Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 4: Single-Cycle CPU Design

http://users.utcluj.ro/~negrum/

Processor Design Phases

1. Analyze instruction set → data-path requirements
   – Write the micro-operation sequences for the target ISA
   – RTL statements specify the data-path components and their interconnection
2. Select a set of data-path components and establish clocking methodology
   – Define when storage or state elements can be read and when they can be written, e.g. clock edge-triggered
   – Find the worst-time propagation delay in the data-path to determine the data-path clock cycle (CPU clock cycle)
3. Assemble data-path meeting the requirements
   – Create an initial data-path (i.e. registers, ALU, memories)
   – Establish the connectivity requirements
   – Whenever *multiple sources* are connected to a *single input* (or destination), a multiplexer of appropriate size is added
   – Complete the micro-operation sequences for all remaining instructions adding data path components + connections/multiplexers as needed

Processor Design Phases

4. Identify and define the function of all control points or signals needed by the data-path
   - For each instruction from the target ISA identify the values of the control signals that affect the register transfers
5. Assemble the control logic
   - Design the control unit based on the identified control signals
   - 3 main types of control unit
     - Combinational logic → single cycle CPU (Any instruction completed in one cycle)
     - Hard-Wired: Finite-state machine implementation
     - Micro-programmed

- MIPS Single-Cycle CPU Design adapted from [1]

Single-Cycle CPU Design – Step 1

- Design Step 1: MIPS-Lite Subset
  - Select a number of representative target instructions

Instruction	RTL Abstract	Program Counter
add $rd, $rs, $rt	RF[rd] ← RF[rs] + RF[rt]	PC ← PC + 4
sub $rd, $rs, $rt	RF[rd] ← RF[rs] – RF[rt]	PC ← PC + 4
ori $rt, $rs, imm	RF[rt] ← RF[rs] | Z_Ext(imm)	PC ← PC + 4
lw $rt, imm($rs)	RF[rt] ← M[RF[rs] + S_Ext(imm)]	PC ← PC + 4
sw $rt, imm($rs)	M[RF[rs] + S_Ext(imm)] ← RF[rt]	PC ← PC + 4
beq $rt, $rs, imm	If(RF[rs] == RF[rt]) then	PC ← PC + 4 + S_Ext(imm) <<2
	else	PC ← PC + 4

- RTL Abstract defines the behavior of each instruction
- Remember the instruction execution cycle (previous lecture)
  - IF, ID/OF, EX, MEM, WB
  - IF, ID and OF are common for all instructions

Single-Cycle CPU Design – Step 1

- R-type Instructions
  - Basic operation: RF[rd] ← RF[rs] op RF[rt]
  - Next instruction PC: PC ← PC + 4
  - OPCODE is always Zero for R-type Instructions

31 … 26	25 … 21	20 … 16	15 … 11	10 … 6	5 … 0
SPECIAL 000000	rs	rt	rd	0 00000	ADD 100000
6	5	5	5	5	6

- ADD $rd, $rs, $rt
  - RF[rd] ← RF[rs] + RF[rt]
  - PC ← PC + 4
  - Add signed 32-bit numbers.
  - Exception on OVERFLOW
  - Addressing Modes
    - Register direct

Necessary Resources	
IF	PC, Instr. Memory, Adder
ID/OF	Register File, Main Control Unit
EX	ALU, ALU Control Unit
MEM	No operation
WB	Register File

Single-Cycle CPU Design – Step 1

- I-type Instructions: Load Word – LW
  - Load a word (32 bits) from the Data Memory into a Register

31 … 26	25 … 21	20 … 16	15 … 0
LW 100011	rs	rt	address / immediate
6	5	5	16

- lw $rt, imm($rs)
  - RF[rt] ← M[RF[rs] + S_Ext(imm)]
  - PC ← PC + 4
  - Addressing Modes
    - Register Direct
    - Base addressing

Necessary Resources	
IF	PC, Instr. Memory, Adder
ID/OF	Register File, Main Control Unit, Extender
EX	ALU, ALU Control Unit
MEM	Data Memory
WB	Register File

MUX

Single-Cycle CPU Design – Step 1

- I-type Instructions: Store Word – SW
  - Store a word (32-bits) from the Register File in the Data Memory

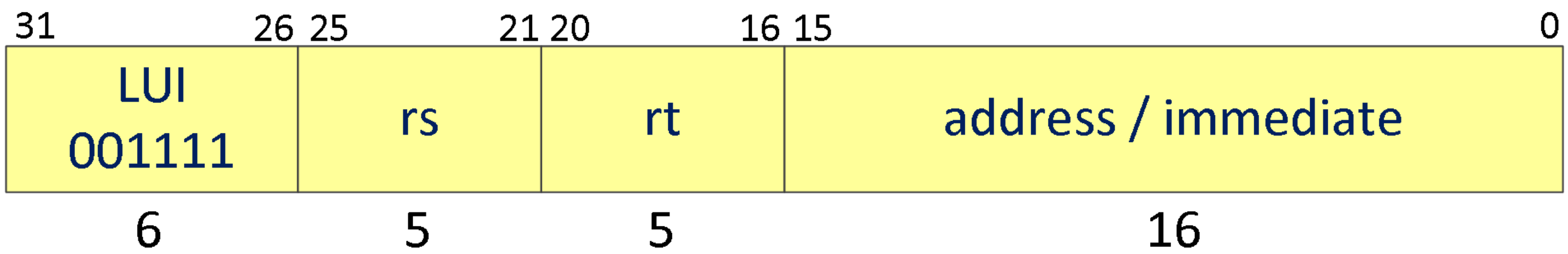
31 – 26	25 – 21	20 – 16	15 – 0
SW 101011	rs	rt	address / immediate
6	5	5	16

- sw $rt, imm($rs)
  - M[RF[rs] + S_Ext(imm)] ← RF[rt]
  - PC ← PC + 4
  - Addressing Modes
    - Register Direct
    - Base addressing

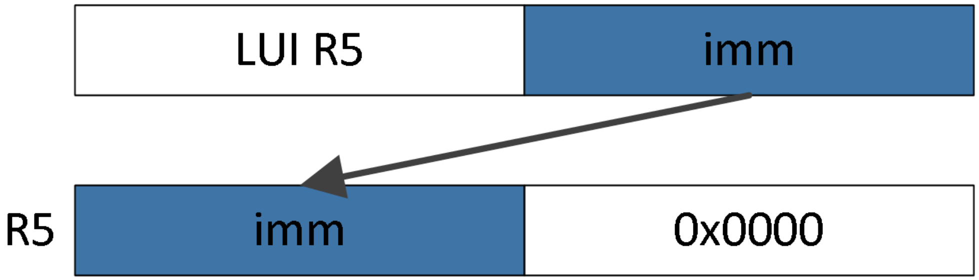
Necessary Resources	
IF	PC, Instr. Memory, Adder
ID/OF	Register File, Main Control Unit, Extender
EX	ALU, ALU Control Unit
MEM	Data Memory
WB	No operation

Single-Cycle CPU Design – Step 1

- I-type Instructions: Load Upper Immediate – LUI
  - Load a constant in high part of a word



- lui $rt, imm
  - RF[rt] ← imm || 0x0000
  - PC ← PC + 4
  - Addressing Modes
    - Register direct
  - Used to form 32 bits constants with ORI
  - Additional Resources: shifter implemented in the ALU



Single-Cycle CPU Design – Step 1

- I-type Instructions: Branch on Equal – BEQ
  - Compare two registers, then perform a conditional jump relative to the PC

31 … 26	25 … 21	20 … 16	15 … 0
BEQ 000100	rs	rt	address / immediate
6	5	5	16

- beq $rt, $rs, imm
  - If(RF[rs] == RF[rt]) → PC ← PC + 4 + S_Ext(imm) <<2 else PC ← PC + 4
  - Addressing Modes
    - PC-relative addressing
  - If condition is not true
    - Sequential execution, + 4
  - If condition is true
    - Jump, PC + 4 + S_Ext(imm)<<2

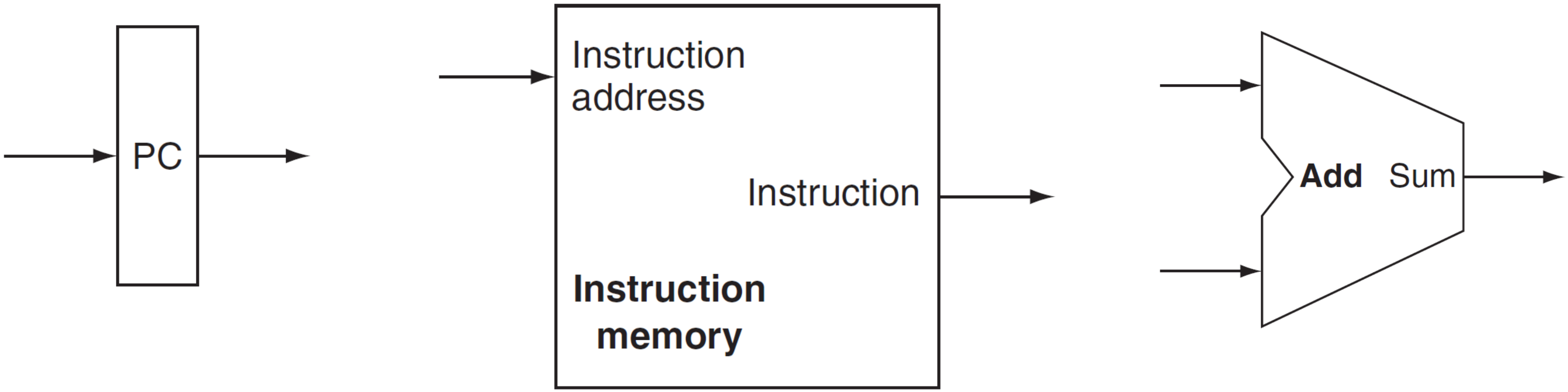
Necessary Resources	
IF	PC, Instr. Memory, Adder, **Adder, MUX**
ID/OF	Register File, Main Control Unit, Extender
EX	ALU, ALU Control Unit
MEM	No operation
WB	No operation

Single-Cycle CPU Design – Step 1

- Needed Resources (so far)
  - PC – Program Counter
  - Memories
    - Instruction and Data Memory
  - Register File (32 x 32 bits)
    - Read R[rs], Read R[rt]
    - Write R[rd] or R[rt] ➡ **MUX**
  - Sign / Zero Extender for address / immediate field
  - Shift Left 2
  - ALU – Arithmetic Logic Unit ➡ **MUX**
    - Arithmetic or Logical operations with two registers
    - Arithmetic or Logical operations with one register and an extended immediate value
  - Add PC with 4 or with 4 + Sign Extended Immediate << 2 for next instructions address (PC) computation ➡ **MUX**

Single-Cycle CPU Design – Step 2

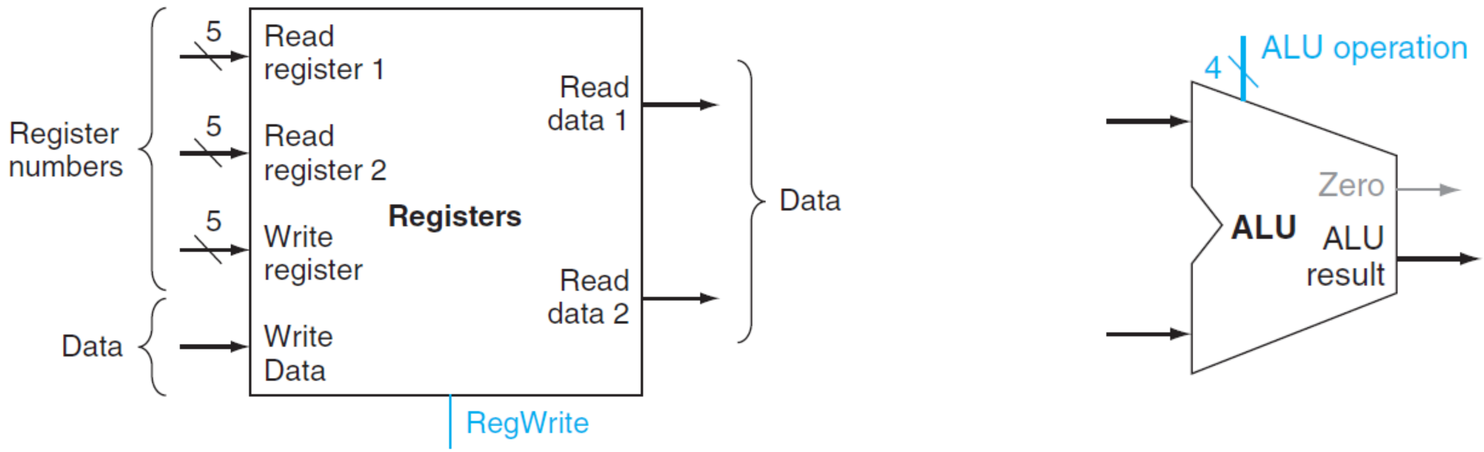
- Design Step 2: Data-Path Components



[1]

- Program Counter – PC
  - 32-bit positive edge triggered D flip-flop
- Instruction Memory (ideal ROM model)
  - One input bus: Instruction address
  - One output bus: Instruction
  - Memory word is selected by Instruction address, no control signals
- Adder
  - 32-bit Ripple Carry Adder to form the next instruction address

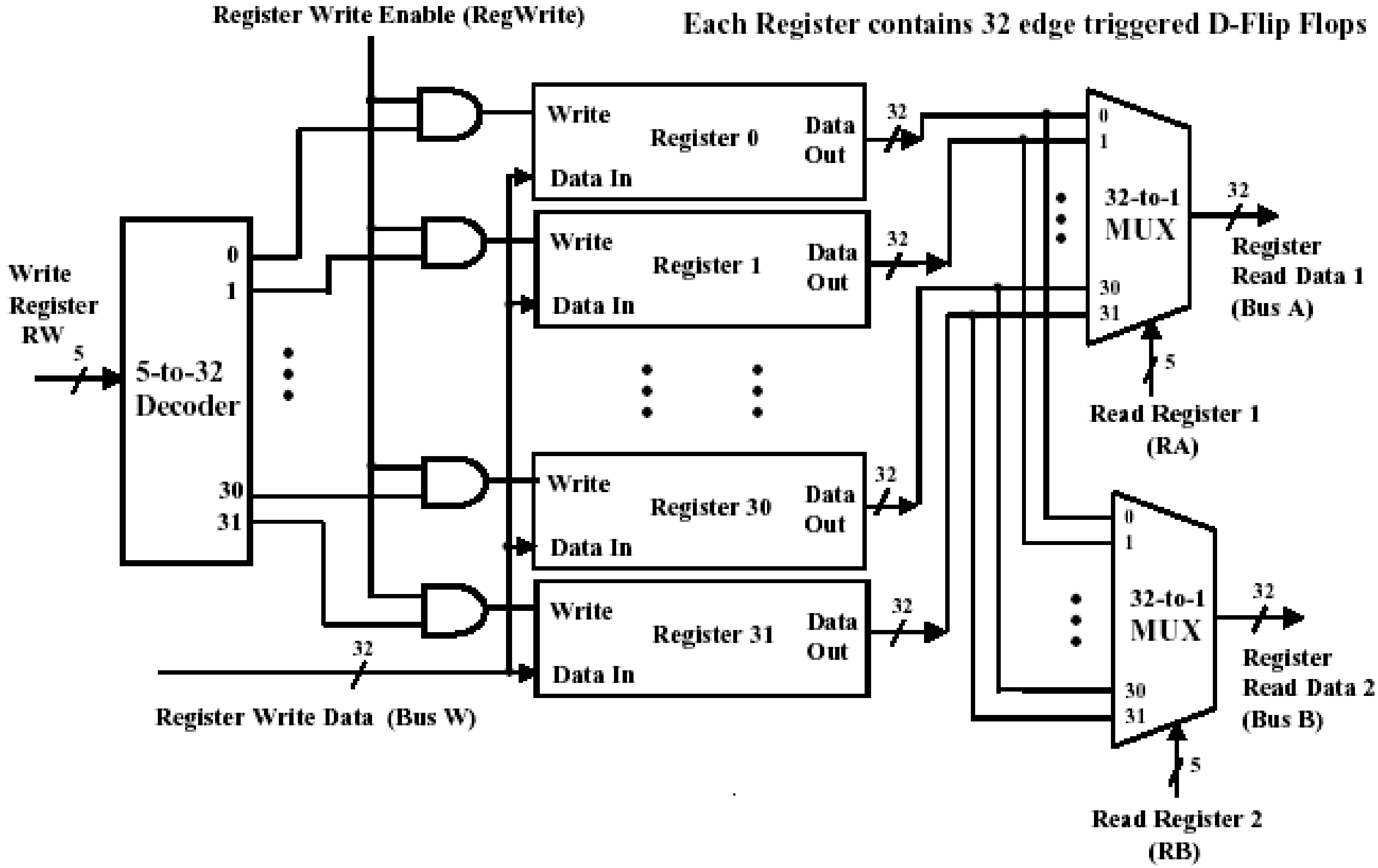
Single-Cycle CPU Design – Step 2



[1]

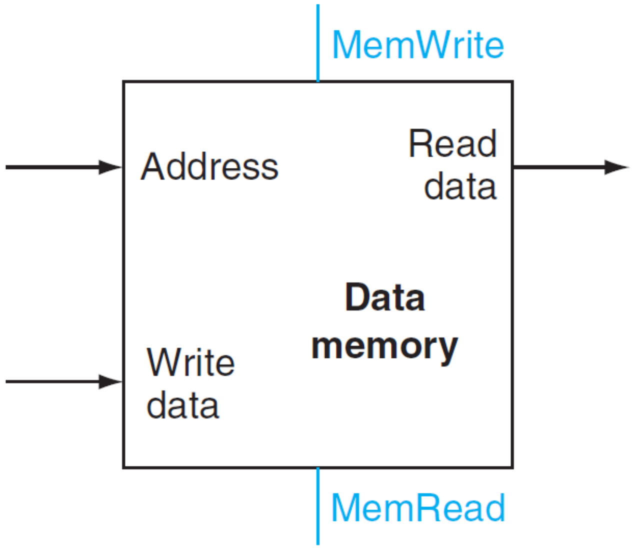
- Register File 32x32-bits
  - Built using D flip-flops (didactic model), SRAM in real machines
  - Two 32-bit data outputs: Read data 1 and Read data 2
  - One 32-bit data input: Write data
  - Multi-access: 2 asynchronous Reads + 1 edge triggered Write in the same clock period
  - Read register 1 selects the register to put on Read data 1 output
  - Read register 2 selects the register to put on Read data 2 output
  - Write register selects the register to be written by Write Data when RegWrite is asserted
  - During read operation, Registers behaves as a combinational logic block
- Arithmetic Logic Unit – ALU
  - Designed according to ISA requirements

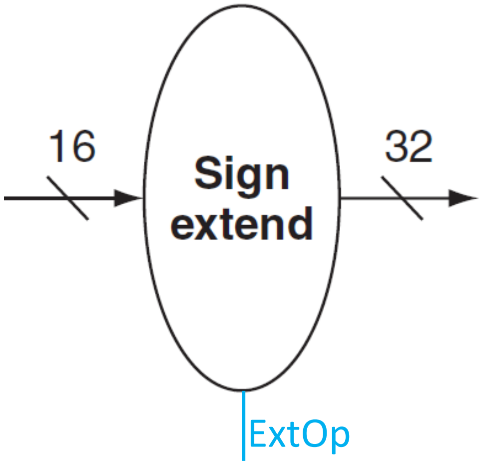
Single-Cycle CPU Design – Step 2



Register File Implementation

Single-Cycle CPU Design – Step 2





[1]

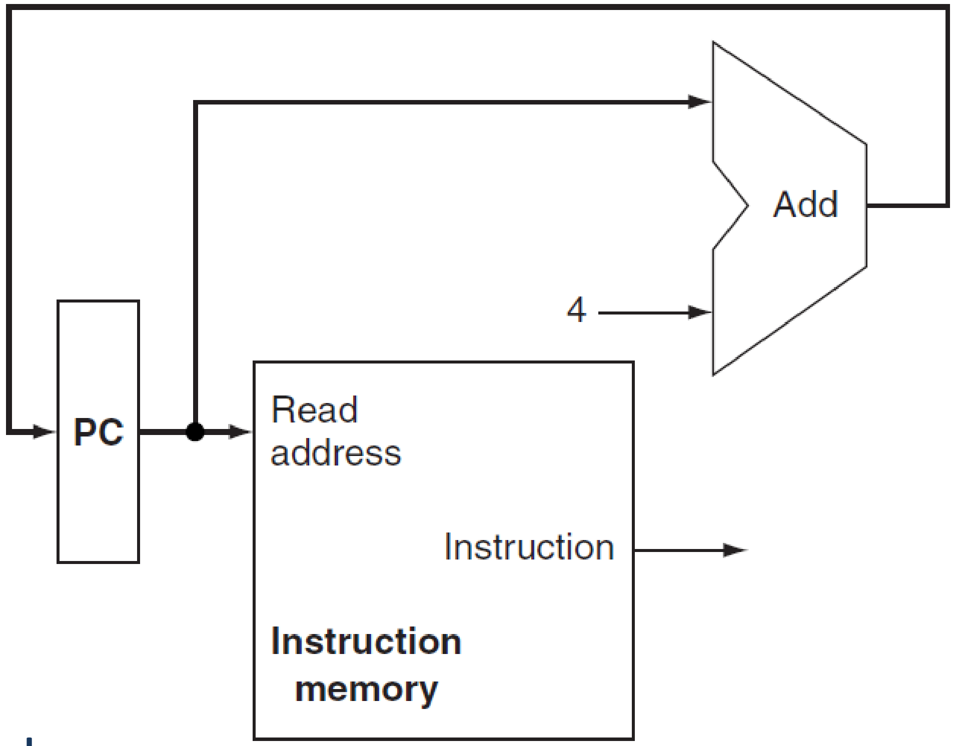
- Data Memory (ideal SRAM model)
  - Two input buses: Address and Write data
  - One output bus: Read data
  - Two control signals: MemRead and MemWrite
- Sign Extension Unit
  - The control signal will only be added later
  - ExtOp = 1 → Sign Extender
  - ExtOp = 0 → Zero Extender

Single-Cycle CPU Design – Step 2

- Design Step 2: Clocking Methodology
  - Clocking methodology defines when signals can be read and written
  - Determines when data is valid and stable relative to the clock
- Clocking alternatives
  - Falling edge triggered system
  - Rising edge triggered system
  - Two phase clocking
- All storage elements (e.g. Flip-Flops, Registers, Data Memory) writes are triggered by the same clock edge.
  - Usually, State elements are written on every clock cycle
  - If not, we need an explicit write control signal.
  - Write only when both the write control is asserted and the clock edge occurs

Single-Cycle CPU Design – Step 3

- Design Step 3: Assemble Data-Path – Single-Cycle

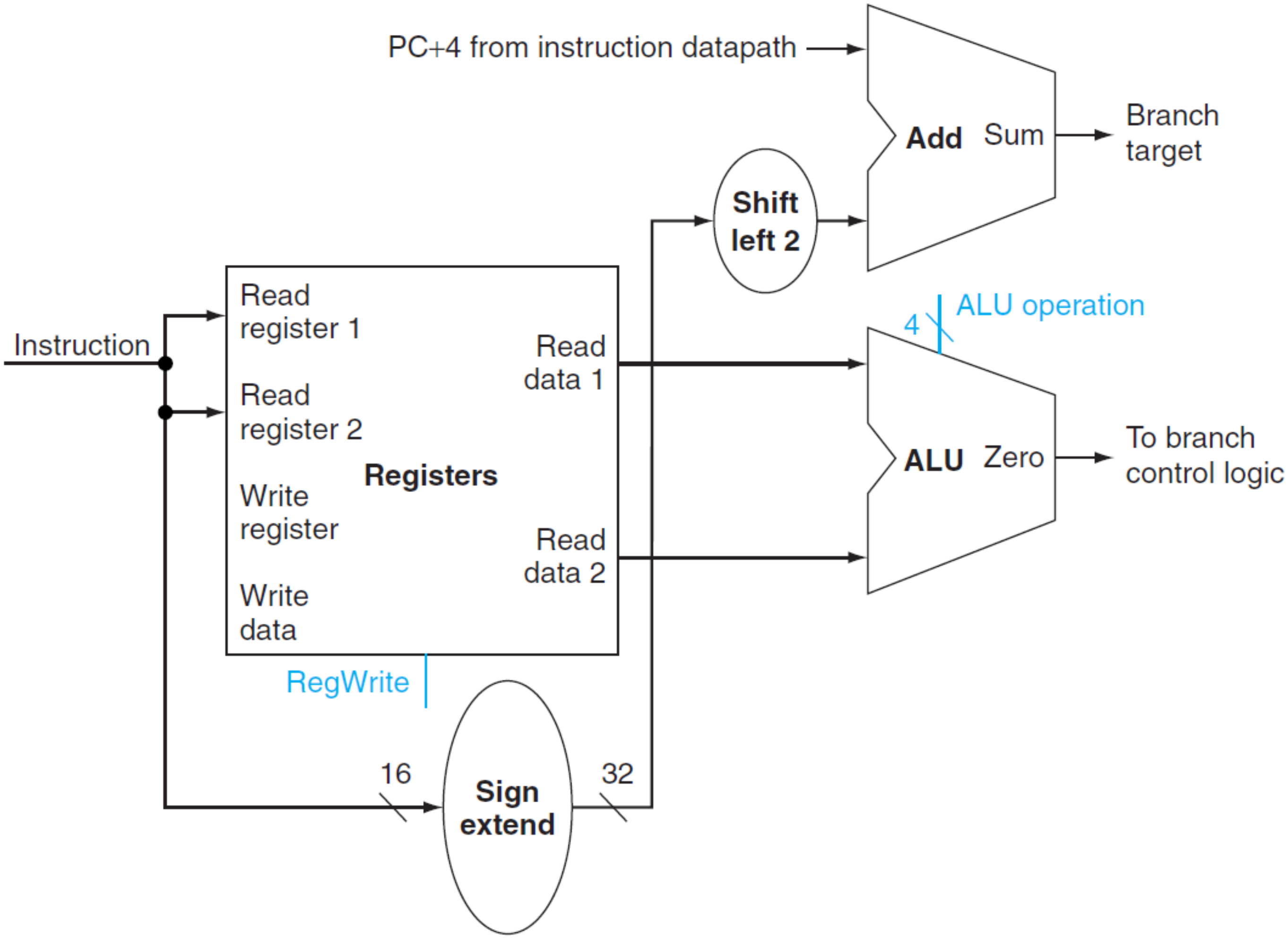


[1]

Instruction Fetch

- 32-bit Program Counter, 32-bit Adder and instruction Memory
- Instruction ← IM[PC]; PC ← PC + 4

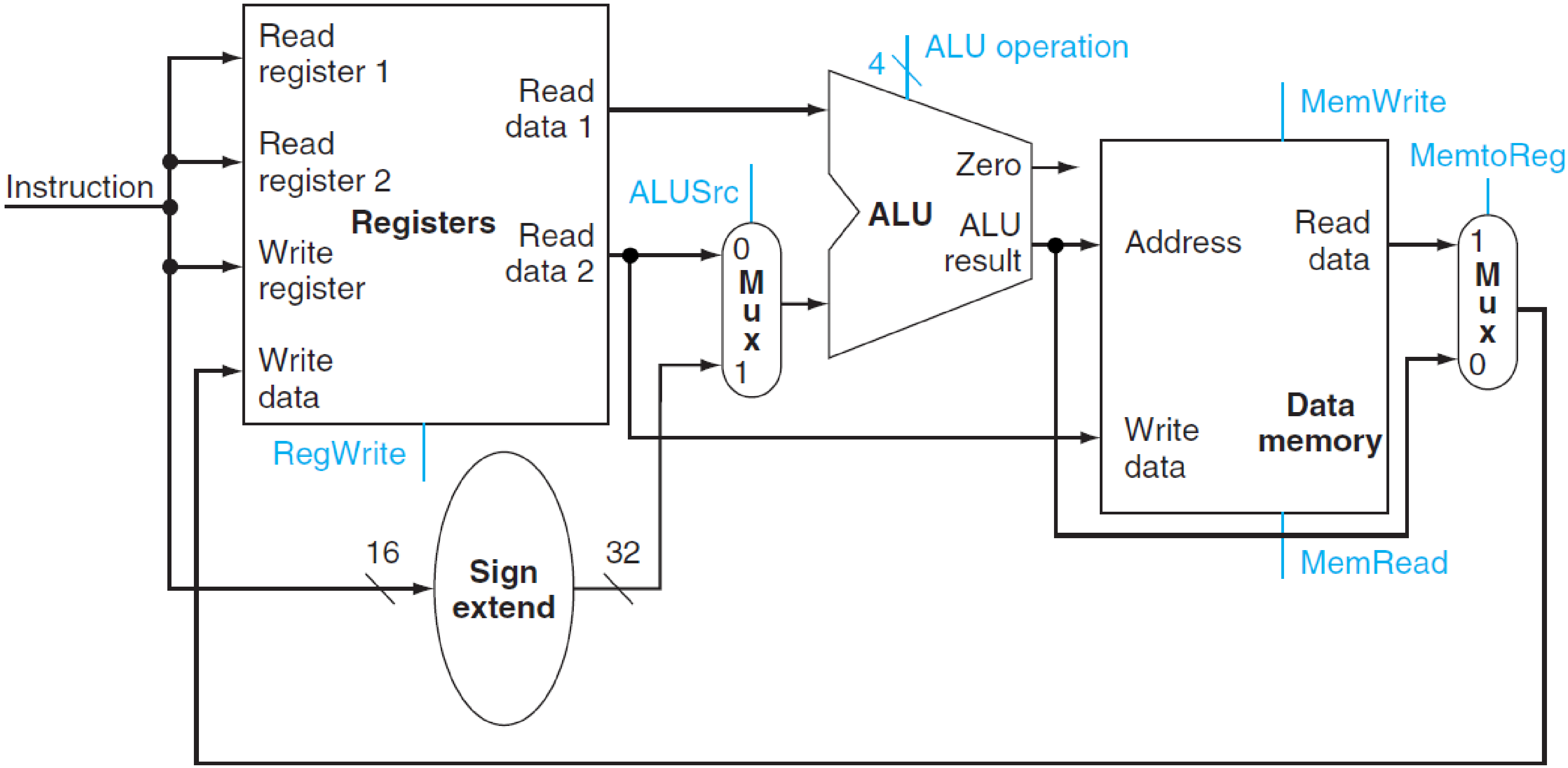
Single-Cycle CPU Design – Step 3



[1]

Branch Instruction: If(RF[rs] == RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2
else PC ← PC + 4

Single-Cycle CPU Design – Step 3

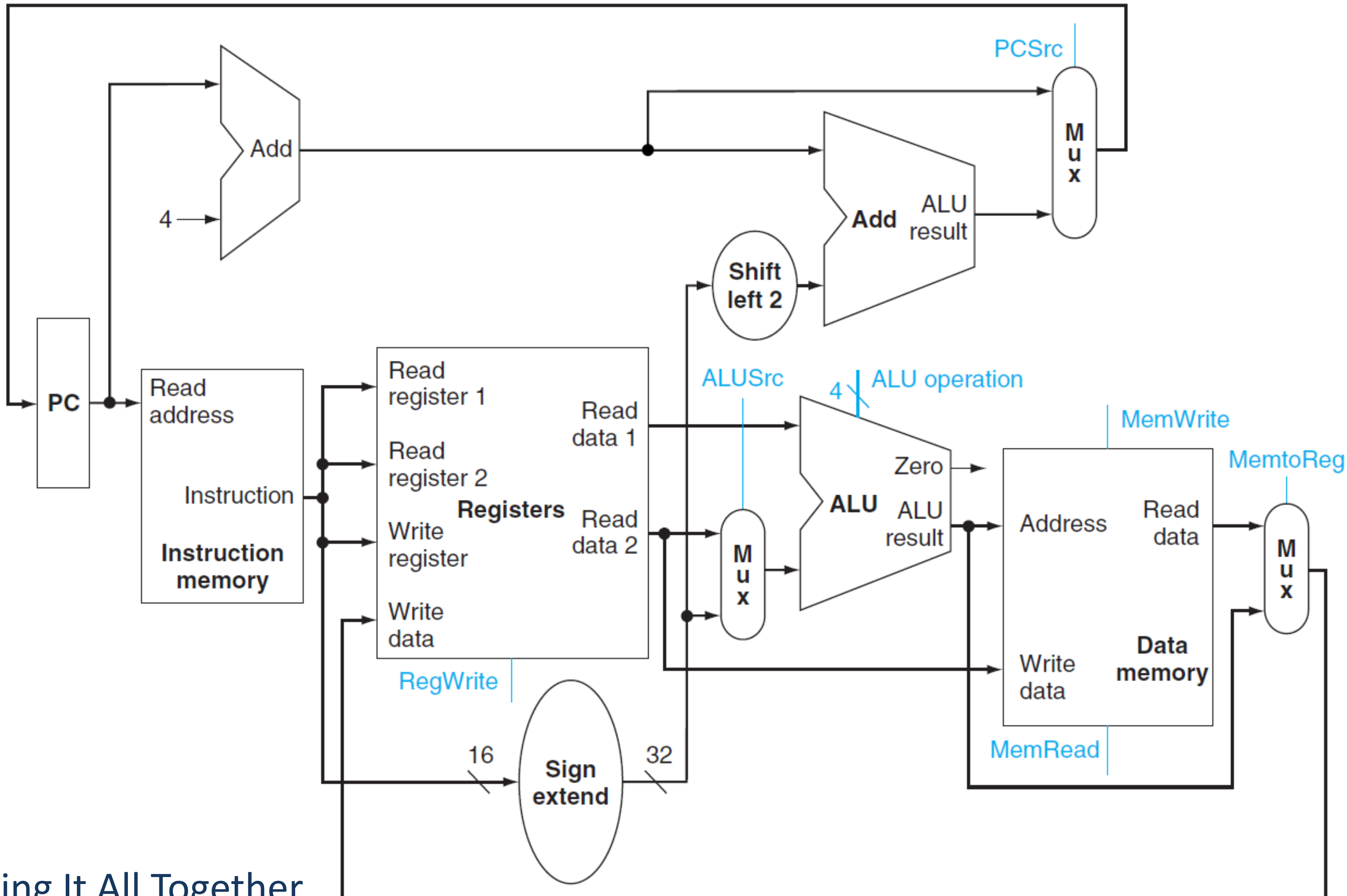


[1]

R-type Instructions:	RF[rd] ← RF[rs] op RF[rt]
I-type Instruction – Load:	RF[rt] ← M[RF[rs] + S_Ext(imm)]
I-type Instruction – Store:	M[RF[rs] + S_Ext(imm)] ← RF[rt]

.....

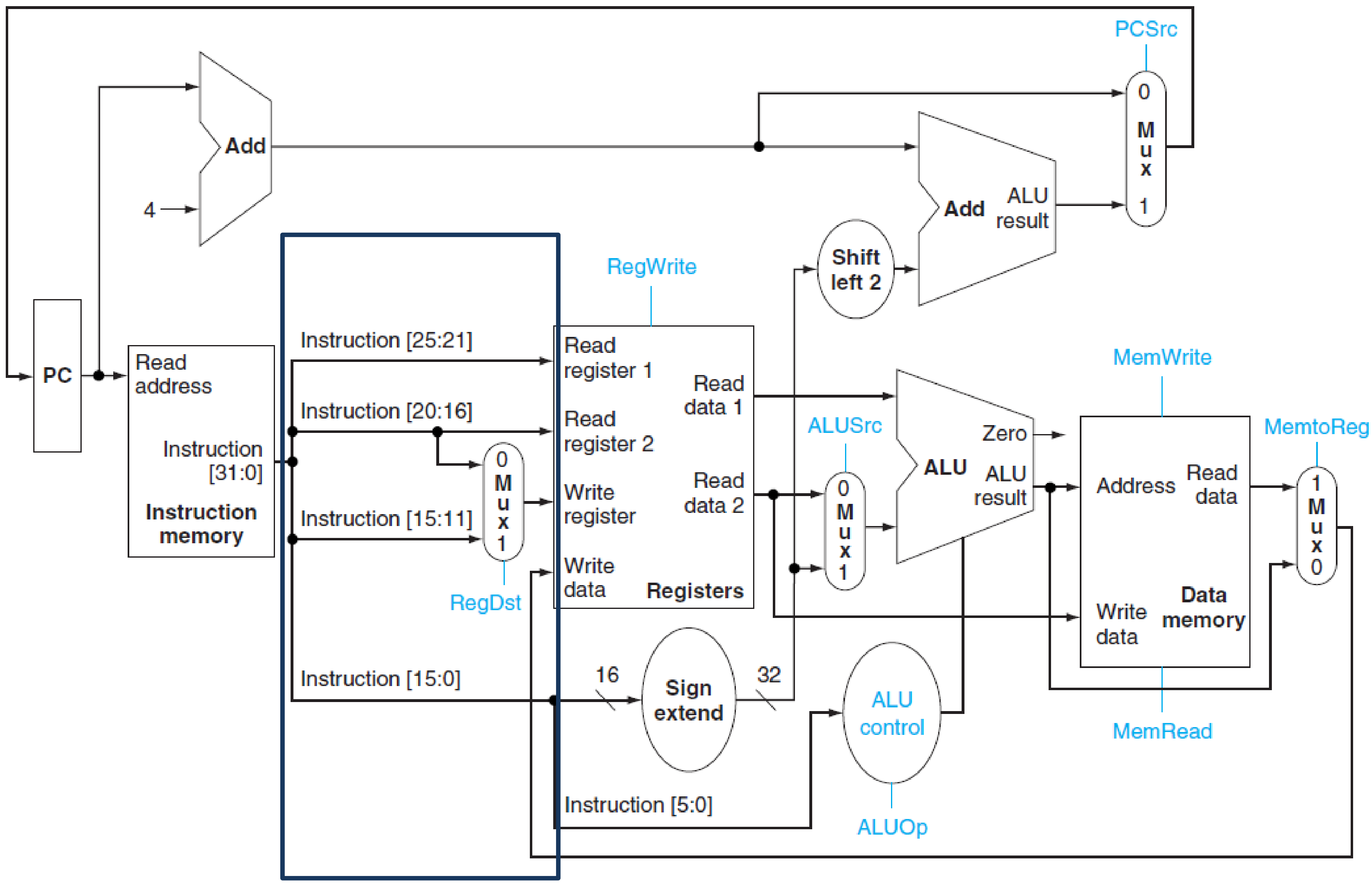
Single-Cycle CPU Design – Step 3



Putting It All Together

[1]

Single-Cycle CPU Design – Step 3

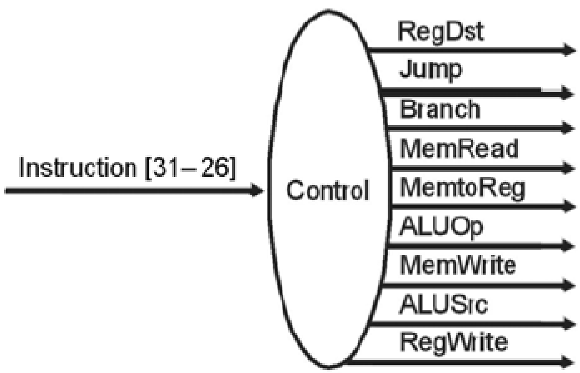


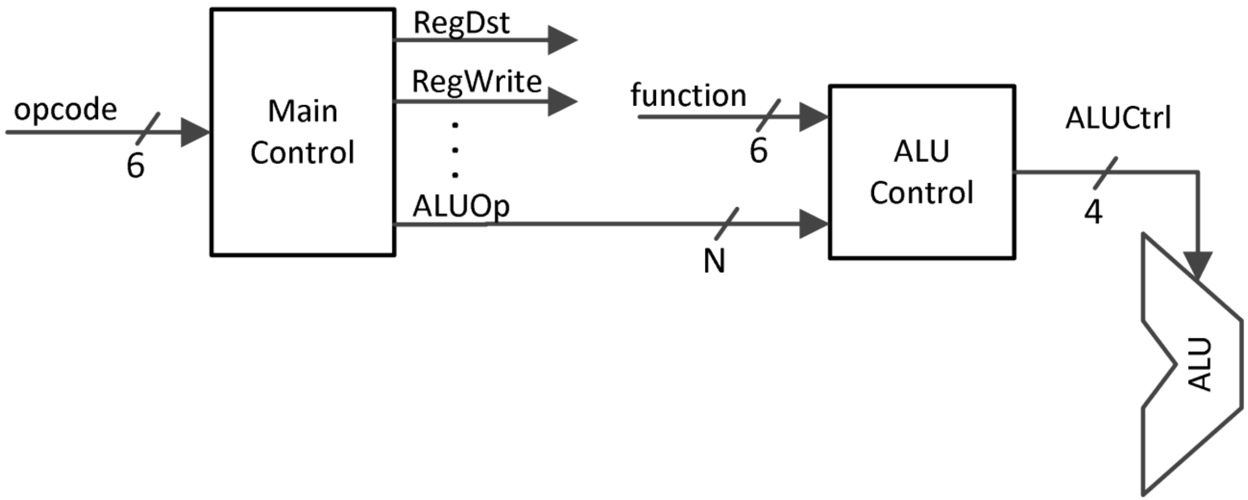
Single-Cycle Data-Path with Control Signals for MIPS-lite

[1]

Single-Cycle CPU Design – Step 4

- Design Step 4: Identifying the Control Signals
  - Identify and define the function of all control signals needed by the data-path
  - Analyze each instruction to determine the setting of control points that affect the register transfers





The main control signal values for MIPS-lite

Instruction	Reg Dst	Reg Write	ALU Src	PC Src	Mem Read	Mem Write	Memto Reg	ALU Op
R- format	1	1	0	0	0	0	0	10
lw	0	1	1	0	1	0	1	00
sw	X	0	1	0	0	1	X	00
beq	X	0	0	1	0	0	X	01

Single-Cycle CPU Design – Step 4

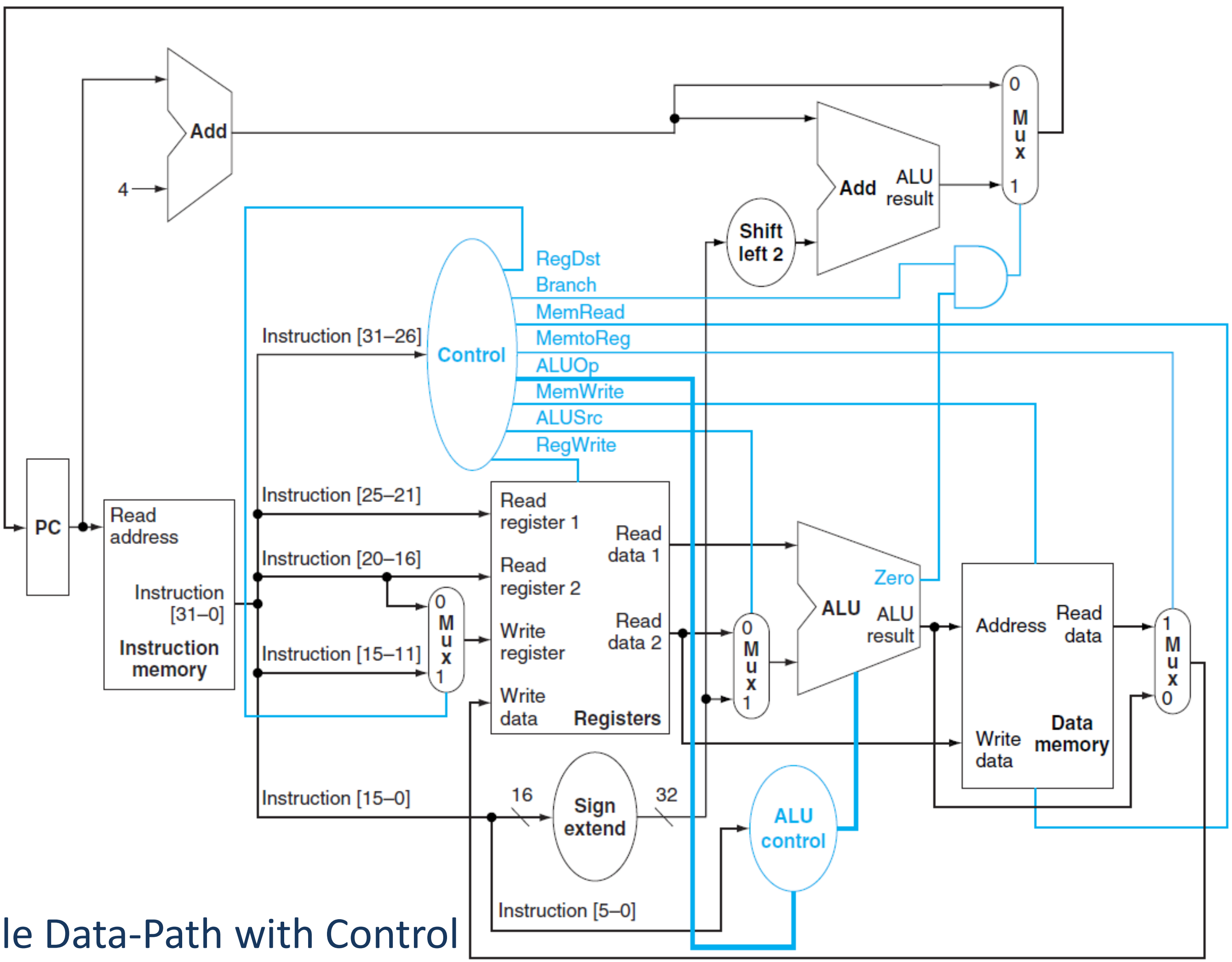
Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field	The register destination number for the Write register comes from the rd field
RegWrite	None	The register on the Write register input is written into with the value on the Write data input
ALUSrc	The second ALU operand comes from the second Register file output	The second ALU operand is the sign-extended lower 16-bits of the instruction
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4	The PC is replaced by the output of the adder that computes the branch address
MemRead	None	Data memory contents at the read address are put on read data output
MemWrite	None	Data memory contents at address given by write address is replaced by value on write data input
MemtoReg	The value fed to the register write data input comes from the ALU	The value fed to the register write data input comes from the data memory

The Meaning of the Control Signals

ALUOp – defines the behavior of the ALU control

PCSrc = Branch AND Zero

Single-Cycle CPU Design – Step 4



Single-Cycle Data-Path with Control [1]

Single-Cycle CPU Design – Step 4

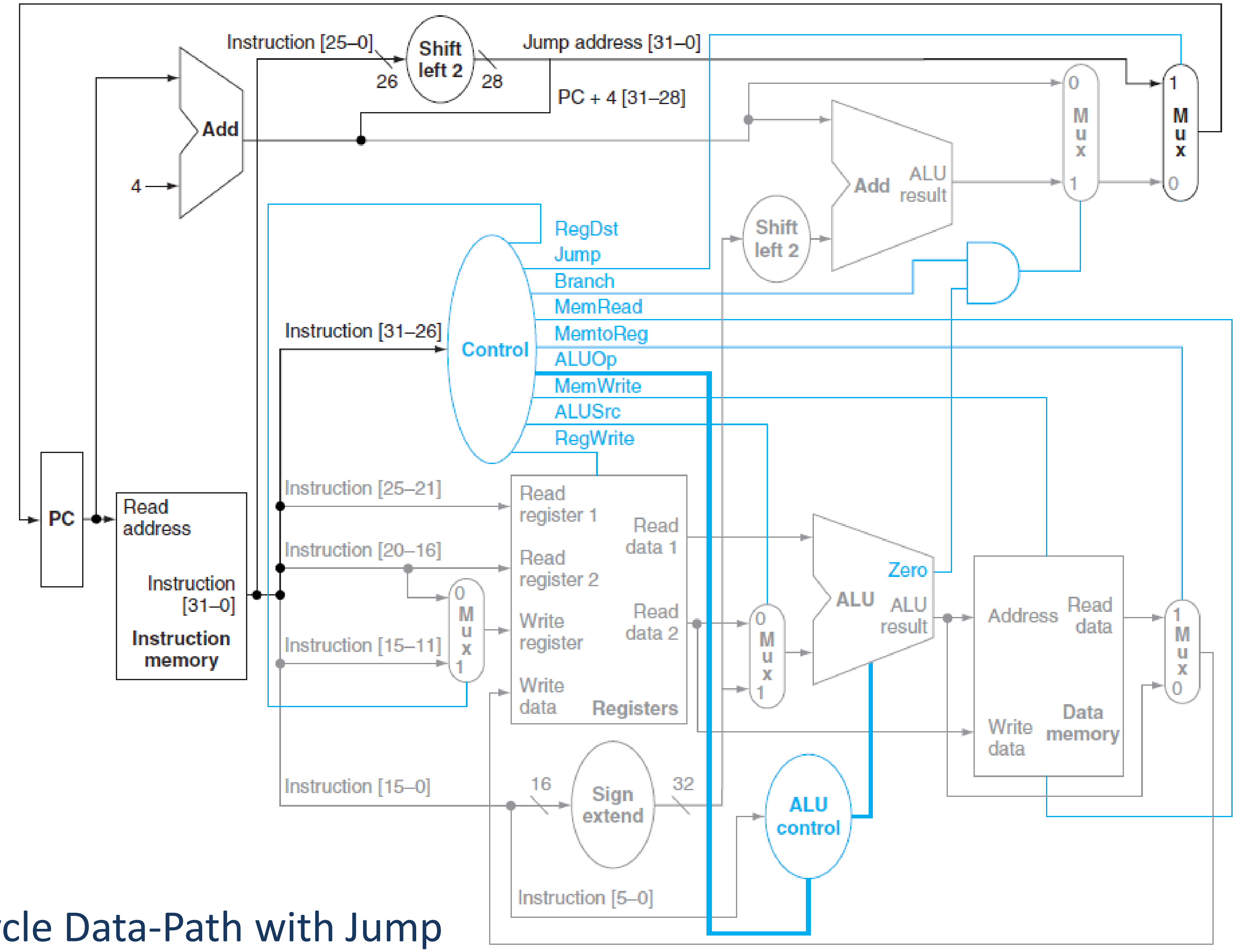
Instruction Opcode	ALUOp	Instruction Operation	Function Field	Desired ALU Operation	ALU Control
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	and	100100	and	0000
R-type	10	or	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

Local Control for the ALU

opcode → ALUOp → ALUCtrl

Our example uses 2-bits for ALUOp. It can be further extended if necessary

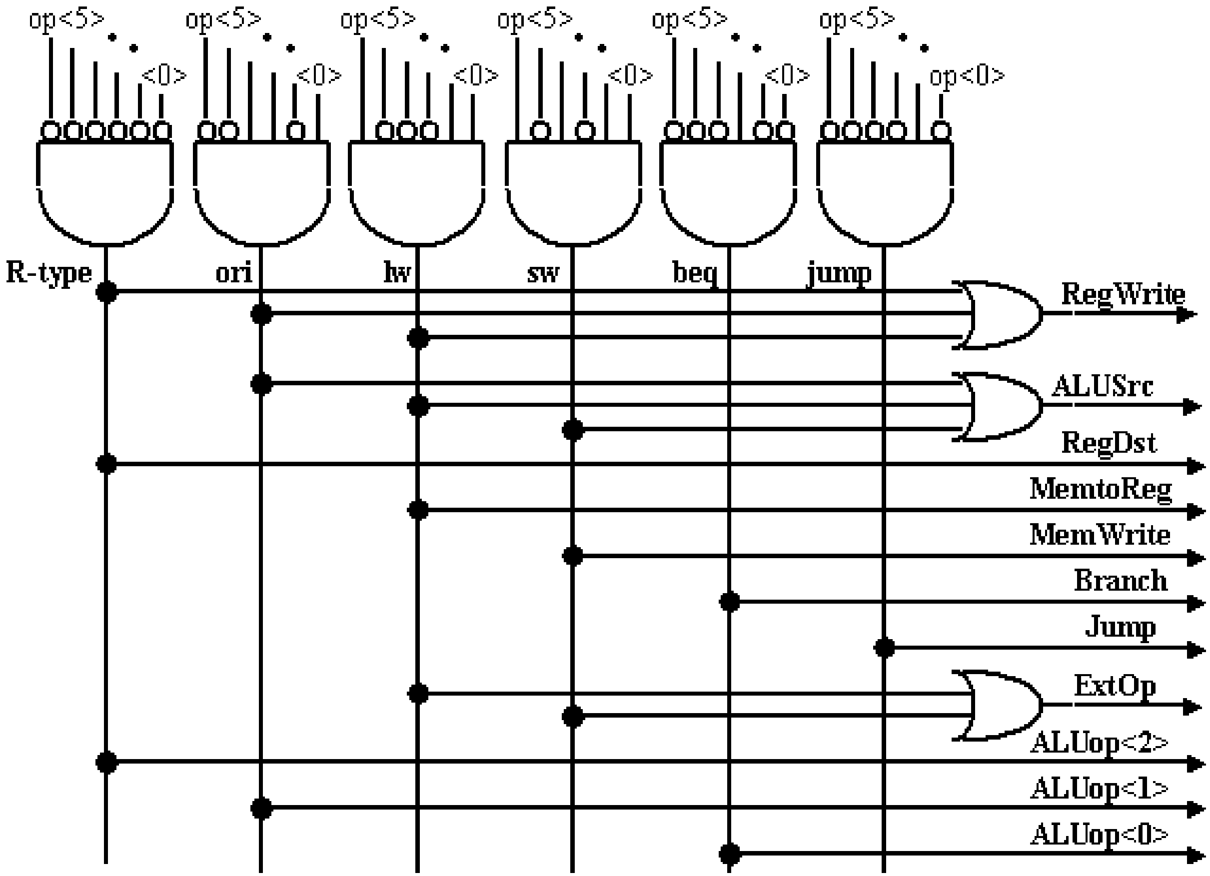
Single-Cycle CPU Design – Step 4



Single-Cycle Data-Path with Jump [1]

Single-Cycle CPU Design – Step 5

- Design Step 5: Implement the Control
  - Only combinational logic is needed for the Single-Cycle Control

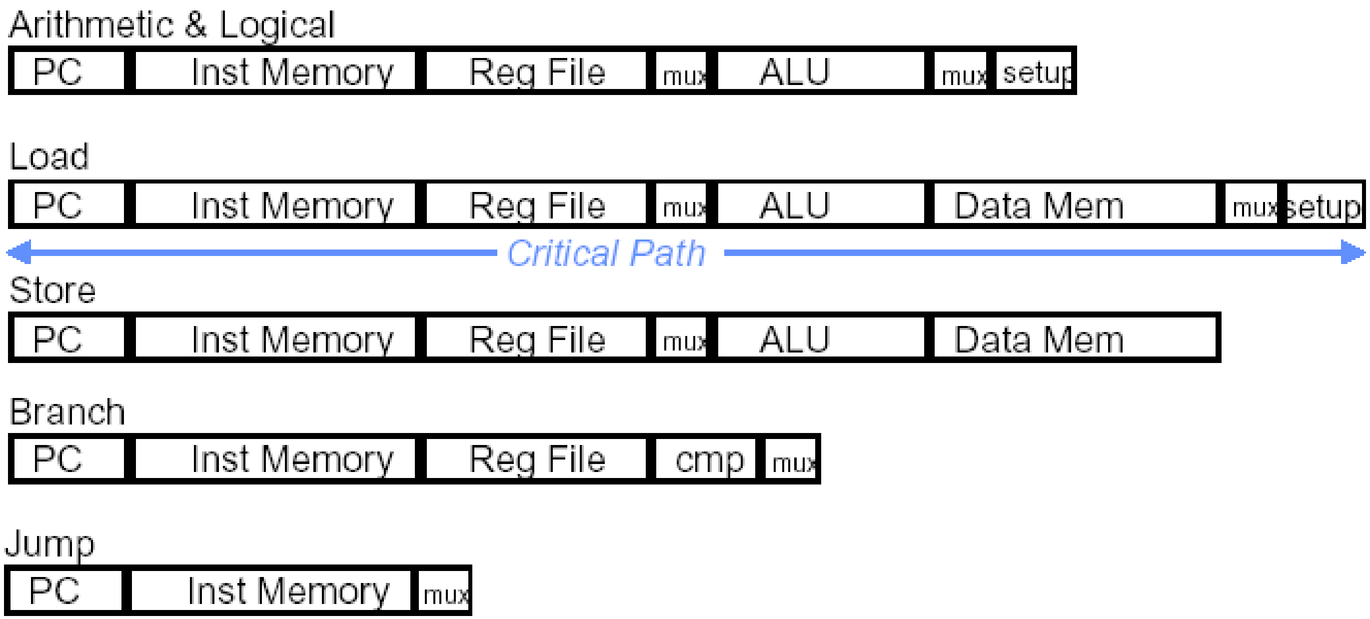


A possible PLA implementation of the Main Control Unit

[1]

Single-Cycle CPU Design

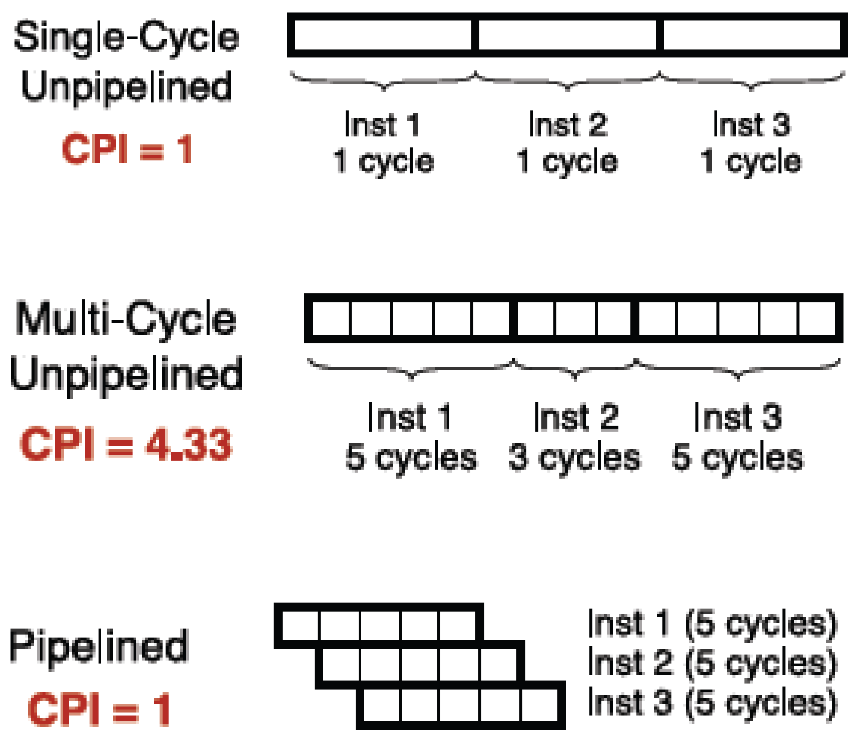
- Critical Path
  - Load Word operation: PC's Clk-to-Q + Instruction Memory's Access Time + Register File's Access Time + ALU to Perform a 32-bit Add + Data Memory Access Time + Setup Time for Register File Write + Clock Skew



Single cycle: Instruction Timing Comparison

Single-Cycle CPU Design

- Single-Cycle disadvantages
  - The clock cycle is chosen to fullfill the critical path (lw) → slow clock
  - The time needed for a load is much larger than for other instructions



Processor type	CPI	CLK cycle time
Single-Cycle	1	Long cycle time
Multi-Cycle	>1	Short cycle time
Pipelined	~1	Short cycle time

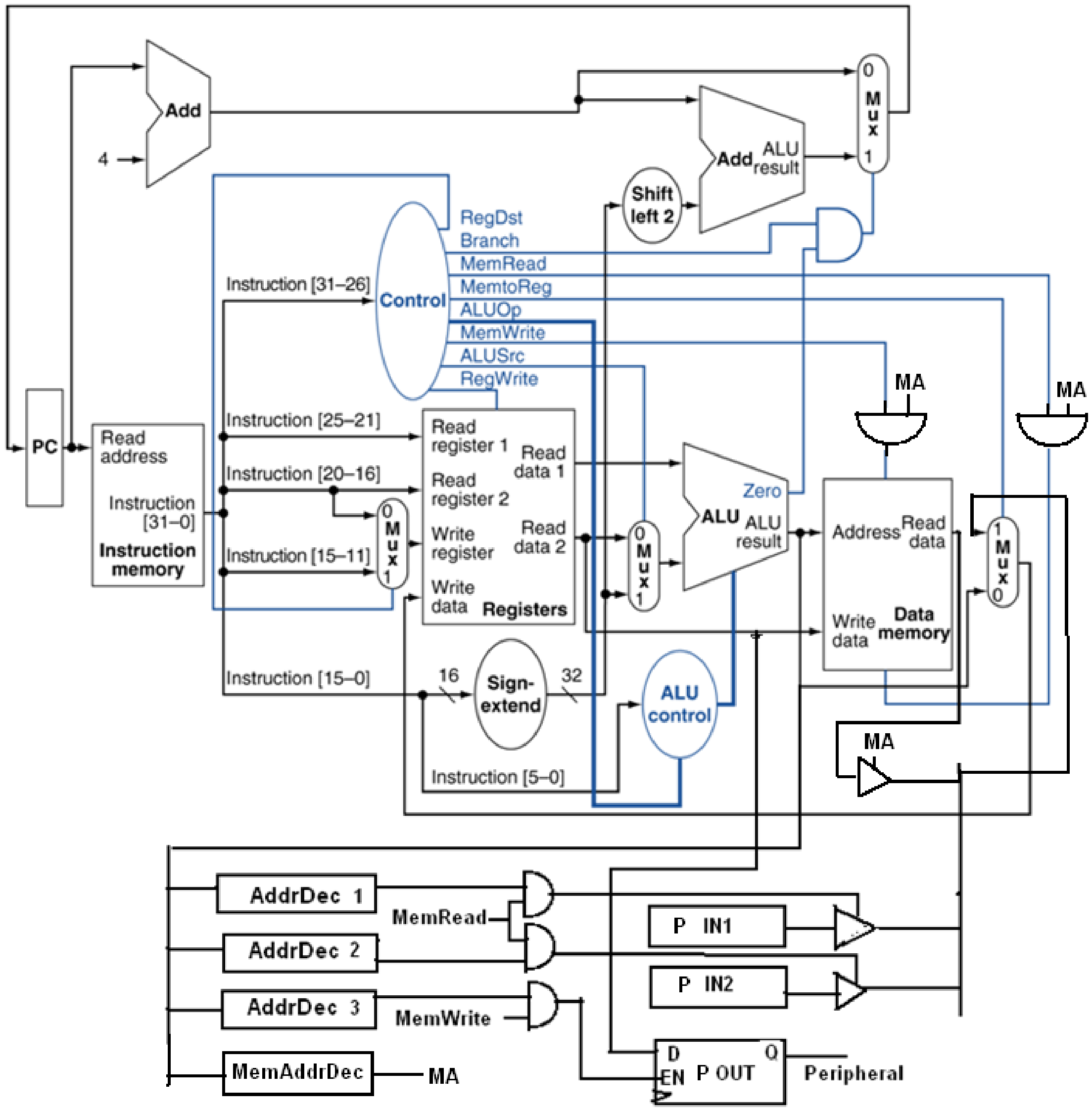
Remember: Computer Performance

$$\frac{Time}{Program} = \frac{Instructions}{Program} \cdot \frac{Cycles}{Instruction} \cdot \frac{Time}{Cycle}$$

Single-Cycle CPU Design – I/O

- Single-Cycle CPU Extensions
  - Problem: define some ports for I/O communication
  - A convenient solution (without introducing dedicated instructions)
    - I/O mapped through the memory address space
    - Some addresses from data memory will be reserved for I/O ports
    - Writing and reading to this I/O ports is carried out by using the standard instructions for memory accesses (lw and sw)
    - LW and SW used for word (32-bits) transfers (use LH and SH for half words, LB and SB for bytes)
    - You need to specify in the RTL description how to handle the reserved addresses for peripherals. From RTL will result the necessary supplementary components (address decoders, etc.)

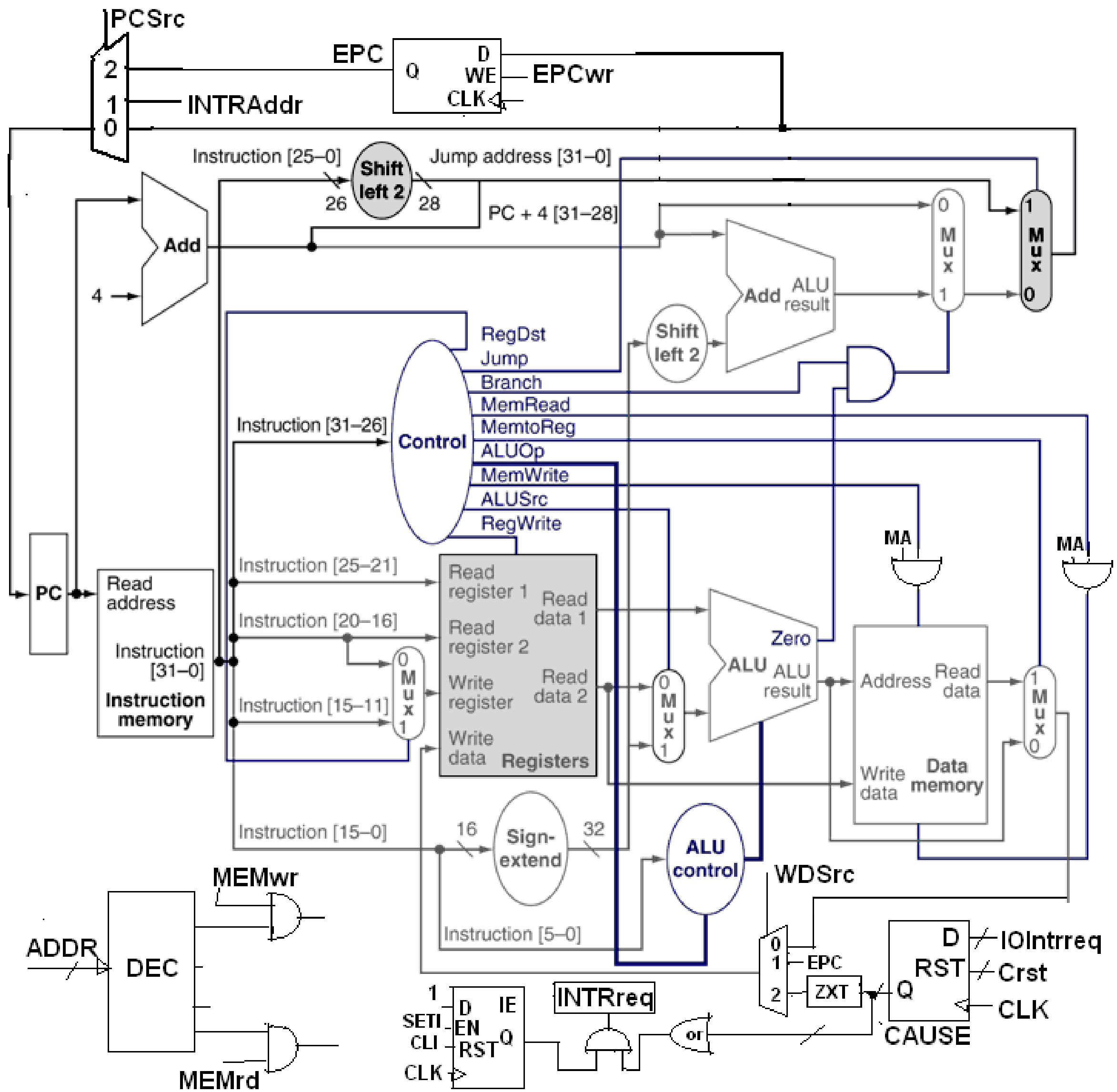
Single-Cycle CPU Design – I/O



Connecting I/O Devices

- Devices are mapped through the memory address space
  - 2 INPUTS
  - 1 OUTPUT
- Control Signals:
  - MemRead
  - MemWrite
  - MA

Single-Cycle CPU Design – I/O



MIPS Interrupt Mechanism

- IE – Interrupt Enable
- ZXT – Zero Extender

- See the previous course

Problems – Homework

- Implement other instructions for the Single-Cycle MIPS CPU
  - add, sub, and, or, lw, sw, beq, j, addi, andi, ori
  - sll, srl, sra, sllv, srlv, srav
  - slt, slti
  - bne , bgez, bltz,…
  - jr, jal
  - ….
- Implement new instructions for the Single-Cycle MIPS CPU
  - LWR, SWR (sums two registers to obtain the memory address)
  - LWA, SWA (uses a single register to obtain the memory address)
  - SWAP two registers
  - Arithmetic/logical instructions with memory operands
    - addm $t2, 100($t3) $t2 ← $t2 + M[$t3+100]

References

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.
2. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011.
3. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
4. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.