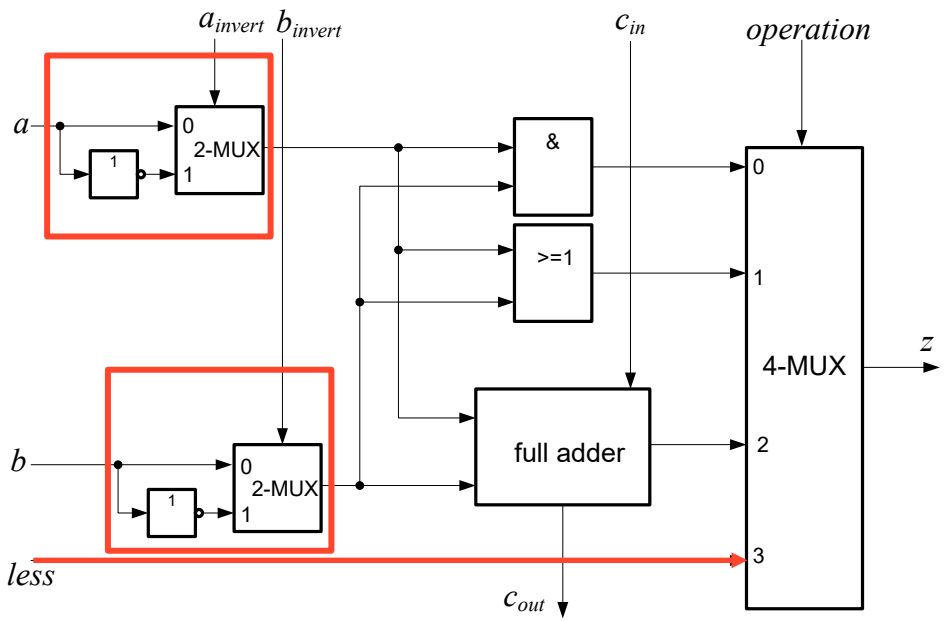
Datapath's Components

- 1-bit ALU for operations AND, OR, ADD

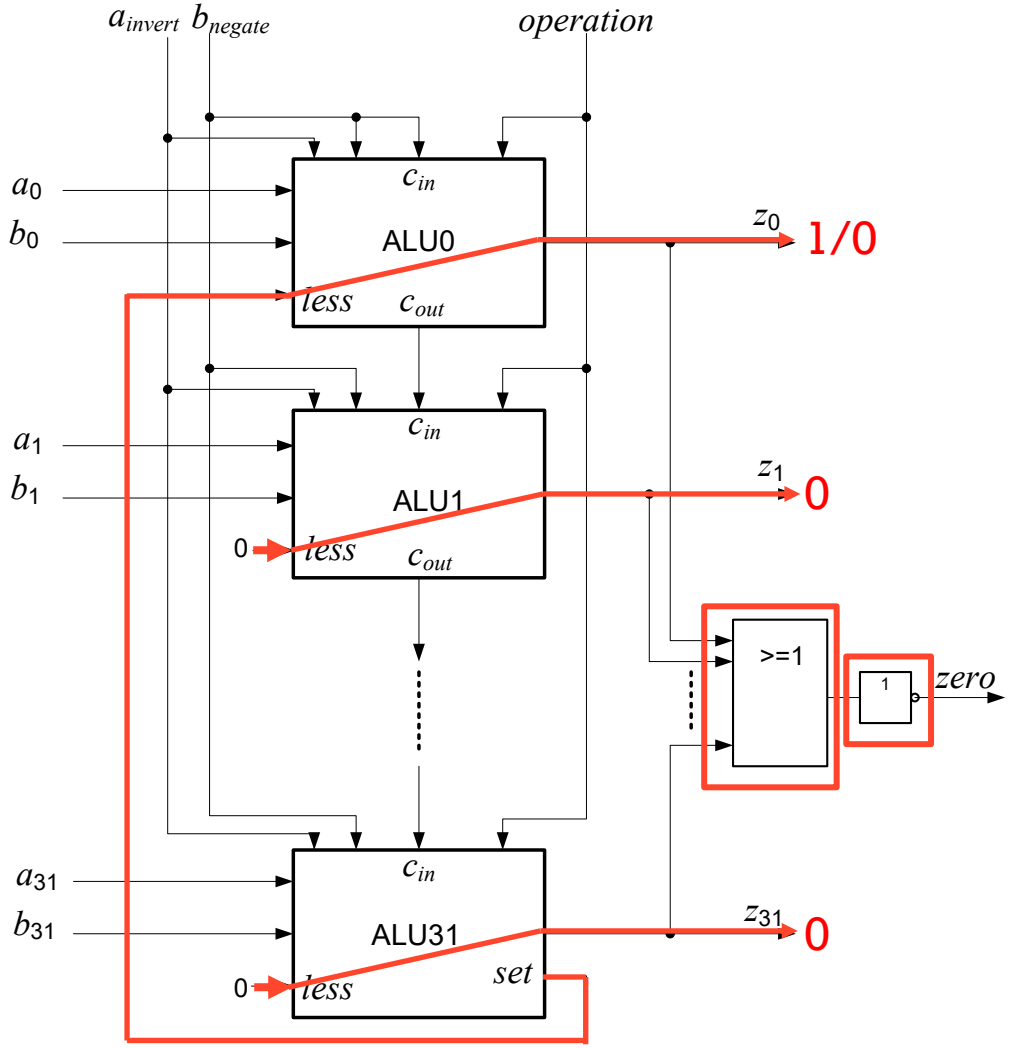


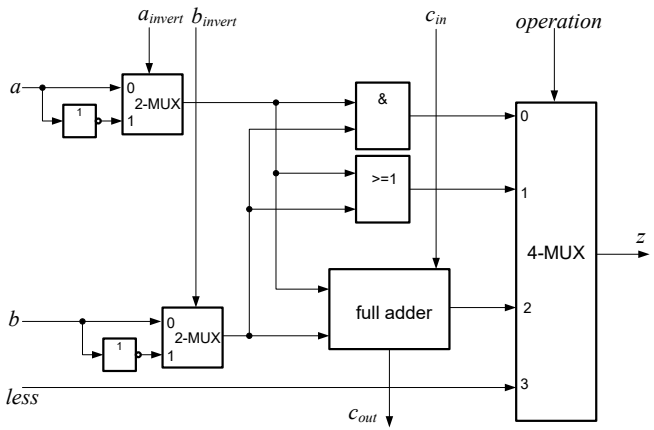
ALU Extension

- `sub` through addition of 2's-complement → invert b, $C_{in}(0) = 1$
- `nor` through $\overline{a+b} = \overline{a} \,.\, \overline{b}$ → invert a, invert b
- `slt` through $a-b < 0$?
  - Most significant bit generates a *set*-Signal that tells if $a<b$;
    less(LSB) = *set*; *less*(Digit 1..31) = 0



32-Bit ALU





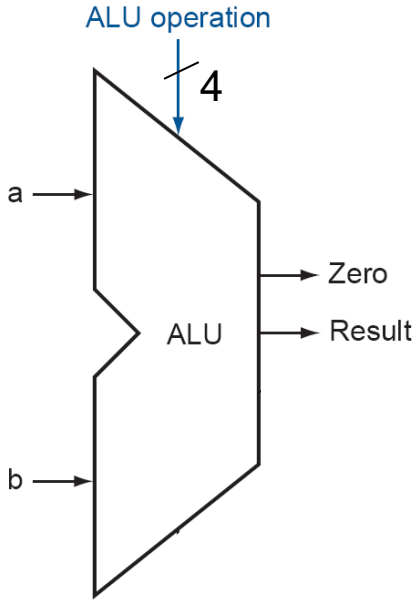
- `sub` through $b_{negate} = 1$

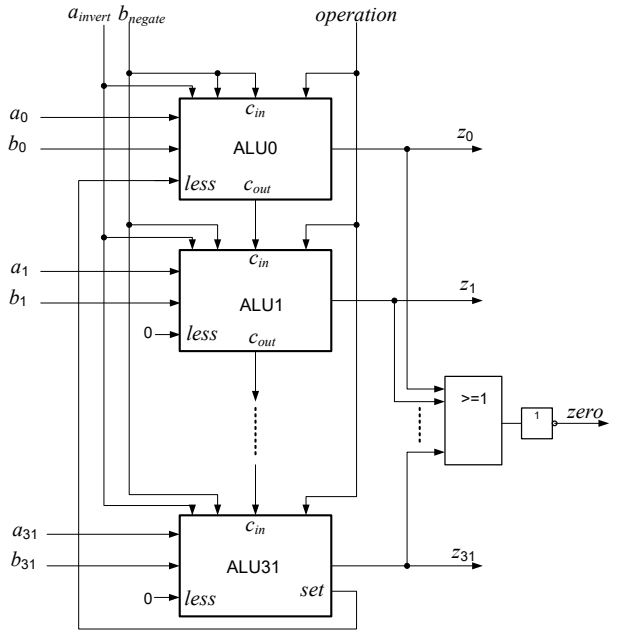
$\rightarrow b_{invert} = C_{in}(0) = 1$

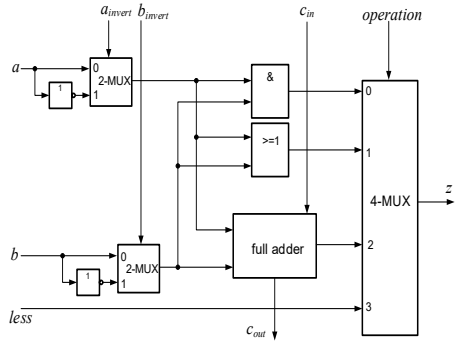
- `beq` through $a–b = 0$?

$\rightarrow zero = z_0 + \ldots + z_{31}$

ALU Input/Output





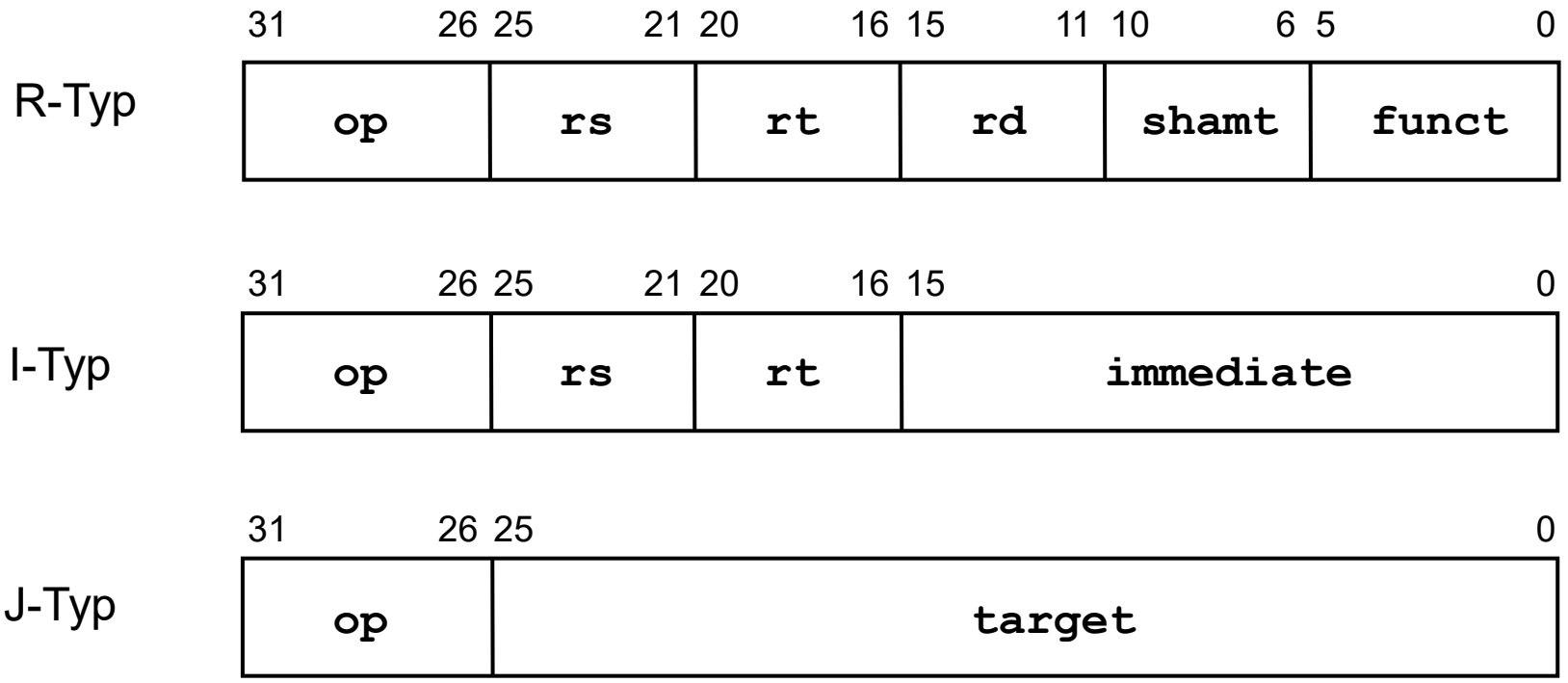


ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

a_{invert} b_{negate} *operation* (2 Bit)

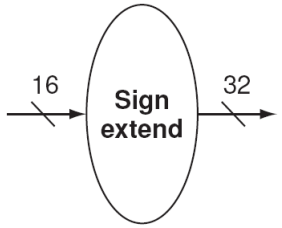
Instruction Format

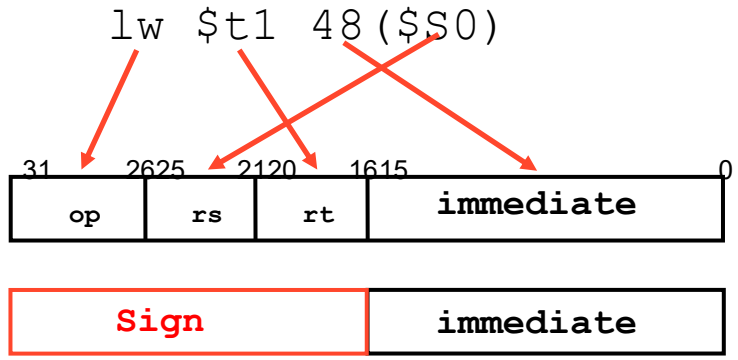
- Instructions: `add, sub, and, or, slt, beq, lw, sw`
- Instruction's format:



Datapath's Components

3. Data transfer instructions
   - Compute a memory address
   - Transfer data from register to memory or vice versa
   - Components
     - Sign extension
     - Data memory
     - ALU,
     - Register file





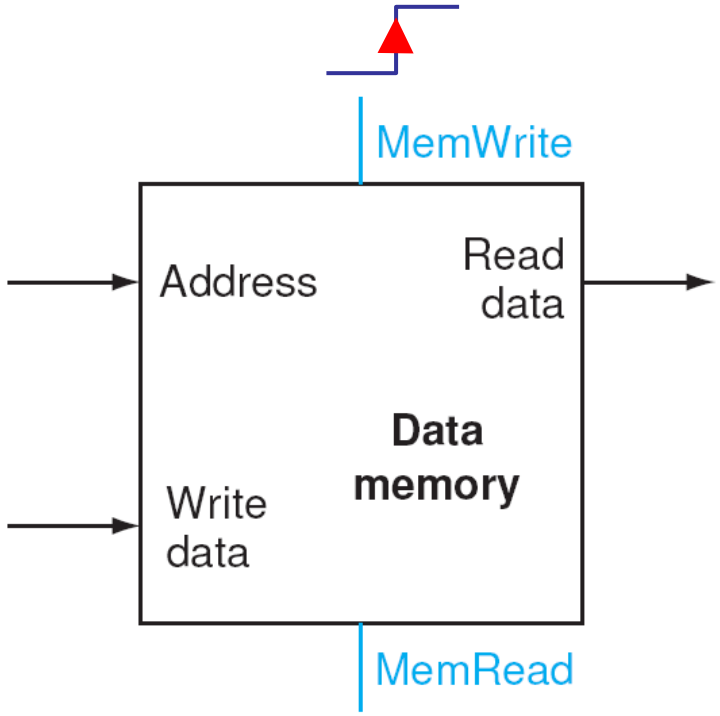
Sign Extension

Input: Operand in 2-complement (16 Bit)
Output: Result in 2-complement (32 Bit)

Copy bit 15 of operand in all higher positions (16-31) of the result

Datapath's Components

- Components



Data Memory

Input:
- Address (32 Bit)
- Write Data (32 Bit)

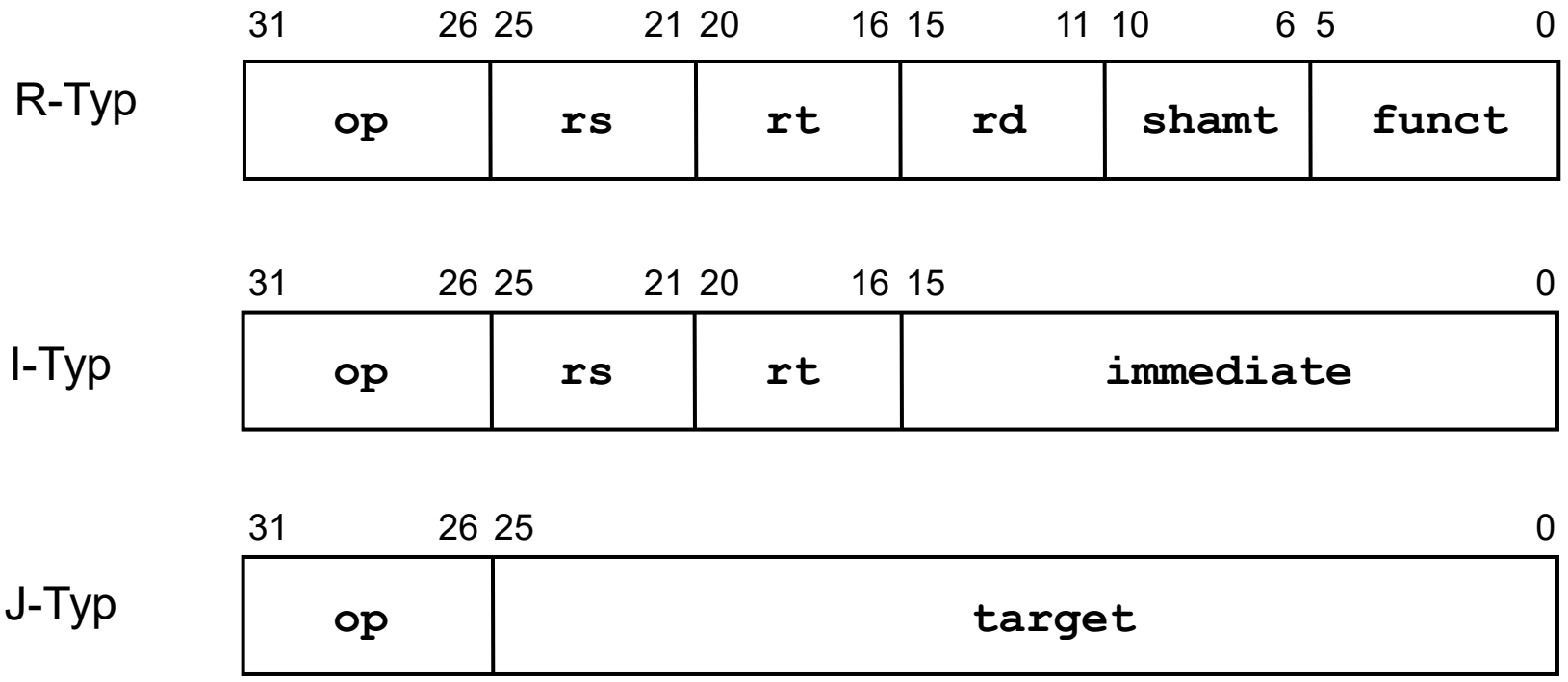
- Clocked control signal MemWrite tells, when data must be written in memory

- Control signal MemRead tells, when data are ready on the read port

Output: read data (32 Bit)

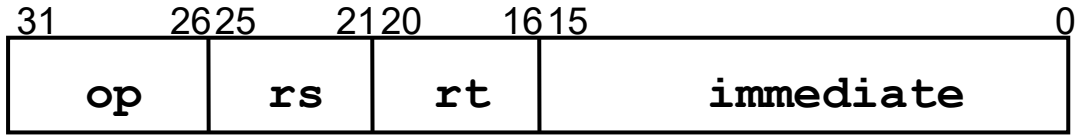
Instruction Format

- Instructions: `add, sub, and, or, slt, beq, lw, sw`
- Instruction's format:



Datapath's Components

4. Branch Instructions

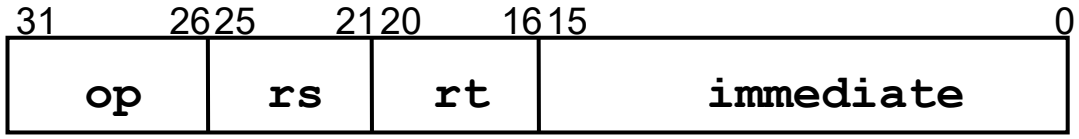


- Define if a branch instruction must be executed or not
  - If branch condition true, go to branch address
  - Otherwise, continue execution sequentially (after current instruction)

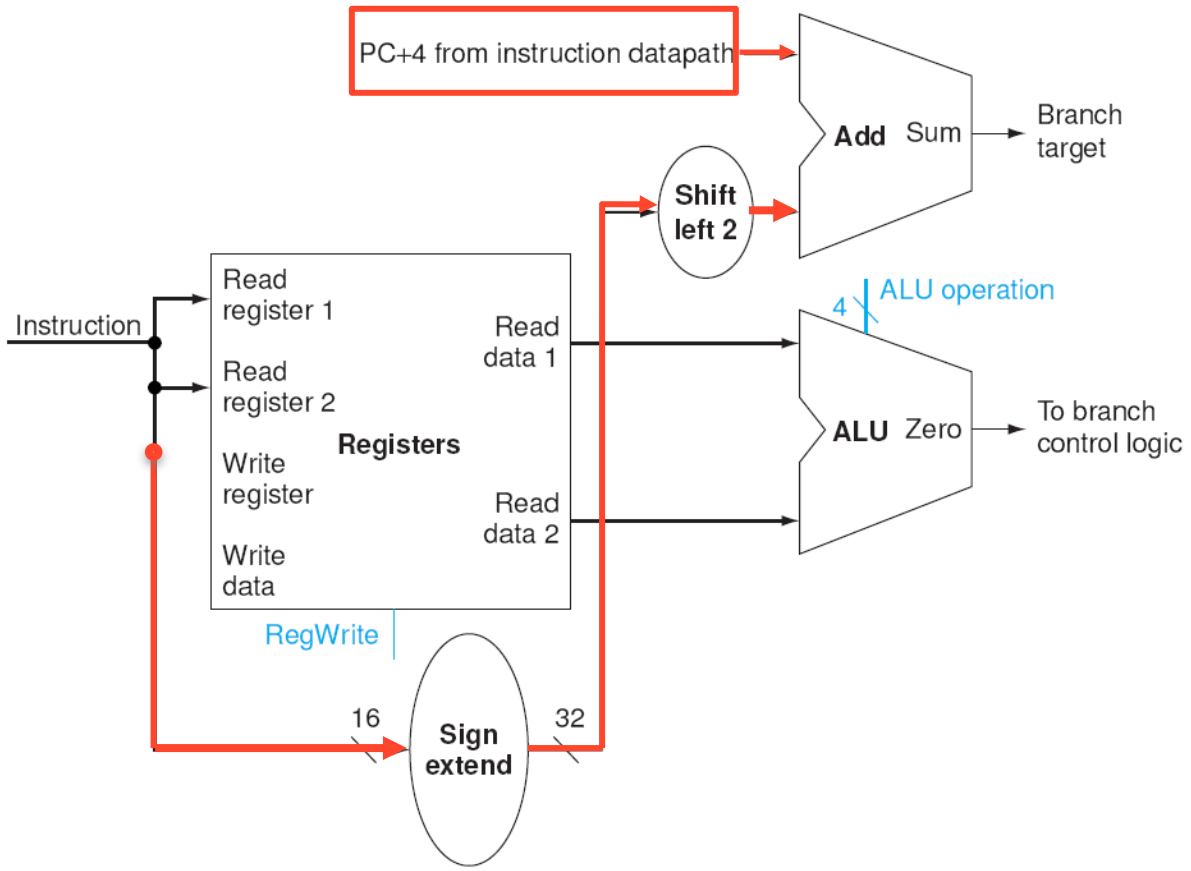
- Compute the branch address from a Base and Offset:
  - In MIPS, the base for conditional branching is the address of the next instruction after the branch (PC+4)
    → the 16-bit Offset must be added to (PC+4)
  - The offset-field must be shifted 2 Bit left (Word address)

- Components
  - Register file, ALU, Sign Extension
  - Shifter, Adder

Datapath's Components



- Part of the data path for conditional branching

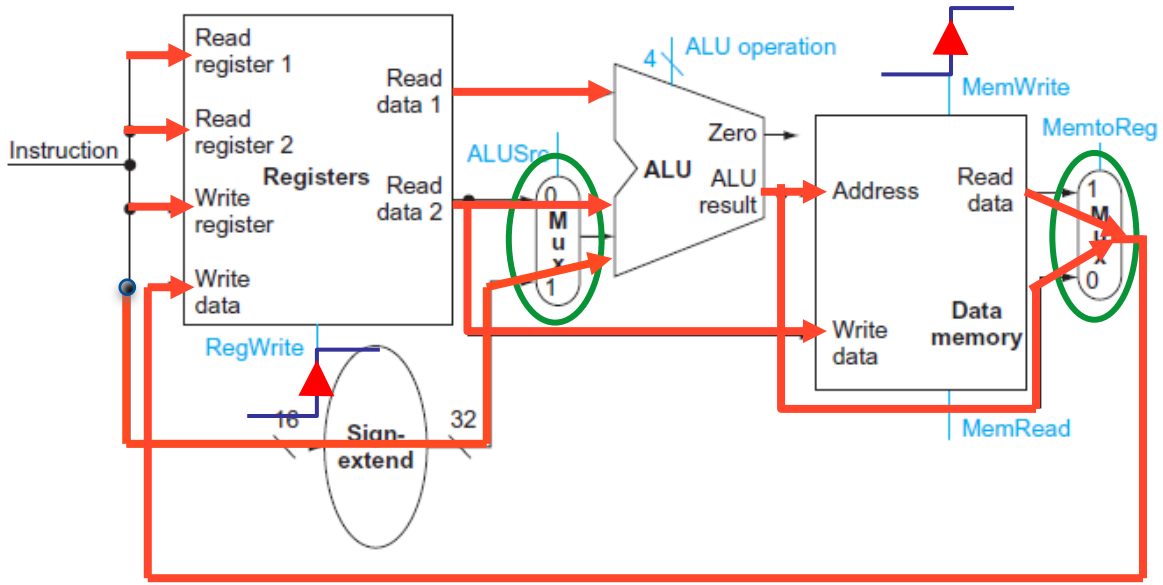


Datapath – 1-Cycle Implementation

- Approach
  - Instructions are executed in only 1 clock cycle
  - Every component of the datapath can be used only once during an instruction cycle
    - Instruction and data memory are separated
  - Multiplexers allow instructions of different classes to (re)use the same datapath
- Advantage
  - simple, in particular controller design
- Drawbacks
  - Instructions need different datapath components
    - Combinational delay varies from instruction to instruction
  - Clock period T = maximum delay
    - Instructions execute in just 1 clock period but a very long one

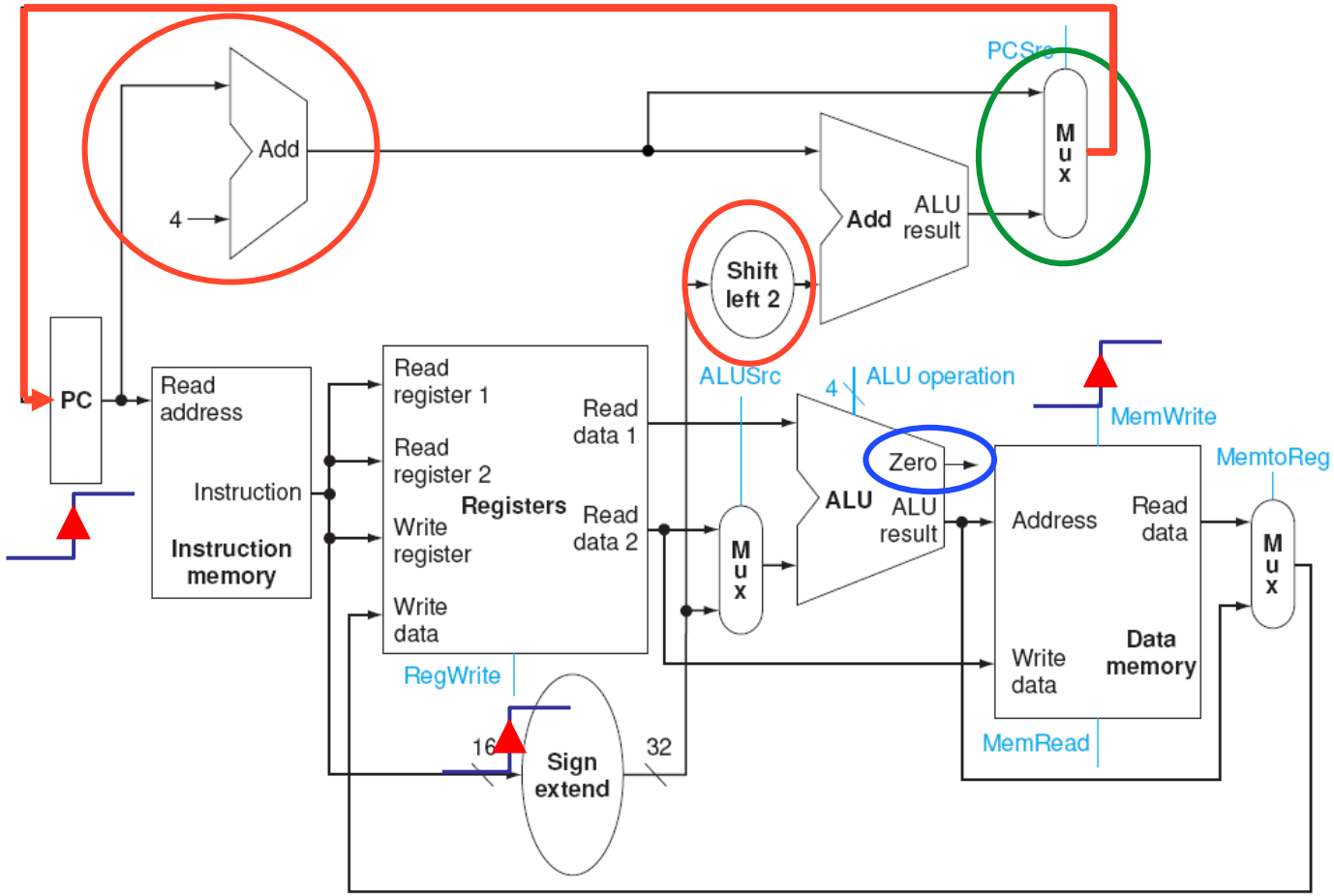
Datapath for 1-cycle Implementation

- Datapath for memory and R-Instructions



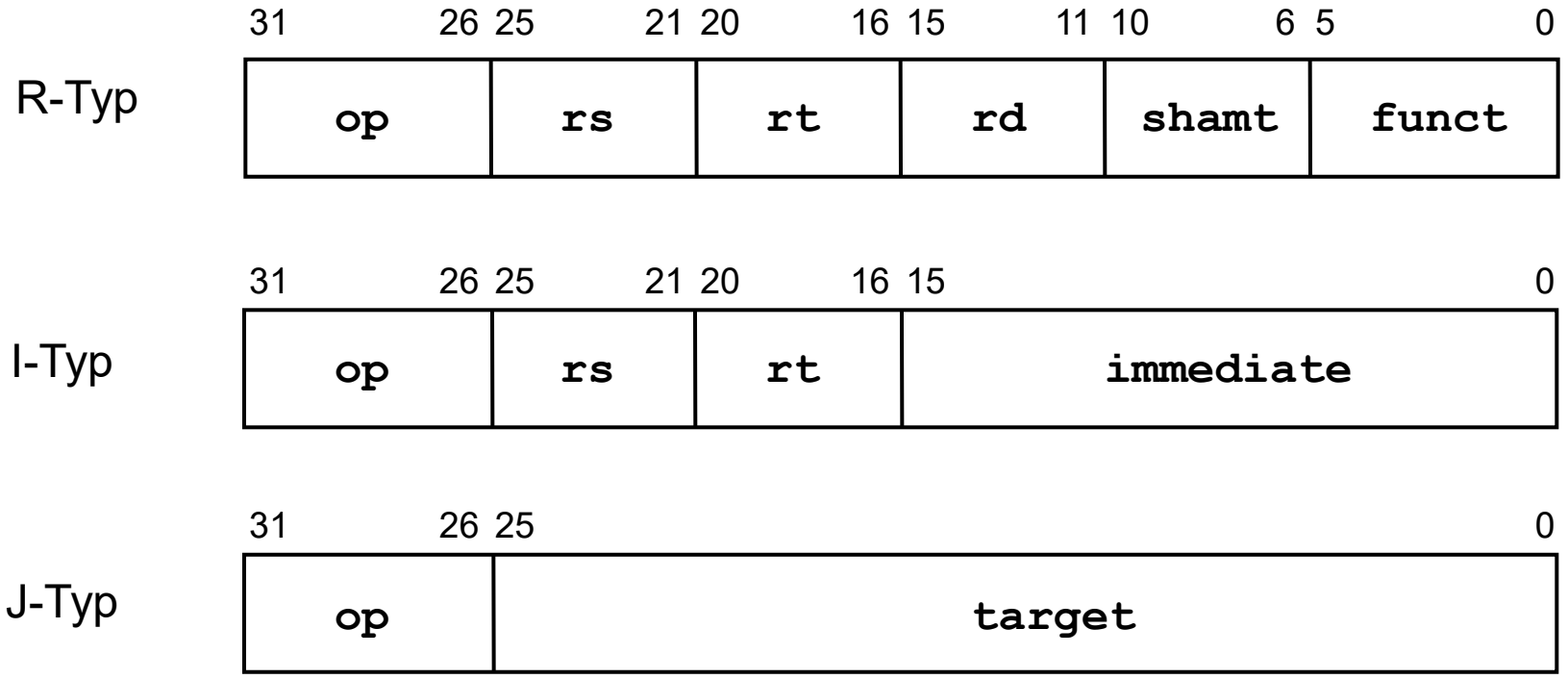
Datapath for 1-cycle Implementation

- Datapath for I- und R-Instructions

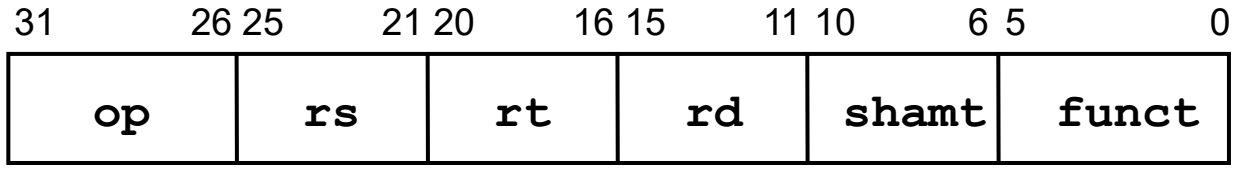


Controller Design

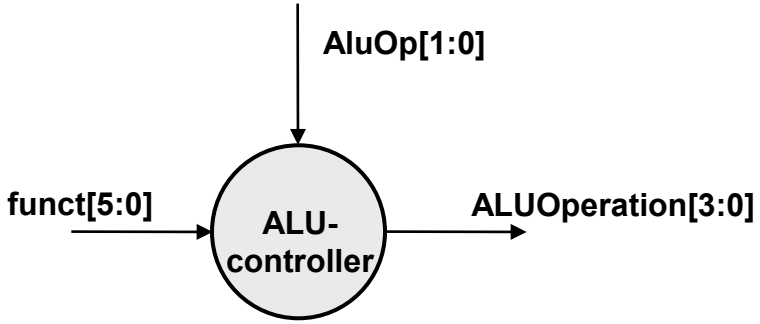
- Instructions: `add, sub, and, or, slt, beq, lw, sw`
- Instruction's format:



ALU-Controller



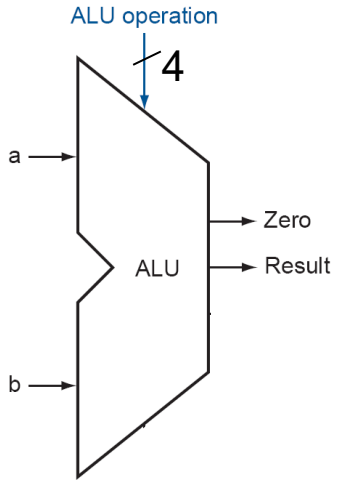
- The ALU has a separate controller
  - ALU has 4 control input, however, only 6 combinations are used (resp. 5, since `nor` instruction not considered)
  - Observation
    - Instructions `lw`, `sw`: ALU must add
    - Instruction `beq`: ALU must subtract
    - Instruction R-Typ: ALU operation depends on field funct
  - ALU-Controller



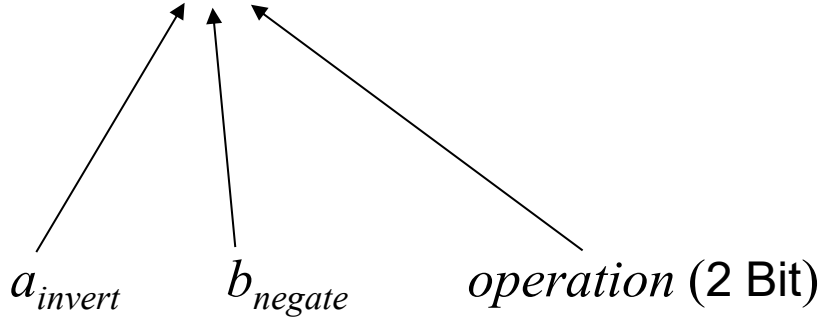
  - Advantage: the (Main)controller is simplified

ALU-Controller

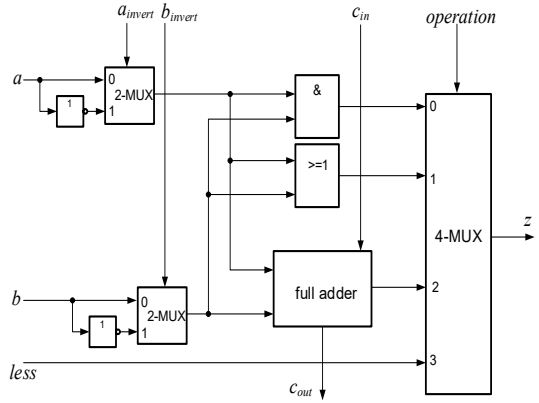
- Specification
  - X (don't care)



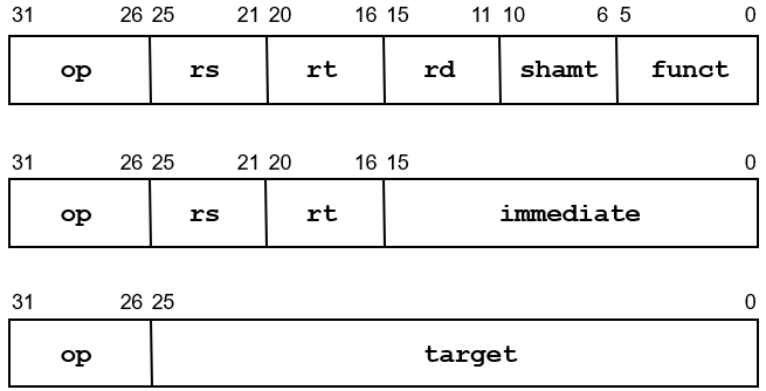
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111



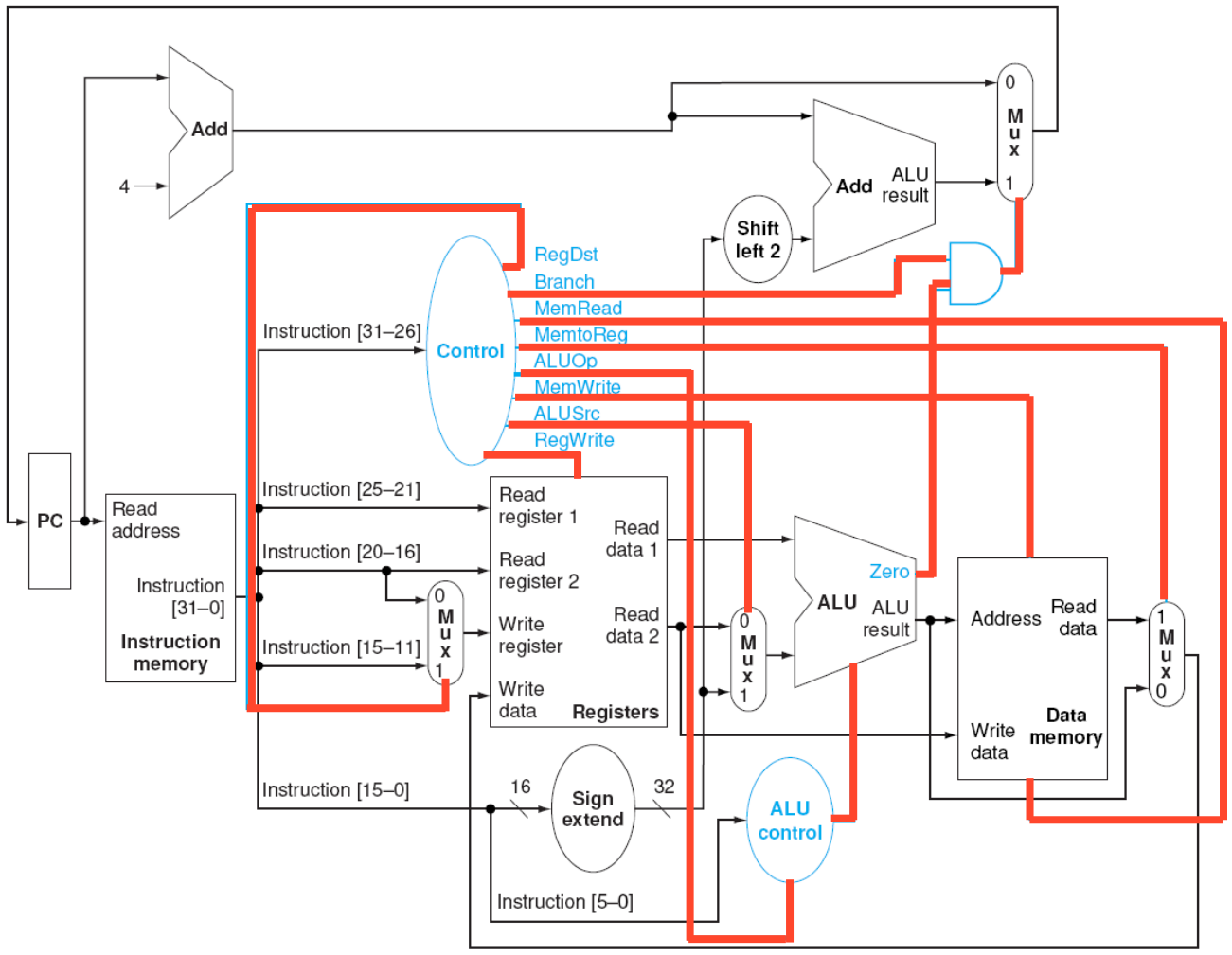
Controller Design

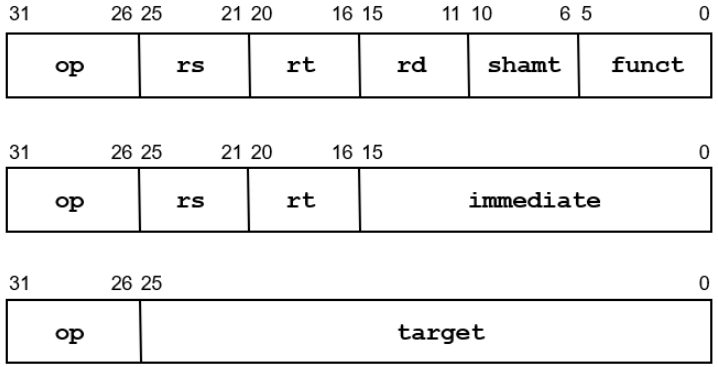


- Specification
  - Controller's input is the OP code: Instruction[31:26]
  - Read registers rs, rt indexed in Instruction[25:21] and Instruction[20:16]
  - Base register rs for `lw` and `sw` indexed in Instruction[25:21]
  - 16-bit offset for `beq`, `lw` and `sw` indexed in Instruction[15:0]
  - Target (Result) register is
    - rt for `lw`, indexed in Instruction[20:16]
    - rd for `add/sub/and/or/slti`, indexed in Instruction[15:11]

  → 1 MUX needed to select between rt and rd
  - Because only one cycle is needed to execute an instruction, the controller is pure combinational circuit!
  - Exercise: which datapath and controlpath extension are required for `j` instructions?

1-Cycle Architecture (Control+Datapath)



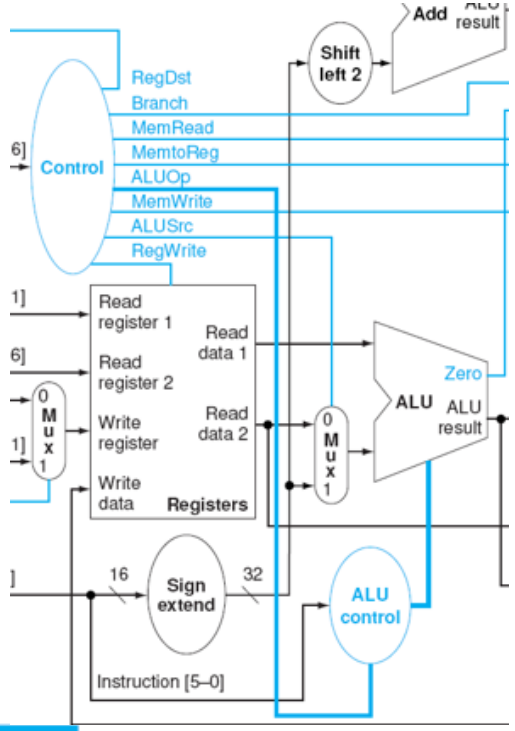


Control Signals

Control signal	Function
RegDst	0: Result Register indexed in field rt of the instruction
	1: Result Register in field rd
Branch	0: PC ← PC+4
	1: PC ← Branch address, if Zero=1
MemRead	1: read data memory
MemWrite	1: write data memory (clocked)
MemtoReg	0: ALU-Result written back into result register
	1: Transfer a memory value into result register
ALUOp[1:0]	together with funct[5:0] defines the ALU-Operation
ALUSrc	0: the lower ALU-Operand is read from the second register file output
	1: the lower ALU-Operand consists of the sign-extended immediate value of the instruction (lower 16 bits)
RegWrite	1: write result register (clocked)

Controller Design

- Truth Table

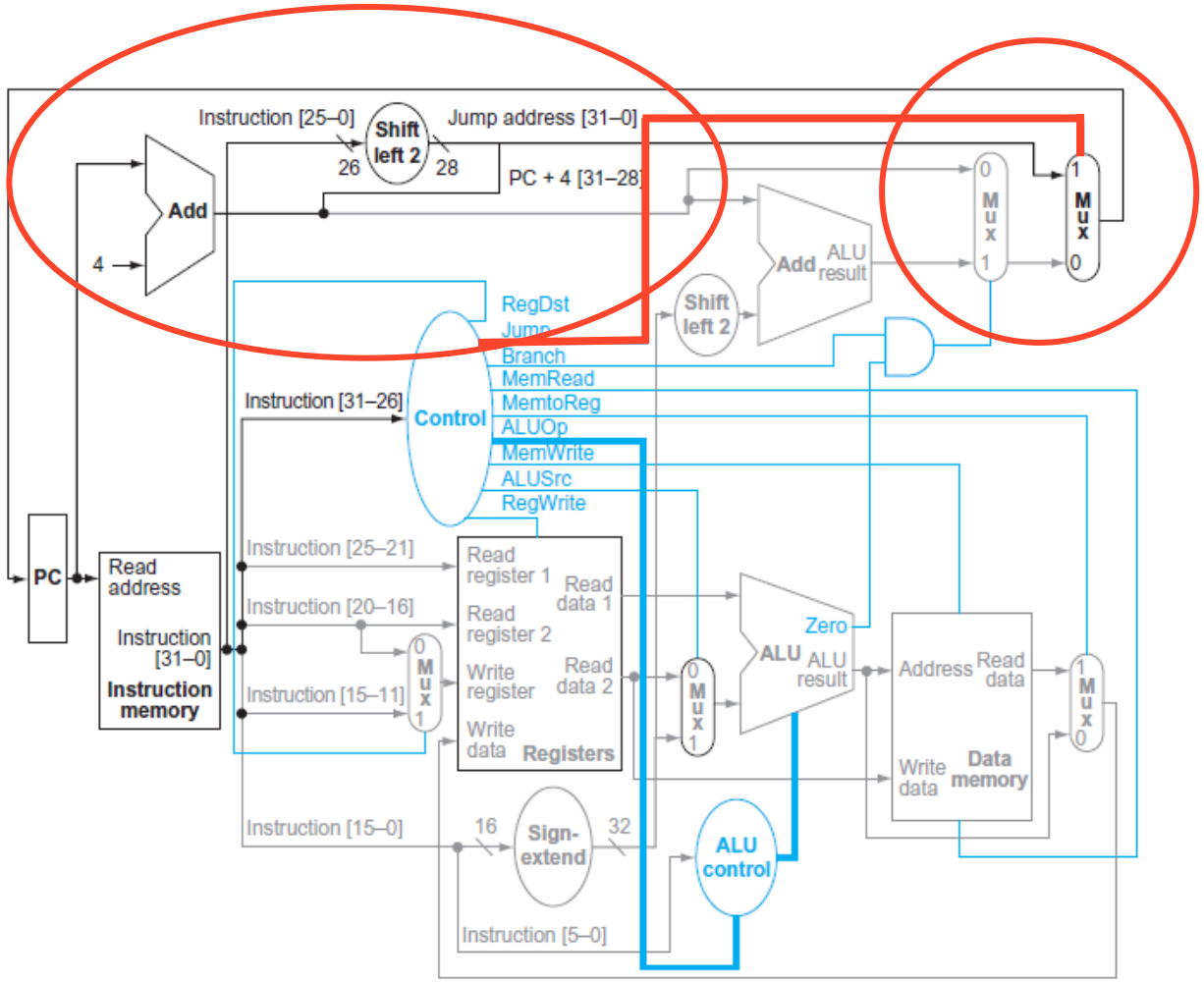


Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Controller Design

- Control and datapath extended to handle jump instructions

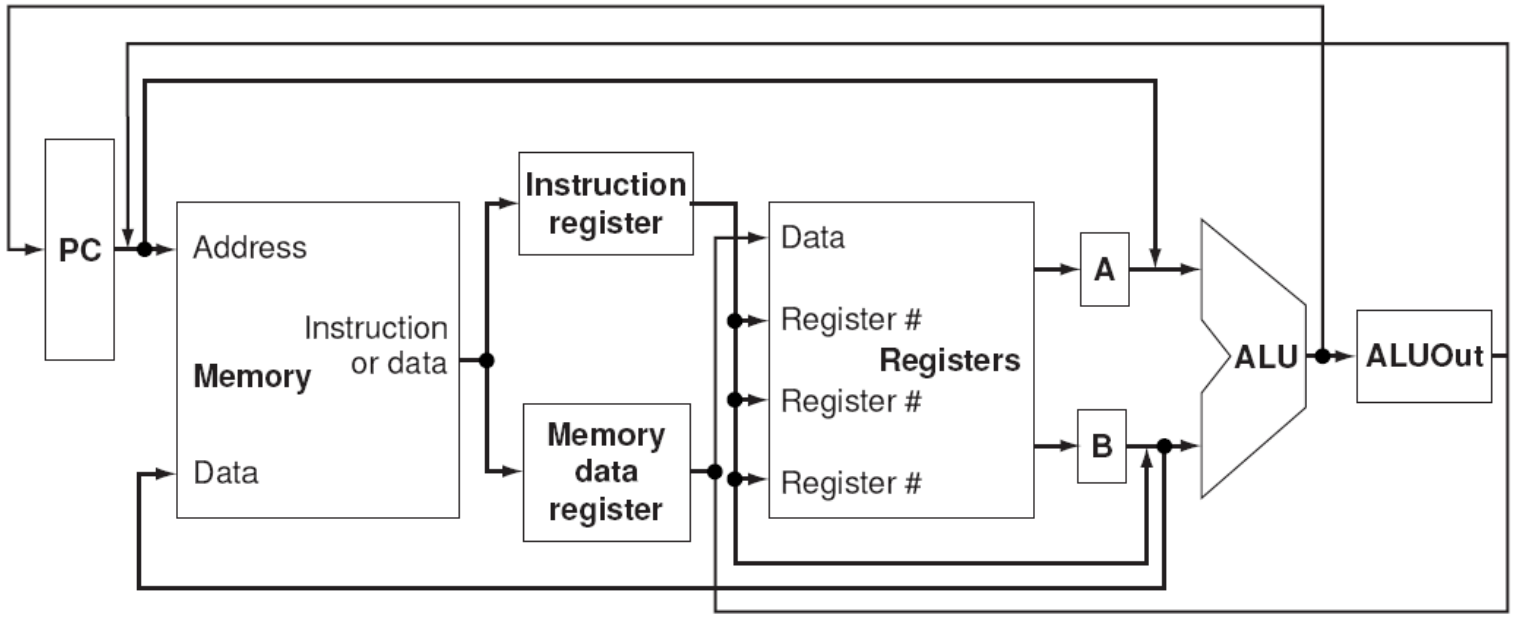


Multi-Cycle Implementation

- Approach
  - Instruction processing is divided in multiple steps
  - Each step executed in one clock cycle
  - The number of cycles varies from instruction to instruction
- Advantages
  - Clock period shorter than 1-cycle implementation → higher performance
  - Datapath components could be used multiple times in one instruction cycle → lower hardware overhead
- Drawbacks
  - Additional registers are needed to store signal values between clock cycles
  - Controller is more complex

Multi-Cycle Implementation

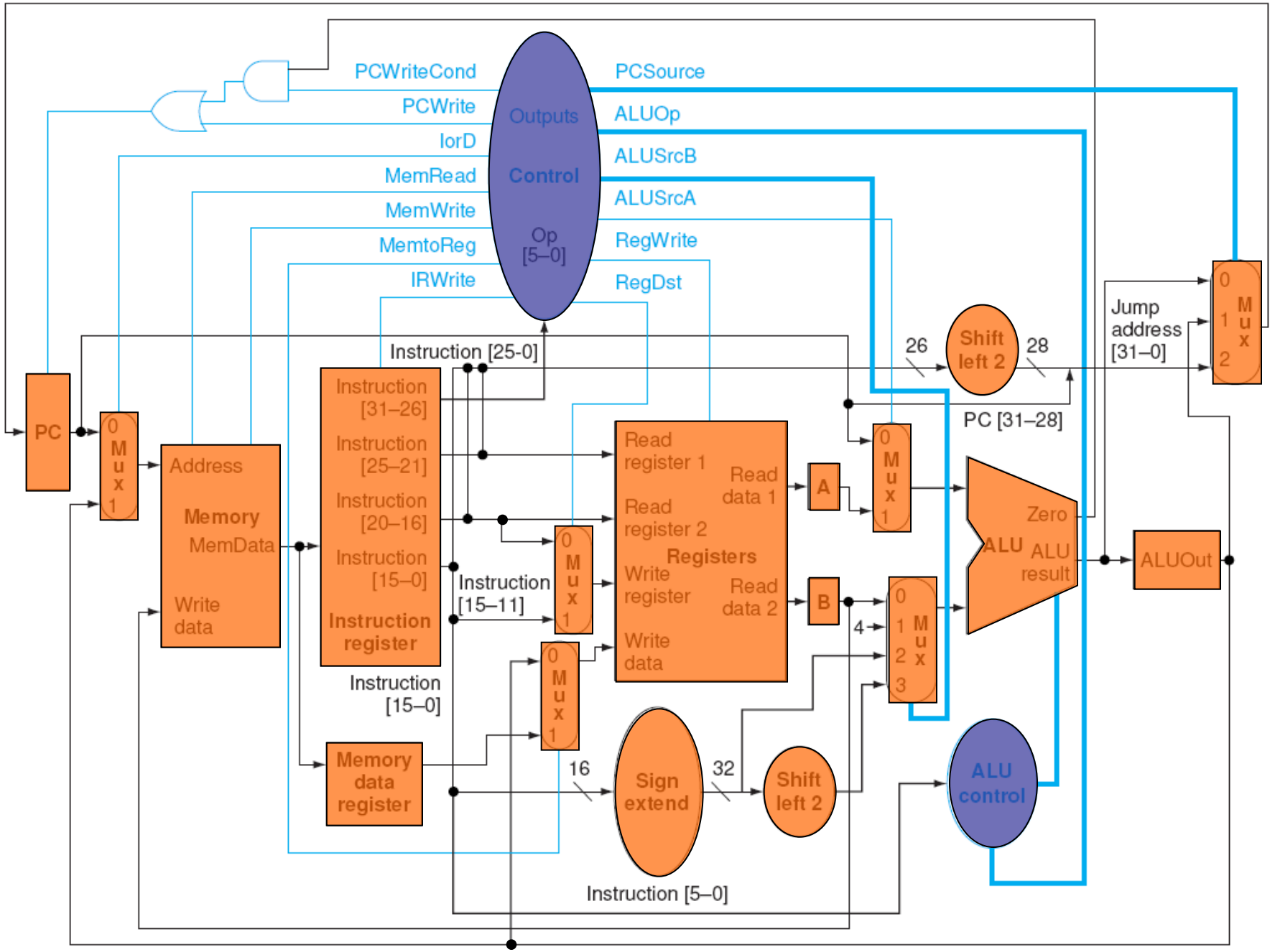
- Difference to 1-cycle implementation
  - Only 1 memory will be used for data and instructions
  - Instead of 1 ALU and 2 adder, only 1 ALU is used
  - Registers are used between datapath‘s components



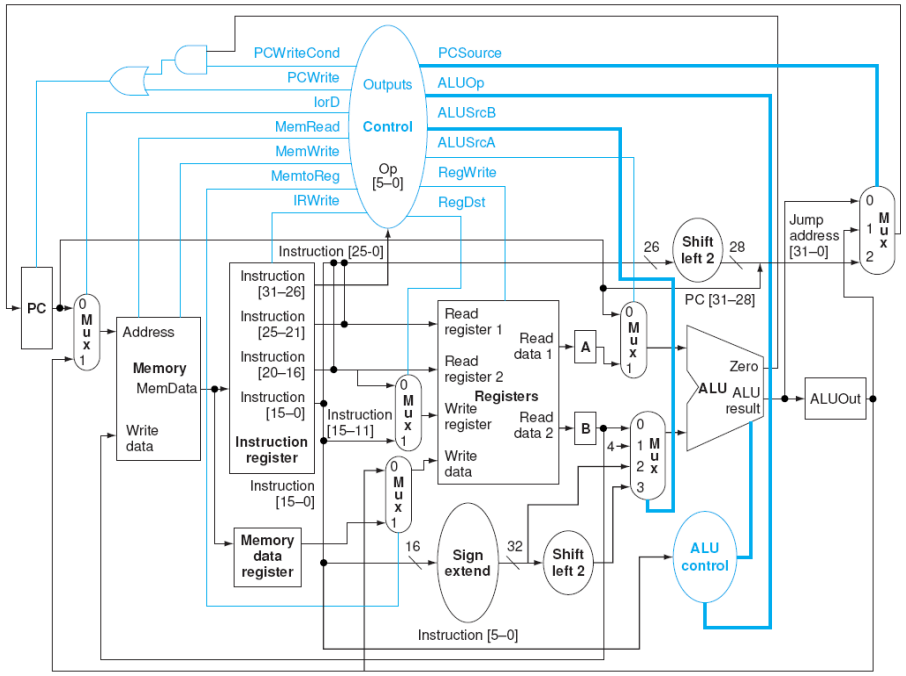
Multi-Cycle Implementation

- Additional Registers
  - Instruction Register (IR): hold the instruction from memory for the whole execution cycle → needs a control signal
  - Memory Data Register (MDR): hold data from the memory, until it is used in the next step
  - Register A, B: hold register values, which will be used in the next step
  - Register ALUOut :hold the result of the ALU-Operation until it's either copied back into memory or register
- Next page:
  Data path and controller for the multi-cycle implementation
  (with extension for conditional branch)

Multi-C


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
Jump
address
[31–0]
M
u
x
PC [31–28]
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
Instruction
[15–0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

Behaviour in every clock cycle



- Step 1: Instruction Fetch

```
IR <= Memory[PC];
PC <= PC + 4;
```

- Step 2: Instruction Decode, Register Fetch

```
A <= Reg[IR[25:21]];
B <= Reg[IR[20:16]];
ALUOut <= PC + (sign-extend(IR[15:0]) << 2);
```

  - The controller interprets the instruction code(opcode, function)
    - The instruction cannot be processed at this stage
  - However: despite unknown instruction, the two input registers rs and rt are read from the register file while the ALU computes the BRANCH address
  - In case this “optimism” was not justified, the results are ignored in the next stept

Behaviour in every clock cycle

- Step 3: Execution, Memory Address Computation, Branch Completion
  - Depending of the instruction, the ALU executes different operations:

Memory address computation

```
ALUOut <= A + sign-extend(IR[15:0]);
```

Arithmetic/Logikc(R-Typ):

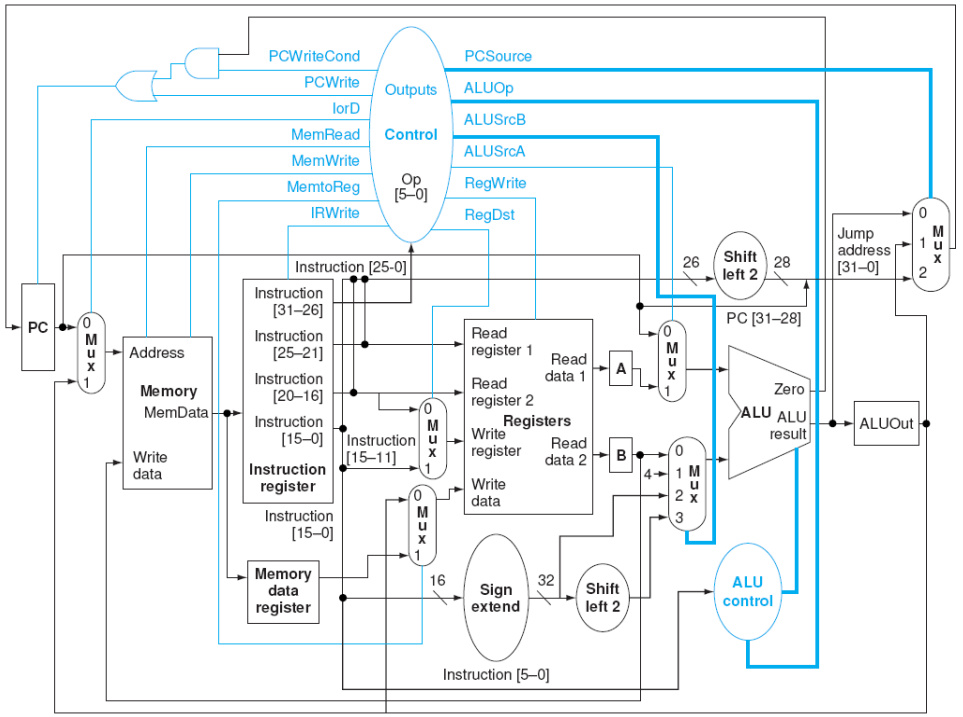
```
ALUout <= A op B;
```

Conditional Branch

```
if (A=B) PC <= ALUOut;
```

Unconditional Branch:

```
PC <= {PC[31:28], IR[25:0], "00"};
```



Behaviour in every clock cycle

- Step 4: Memory Access, R-type Instruction Completion
  - The data transfer instruction accesses the memory to store the result

Memory access:

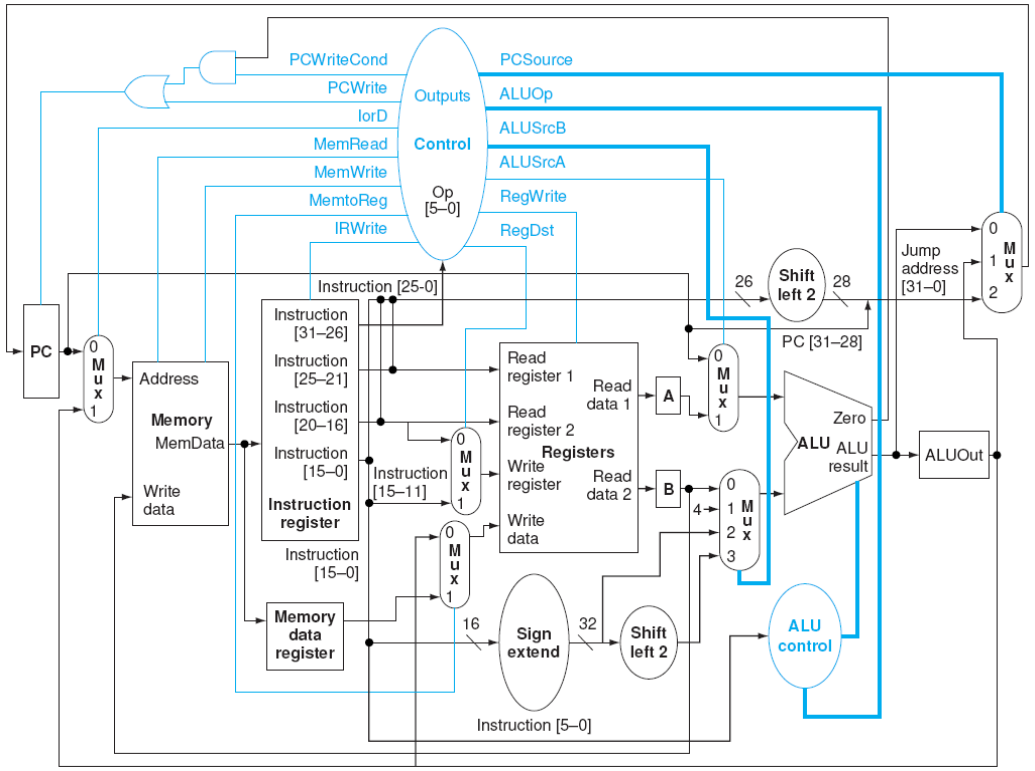
```
MDR <= Memory[ALUout]; or
Memory[ALUOut] <= B;
```

Arithmetic/Logic (R-Typ):

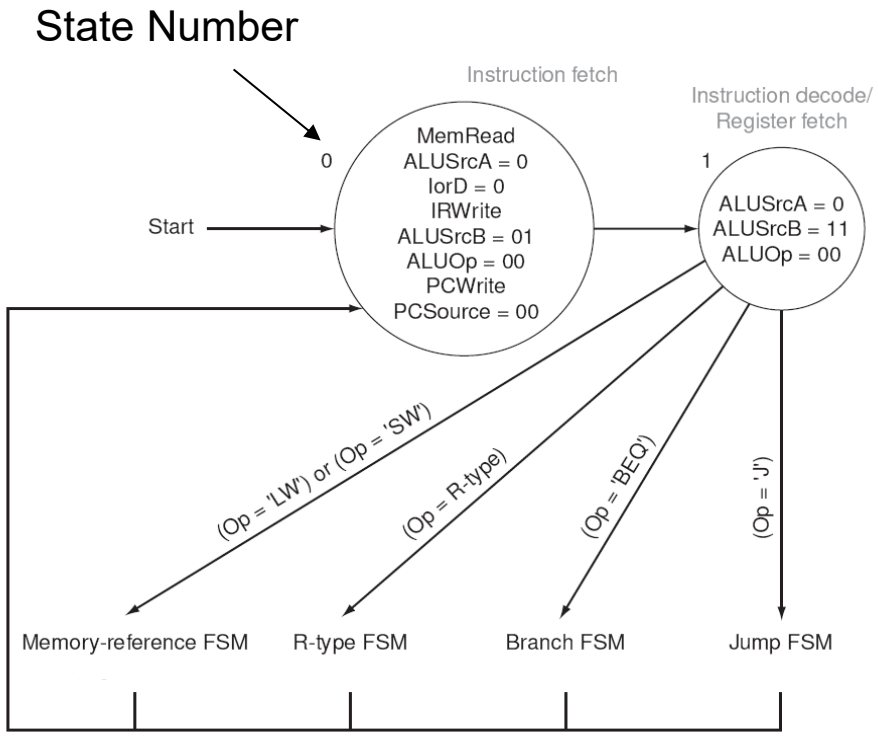
```
Reg[IR[15:11]] <= ALUOut;
```

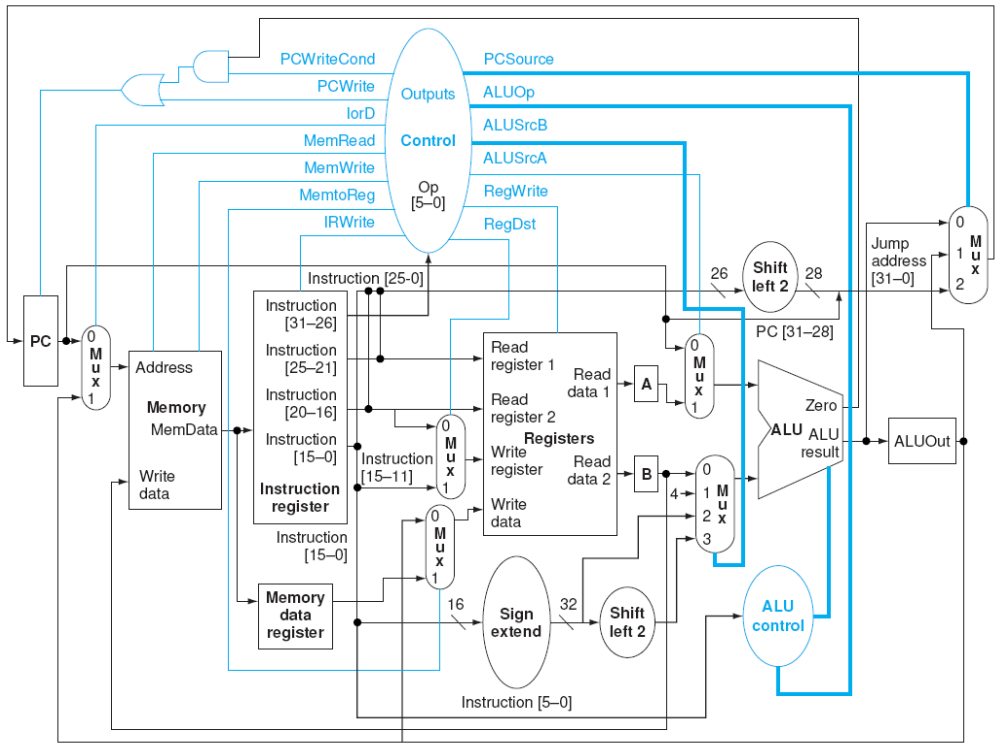
- Step 5: Memory Read Completion

```
Reg[IR[20:16]] <= MDR;
```

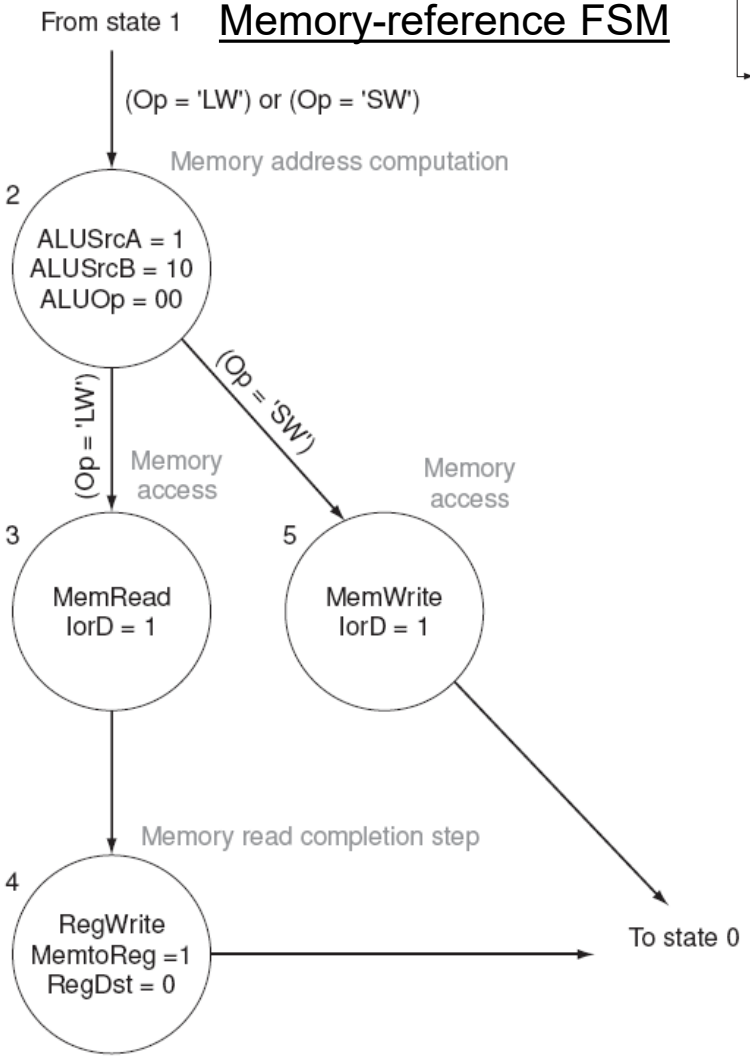


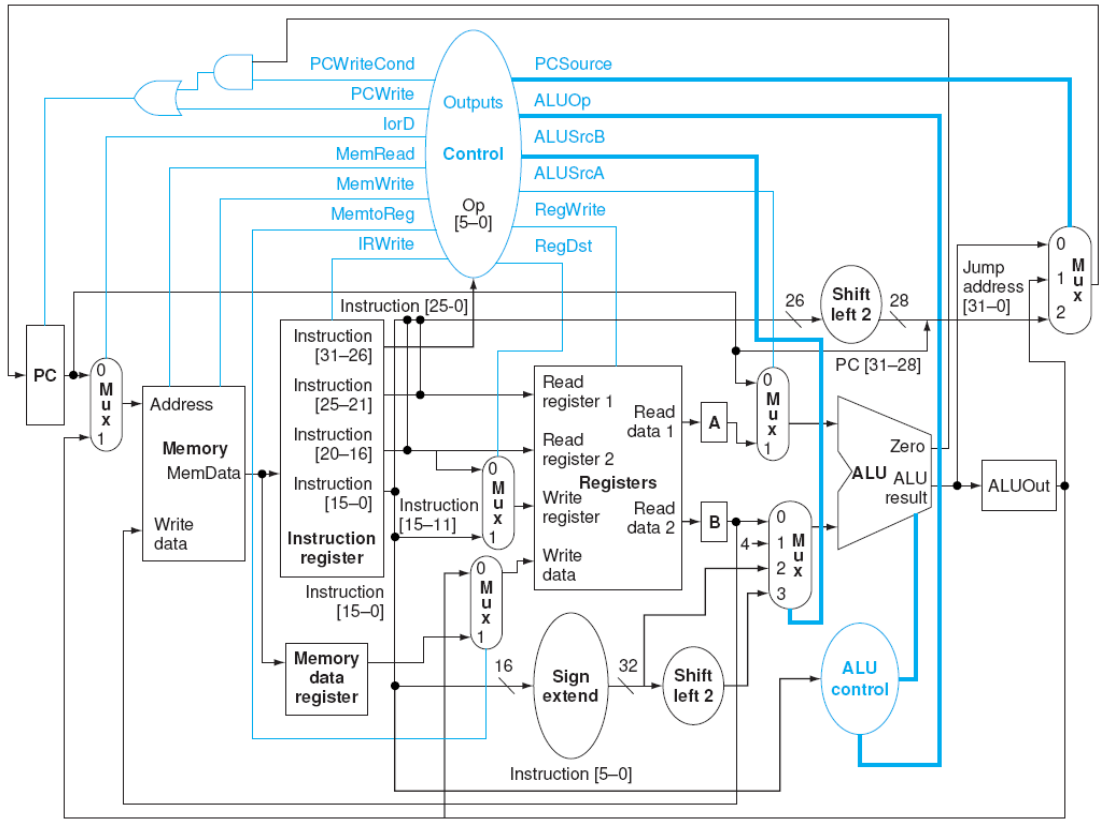
Controller's Specification



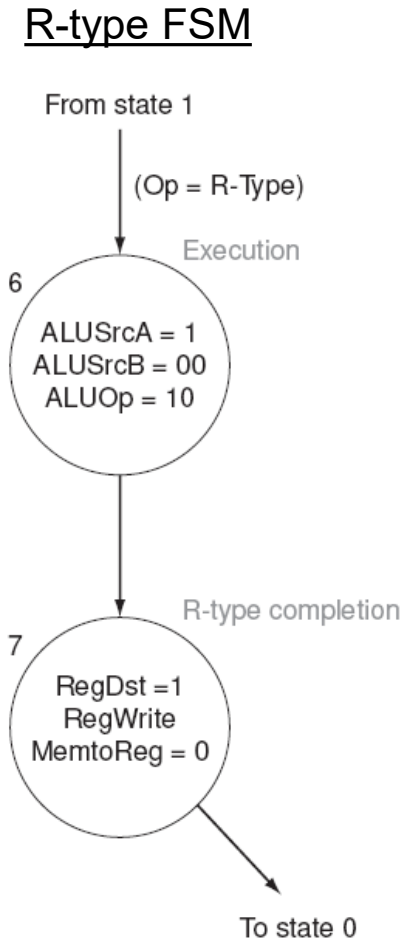


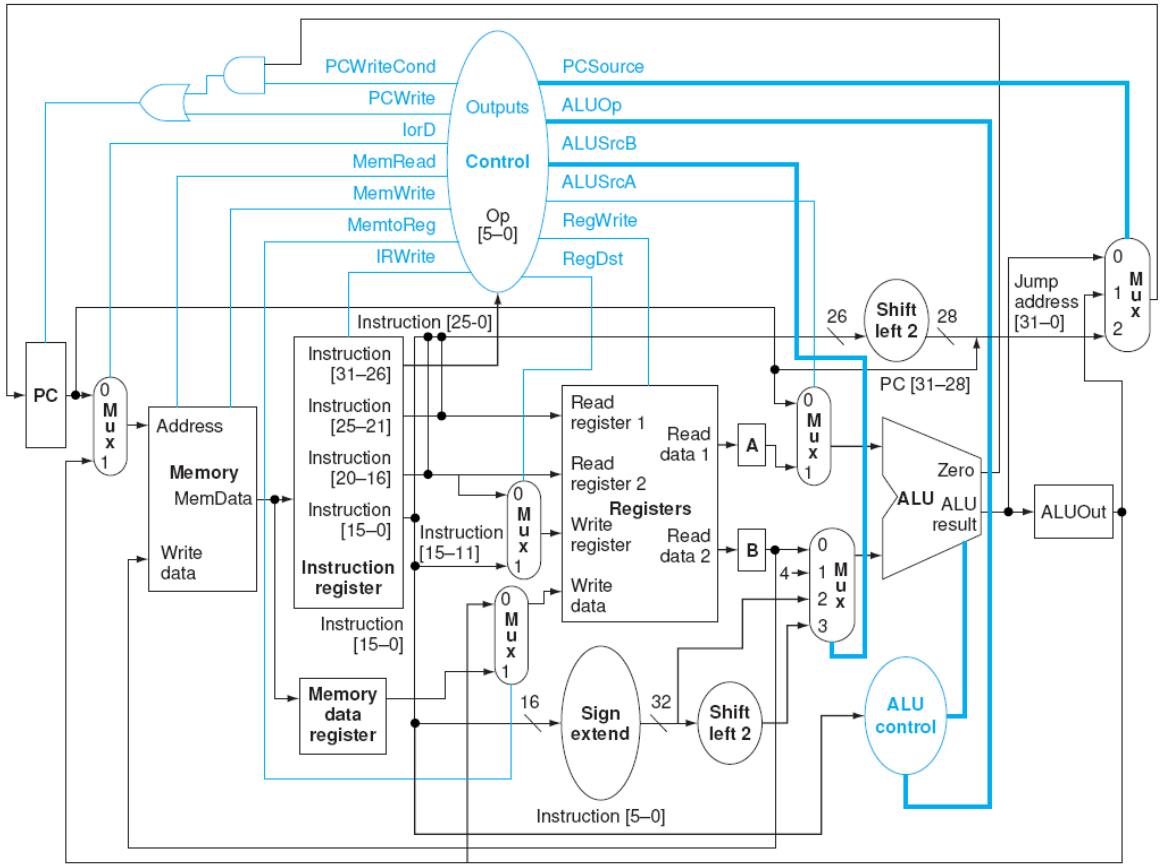
Controller's Specificatio



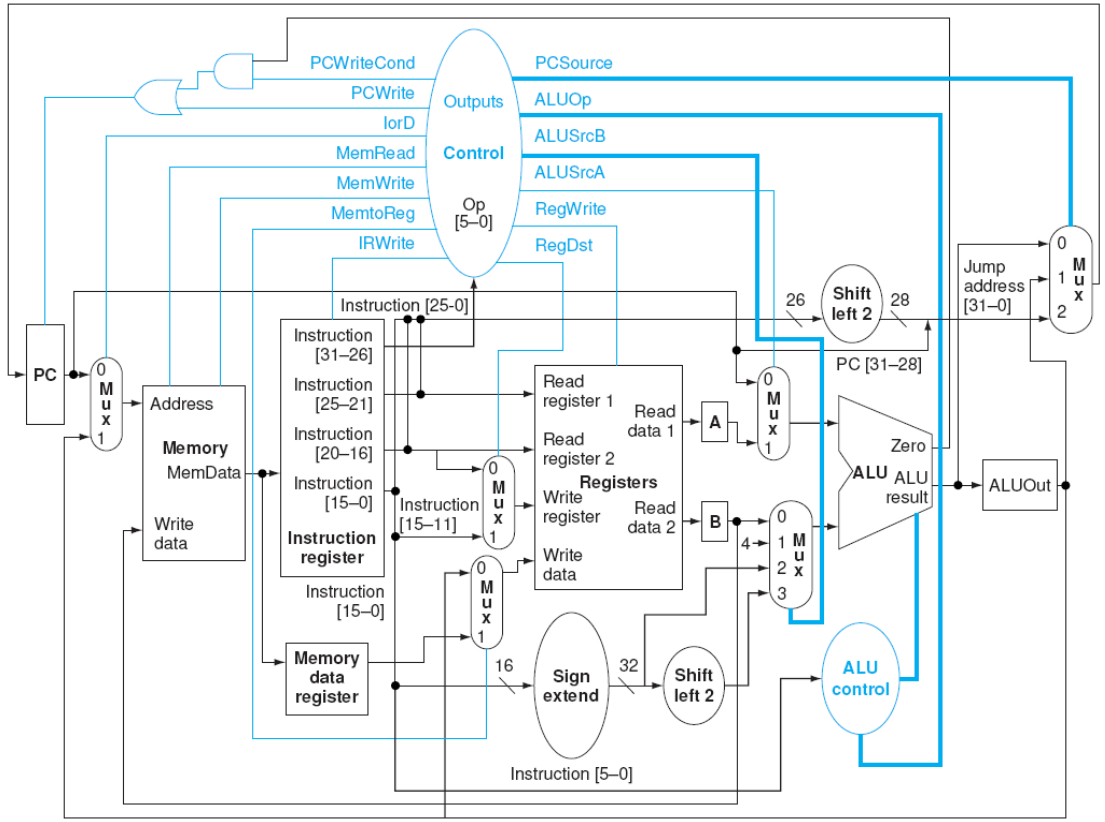


Controller's Specification

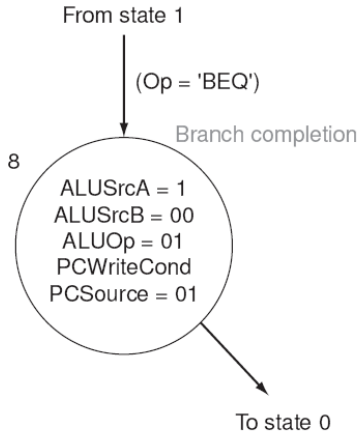




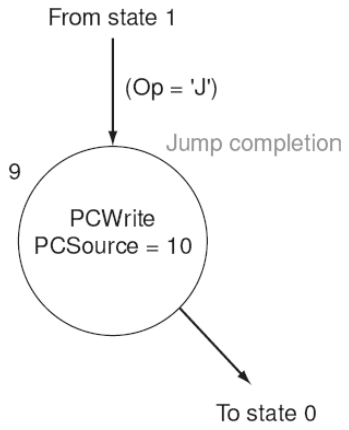
Controller's Specification



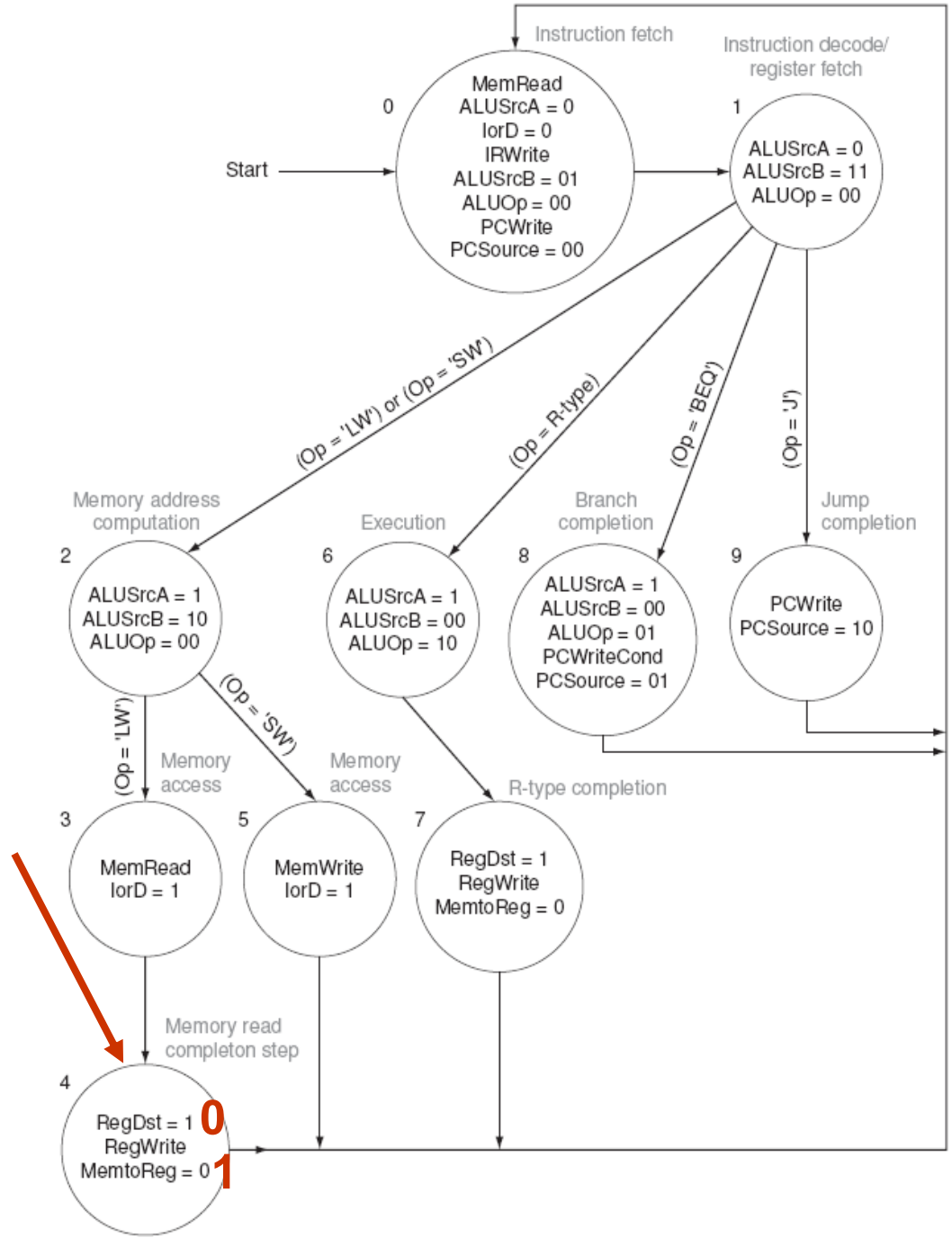
Branch FSM

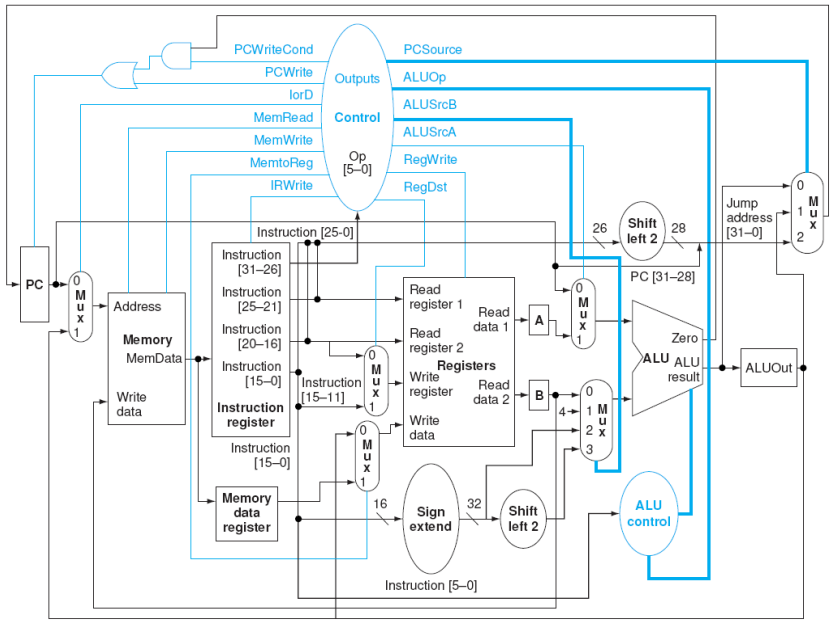


Jump FSM

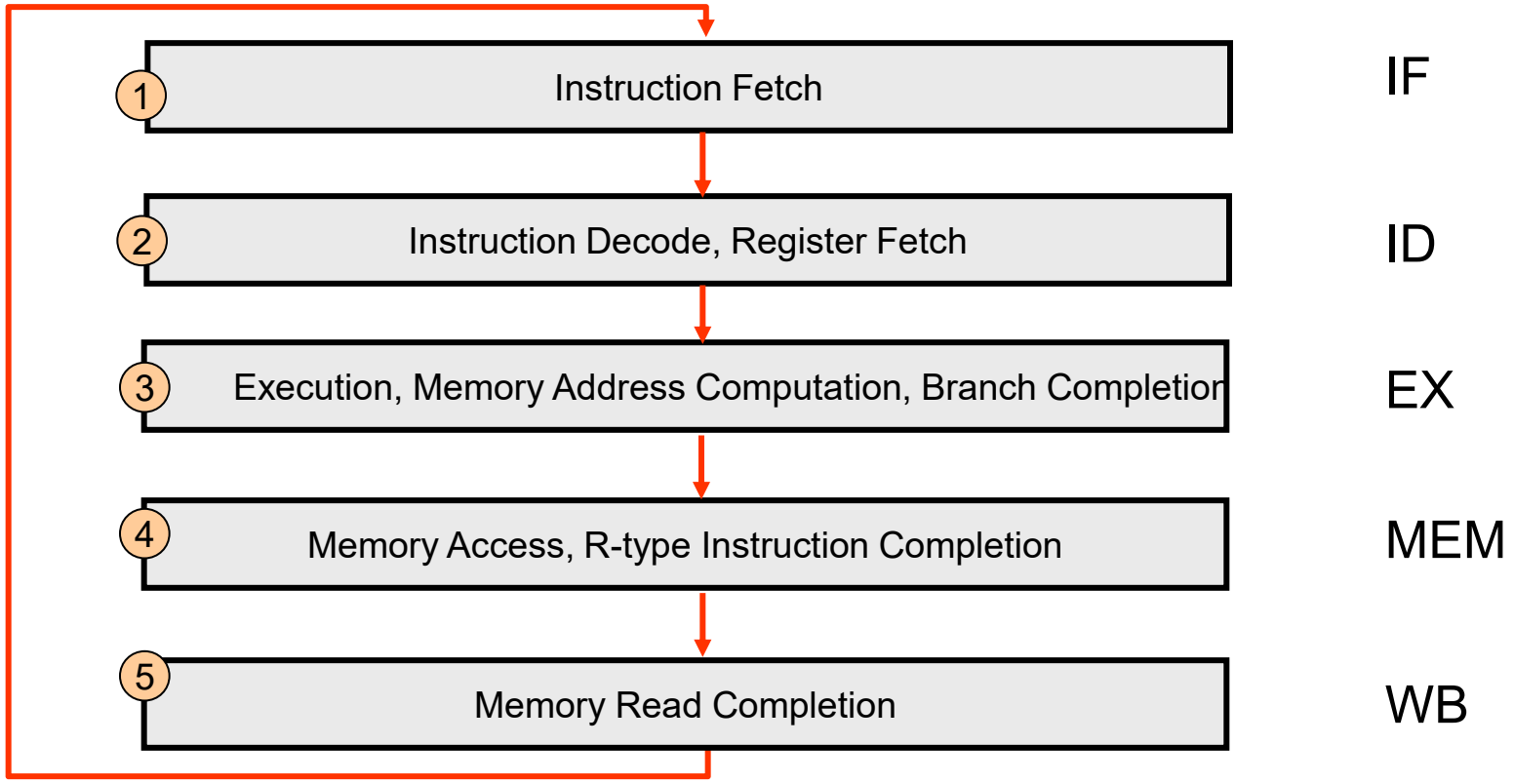


Auto


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Start
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 1
RegWrite
MemtoReg = 0
0
1

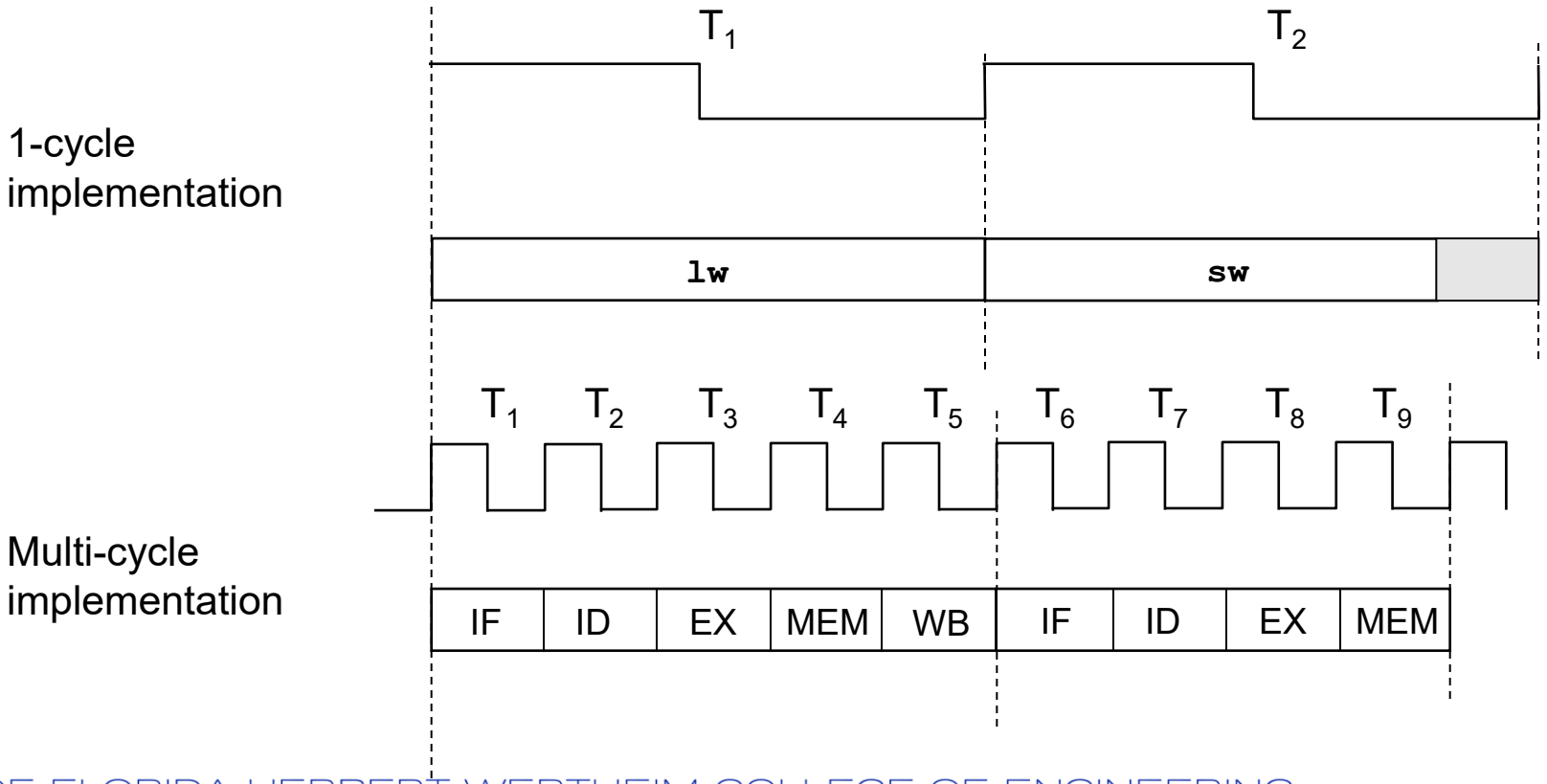

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
Instruction
[15–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

Steps of 1 Instruction Cycle



WB = write back

Comparison 1-cycle vs multi-cycle implementation




UF
Herbert Wertheim
College of Engineering
UNIVERSITY of FLORIDA
LEADING THE CHARGE, CHARGING AHEAD