Technical University of Cluj-Napoca
Computer Science Department

Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 6: Multi-Cycle CPU Design

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

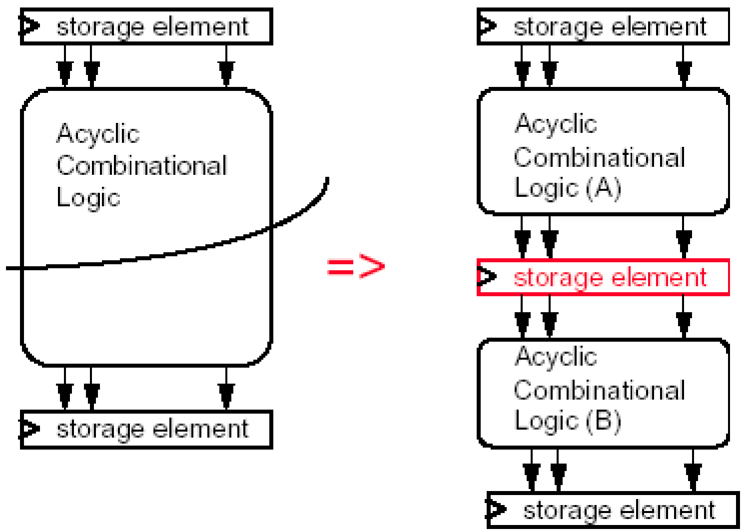
Multi-Cycle Processor Design

- Step-by-step Processor Design → Multi cycle MIPS
  - Step 1: ISA → Abstract RTL
  - Step 2: Components of the Data-Path
  - Step 3: RTL + Components → Data-Path
  - Step 4: Data-Path + Abstract RTL → Concrete RTL
  - Step 5: Concrete RTL → Control
- Single Cycle Problems
  - Long Cycle Time
    - All instructions take as much time as the slowest
    - What happens for floating point?
  - Waste of area: no component reuse
- One Possible Solution
  - use a “smaller” cycle time
  - different instructions take different numbers of cycles

→ Multi-Cycle Data-Path

Reducing the Clock Cycle Time

- Cut the combinational dependency graph and insert registers
  - Do the same amount of work in two fast cycles, rather than one slow one



Break up the long combinational stages

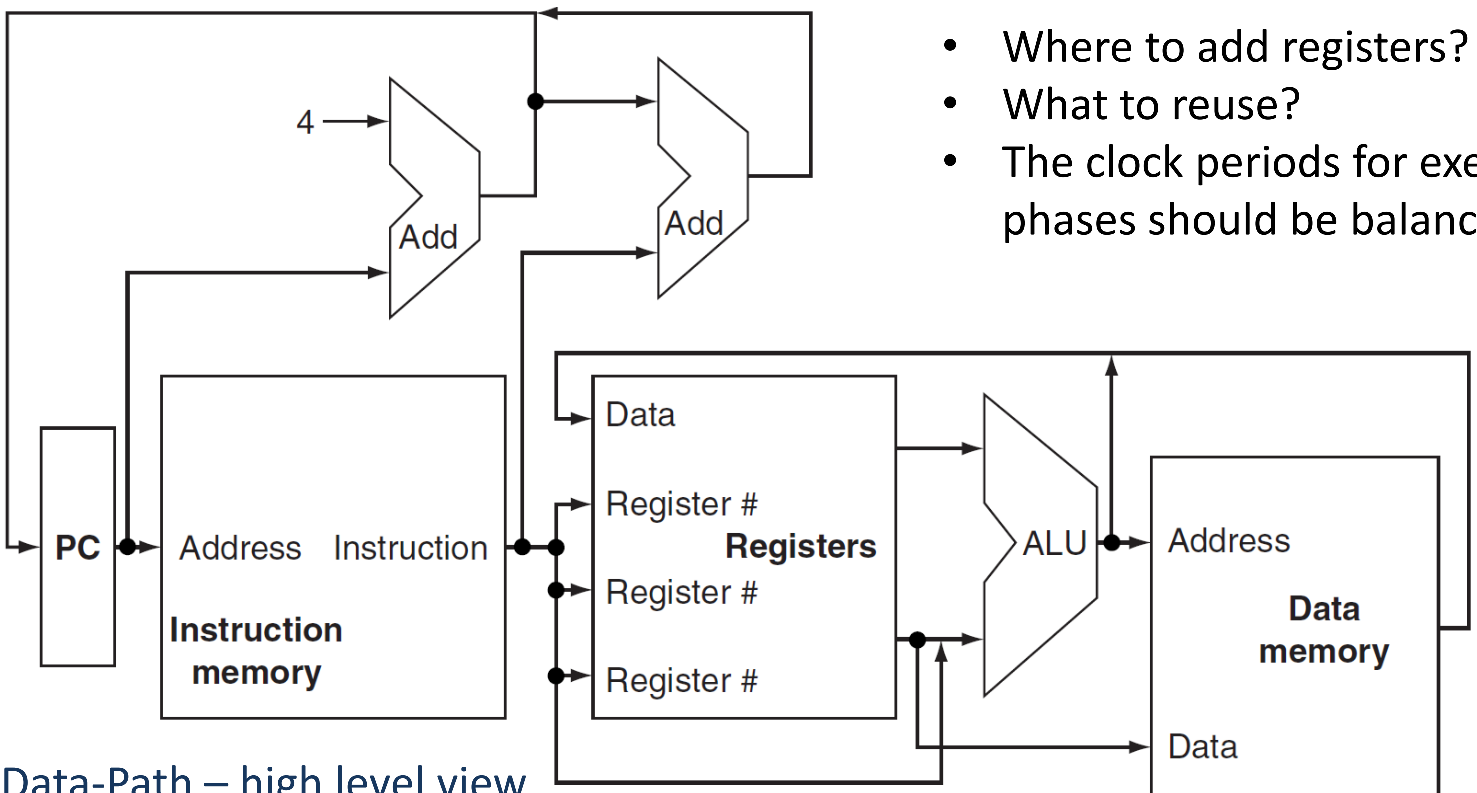
Limits on Cycle Time in different stages		
Next address logic	PC ← branch ? PC + offset : PC + 4	Address logic computation time
Instruction Fetch	IR ← M[PC]	Memory access time
Register Access	A ← RF[rs]	Register file access time
ALU operation	RF[rd] ← A + B	ALU operation delay

Multi-Cycle Approach

- Break up the instructions into steps, each step takes one cycle
  - Balance the amount of work to be done.
  - Restrict each cycle to use only one major functional unit.
- At the end of a cycle
  - Store values for use in later cycles.
  - Introduce additional “internal” registers (not programmer visible).
- Reuse functional units
  - ALU used to compute address and to increment PC (beside usual ALU operations)
  - Only one Memory used for Instruction and Data!
- Use a finite state machine (FSM) for control

Multi-Cycle CPU Design – Step 1, 2

- Step 1: ISA → Abstract RTL
  - The same instructions as for the Single-Cycle MIPS
- Step 2: Components of the Data-Path
  - Partitioning the Single-Cycle Data-Path

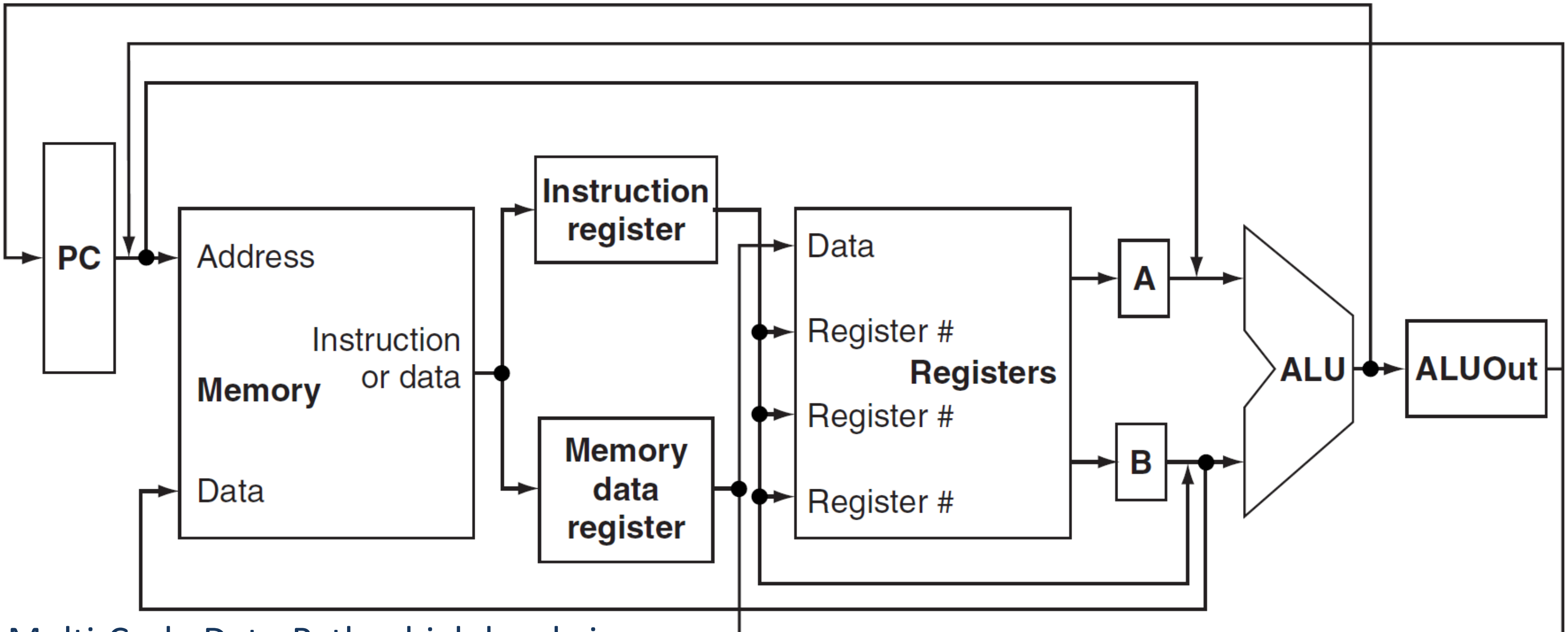


Single-Cycle Data-Path – high level view

[1]

- Where to add registers?
- What to reuse?
- The clock periods for execution phases should be balanced

Multi-Cycle CPU Design – Step 2

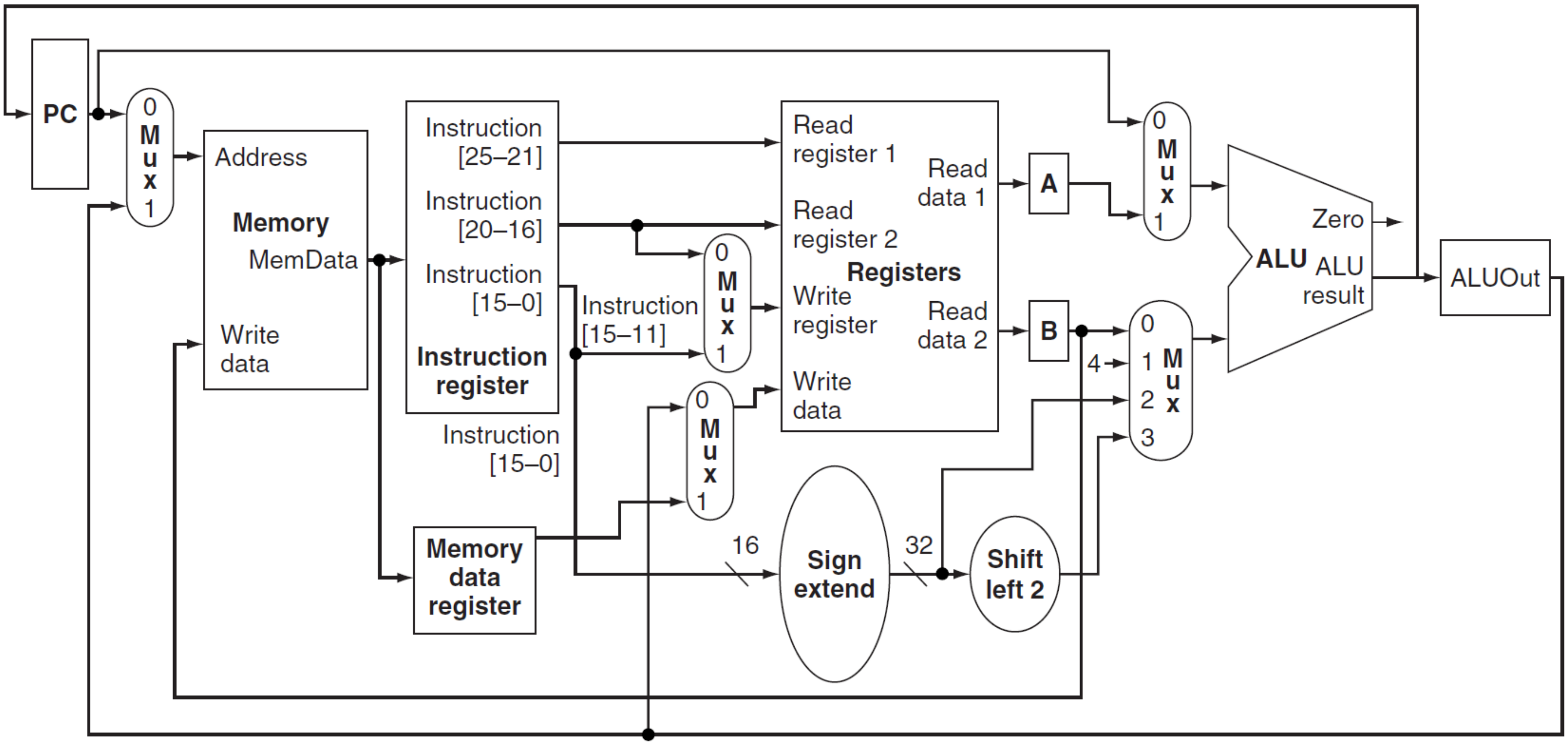


Multi-Cycle Data-Path – high level view

[1]

- Added registers (not visible to the programmer):
  - IR – Instruction Register; MDR – Memory Data Register
  - A, B – register file read data registers; ALUOut – ALU output register.
  - Data used by subsequent instructions are stored in programmer visible registers (i.e., register file, PC) or memory.
- Memory and ALU reused

Multi-Cycle CPU Design – Step 2

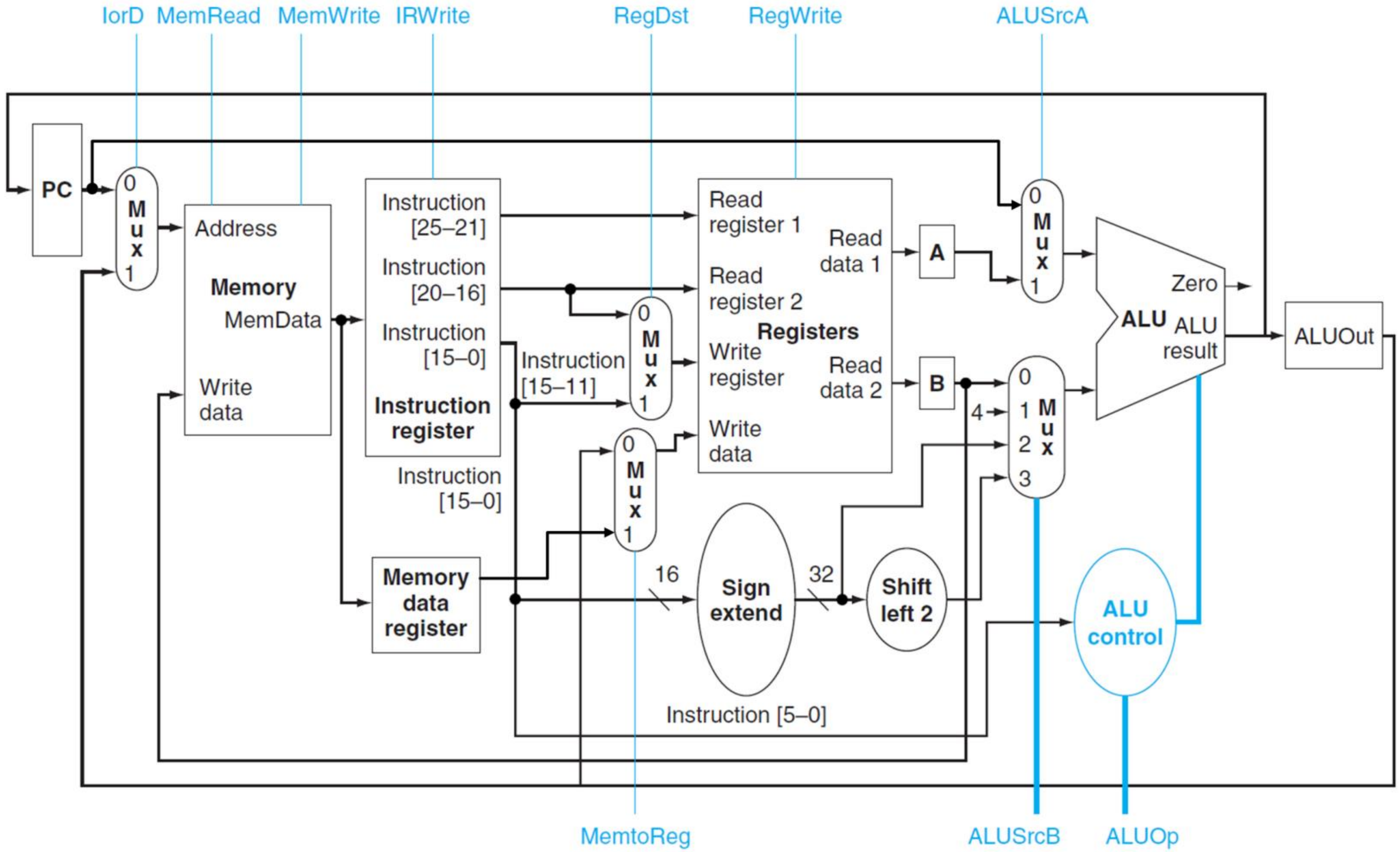


[1]

Multi-Cycle Data-Path derived from Single-Cycle MIPS

Multi-Cycle CPU Design – Step 3

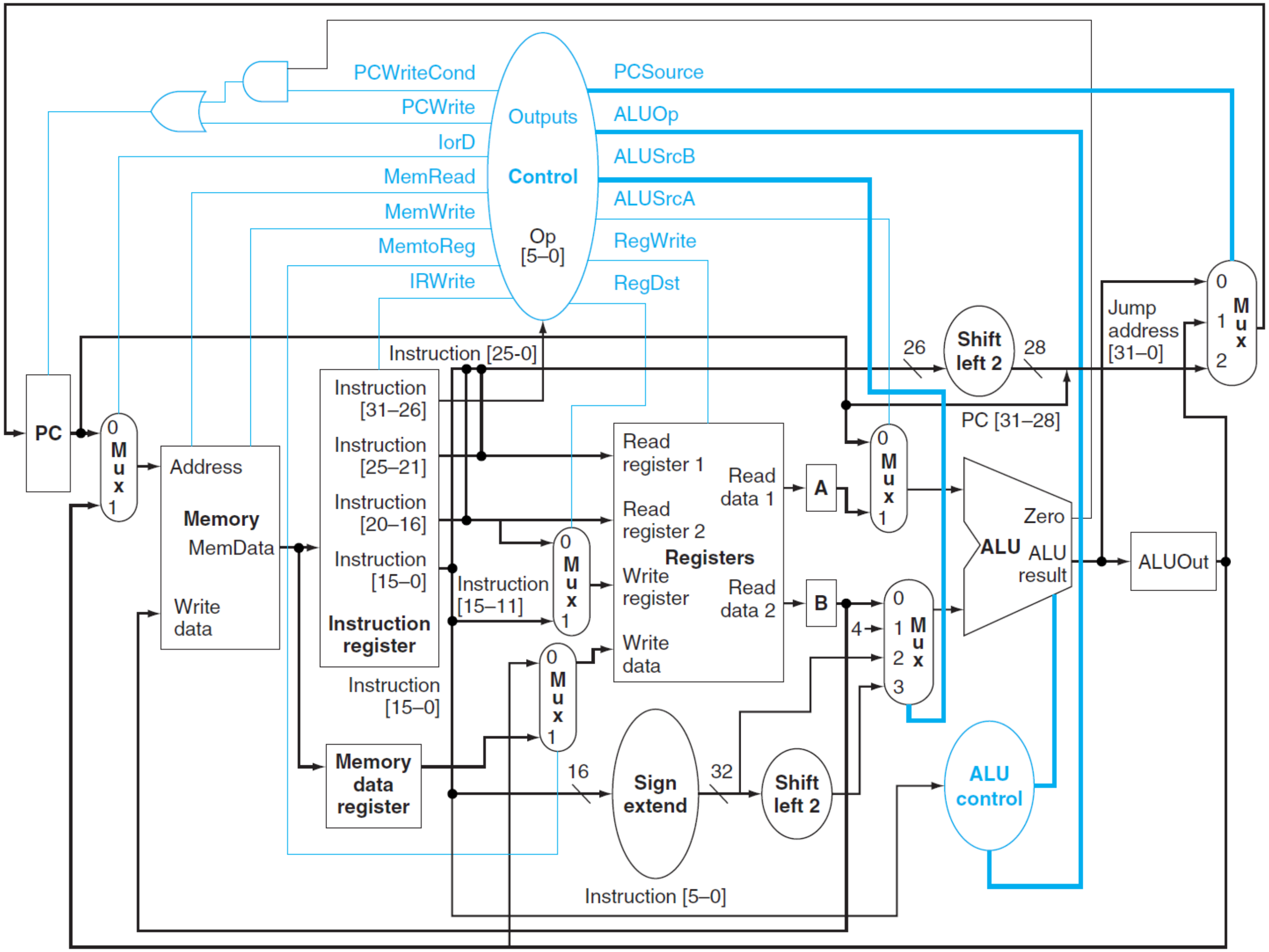
- Step 3: RTL + Components → Data-Path
  - We connect the components to build the Data-Path, and specify the Control Signals
  - Instruction Register (IR): IR[25:21] → rs, IR[20:16] → rt, IR[15:11] → rd



[1]

Multi-Cycle Data-Path with Control Signals

Multi-Cycle CPU Design – Step 3



[1]

Multi-Cycle Data-Path with Control Unit

Multi-Cycle CPU Design – Step 3

Signal name	Effect when deasserted (=0)	Effect when asserted (=1)
RegDst	The register destination number for the write register comes from the rt field (instruction bits 20:16)	The register destination number for the write register comes from the rd field (instruction bits 15:11)
RegWrite	None	The register on the write register input is written with the value on the Write data input
ALUSrcA	The first ALU operand is the PC (default)	The first ALU operand is register A (i.e. R[rs])
MemRead	None (default)	Content of memory specified by the address input are put on the memory data output
MemWrite	None (default)	Memory contents specified by the address inputs is replaced by the value on the Write data input
MemtoReg	The value fed to the register write data input comes from ALUOut register (default)	The value fed to the register write data input comes from data memory register (MDR)
IorD	The PC is used to supply the address to the memory unit (default)	The ALUOut register is used to supply the address to the memory unit
IRWrite	None (default)	The output of the memory is written into the Instruction Register (IR)
PCWrite	None (default)	The PC is written; the source is controlled by PC source
PCWriteCond	None (default)	The PC is written if the Zero output of the ALU is also active

The Meaning of the 1-bit Control Signals

Multi-Cycle CPU Design – Step 3

Signal Name	Value(Binary)	Effect
ALUOp	**00**	The ALU performs an add operation
	01	The ALU performs a subtract operation
	10	The function field of the instruction determines the ALU operation (R-Type)
ALUSrcB	**00**	The second input of the ALU comes from the B register
	01	The second input of the ALU is the constant 4
	10	The second input of the ALU is the sign-extended 16-bit immediate filed of the instruction in IR
	11	The second input of the ALU is the sign-extended 16-bit immediate field of IR shifted left 2 bits
PCSource	**00**	Output of the ALU (PC + 4) is sent to the PC for writing
	01	The content of the ALUOut (the branch target address) is sent to the PC for writing
	10	The jump target address (IR[25:0) shifted left 2 bits and concatenated with PC+4[31:28] is sent to the PC for writing

The Meaning of the 2-bit Control Signals

Multi-Cycle CPU Design – Step 4

- Step 4: Data path + Abstract RTL → Concrete RTL
  - For the multi cycle data path we write the RTL codes of the basic instructions to establish the necessary Control Signal settings
- Instructions from ISA perspective
  - RTL Abstract
    - Specifies the instruction independent of a concrete implementation
    - Example: arithmetic, R-type instruction

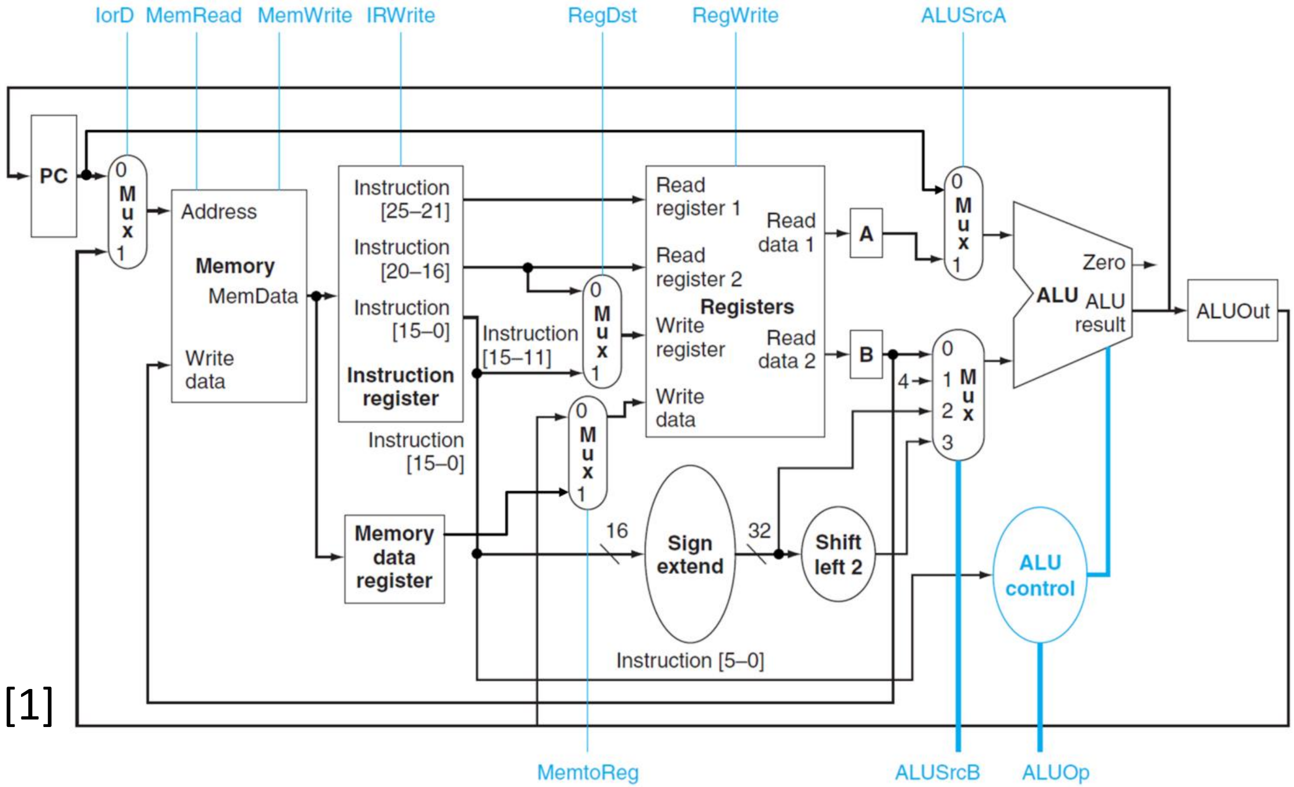
RF[rd] ← RF[rs] op RF[rt]

  - RTL Concrete
    - Describes the execution phases of the instruction for a given implementation
    - Example: arithmetic, R-type instruction

T0 → IR ← M[PC]
T1 → A ← RF[rs], B ← RF[rt]
T2 → ALUOut ← A op B
T3 → RF[rd] ← ALUOut

    - We forgot an important part of the definition! PC ← PC + 4

Multi-Cycle CPU Design – Step 4



[1]

add \$rd, \$rs, \$rt

Abstract RTL:

RF[rd] ← RF[rs] + RF[rt],
PC ← PC + 4

Concrete RTL:

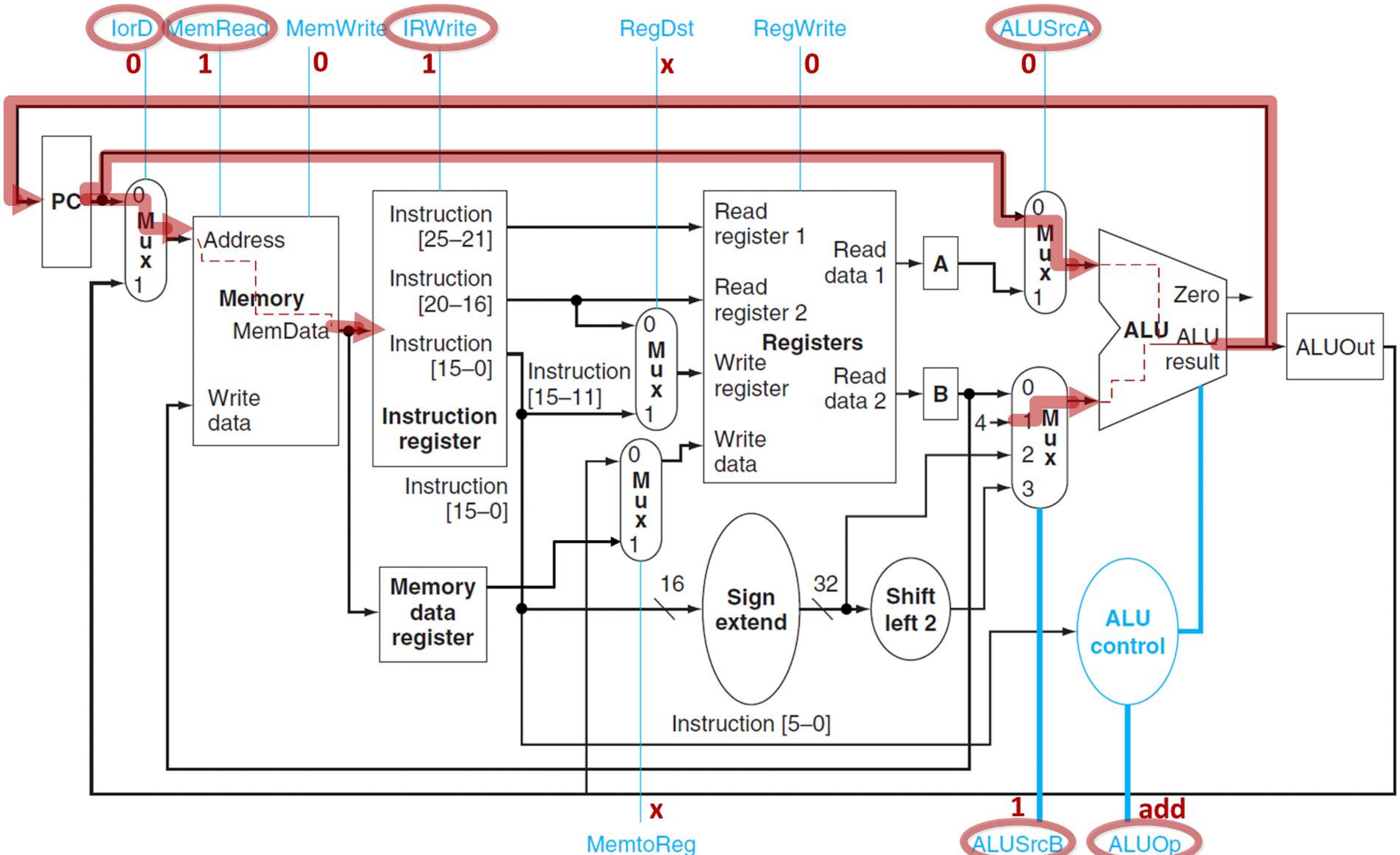
T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt];
ADD & T2 →	ALUOut ← A + B;
ADD & T3 →	RF[rd] ← ALUOut;

	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	x	1	0	func
T3	x	0	0	0	1	0	1	x	x	x	x

Note: the operation is defined by the function field

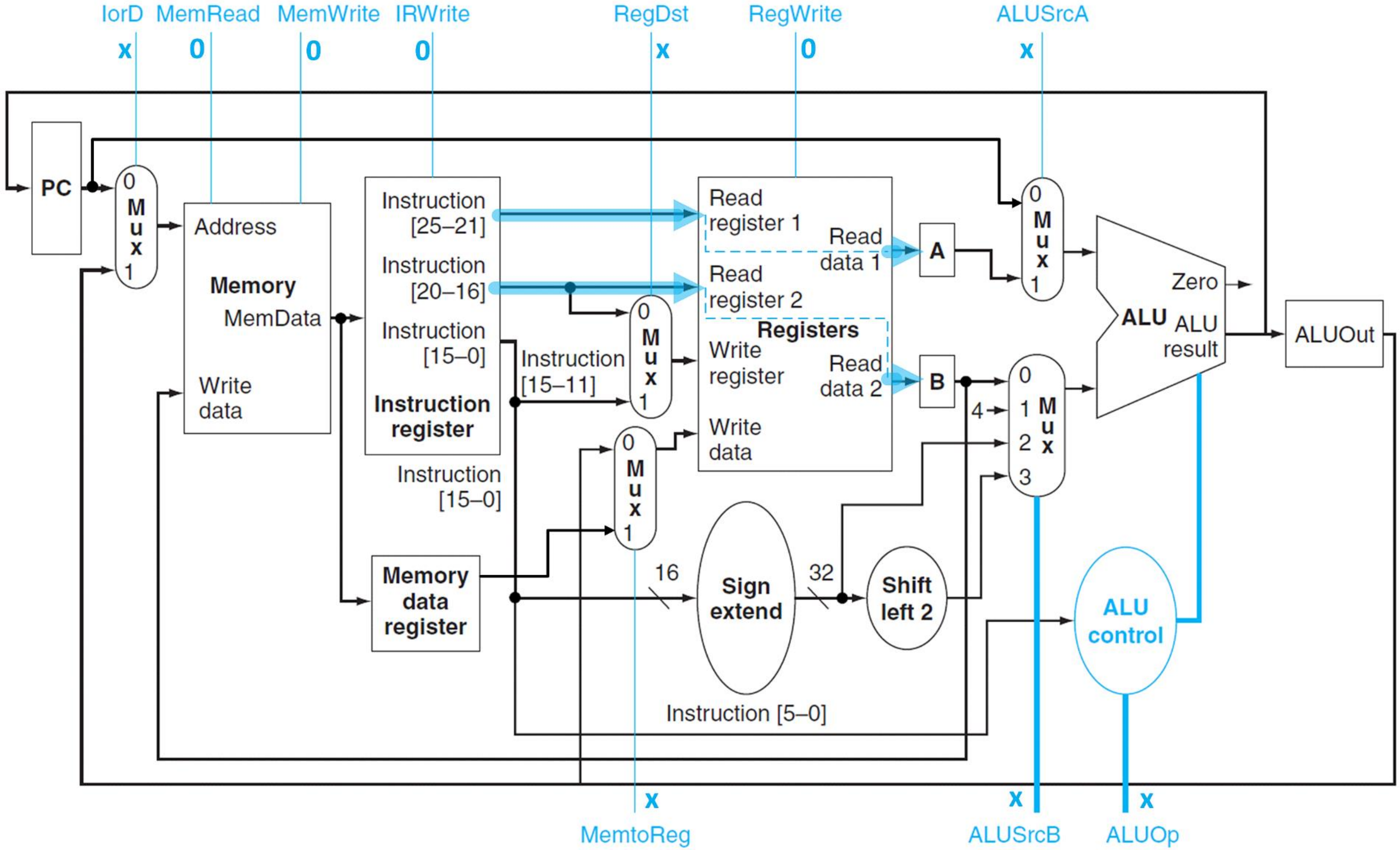
Multi-Cycle CPU Design – Step 4

Add: T0 → IR ← M[PC], PC ← PC+4;



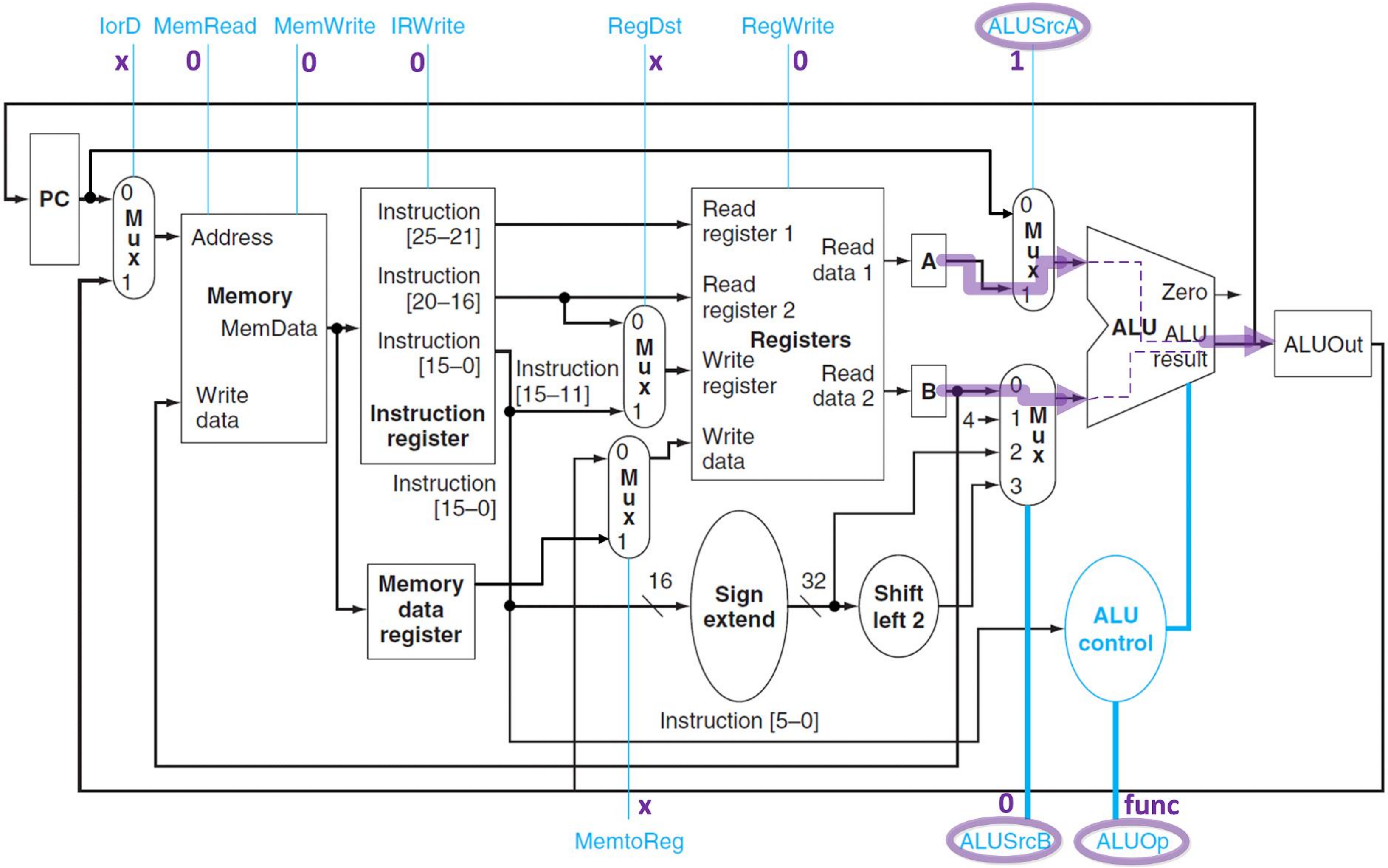
Multi-Cycle CPU Design – Step 4

Add: **T1** → A ← R[rs], B ← R[rt];



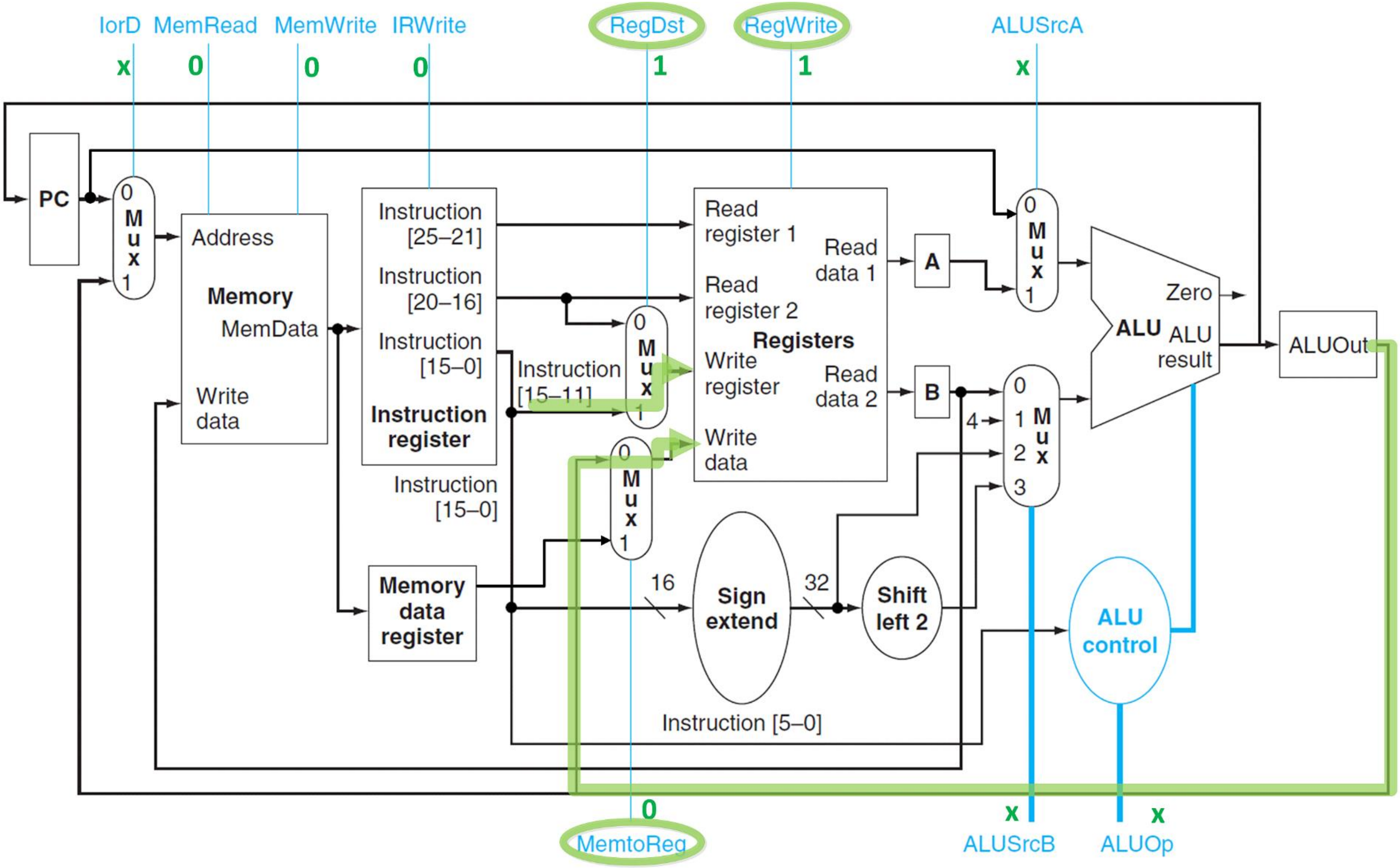
Multi-Cycle CPU Design – Step 4

Add: ADD & T2 → ALUOut ← A + B;

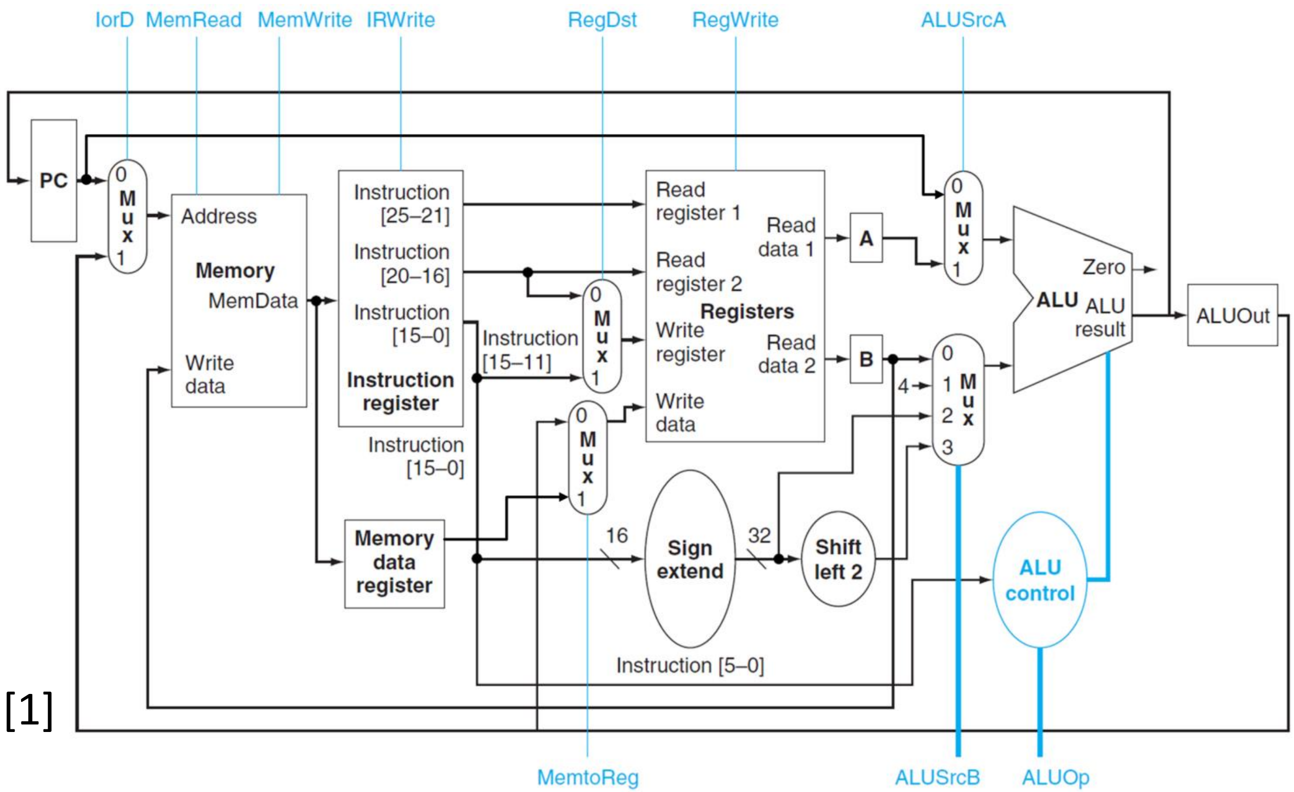


Multi-Cycle CPU Design – Step 4

Add: ADD & T3 → R[rd] ← ALUOut;



Multi-Cycle CPU Design – Step 4



[1]

ori $rs, $rt, imm

Abstract RTL:

RF[rt] ← RF[rs] | Z_Ext(imm),
PC ← PC + 4

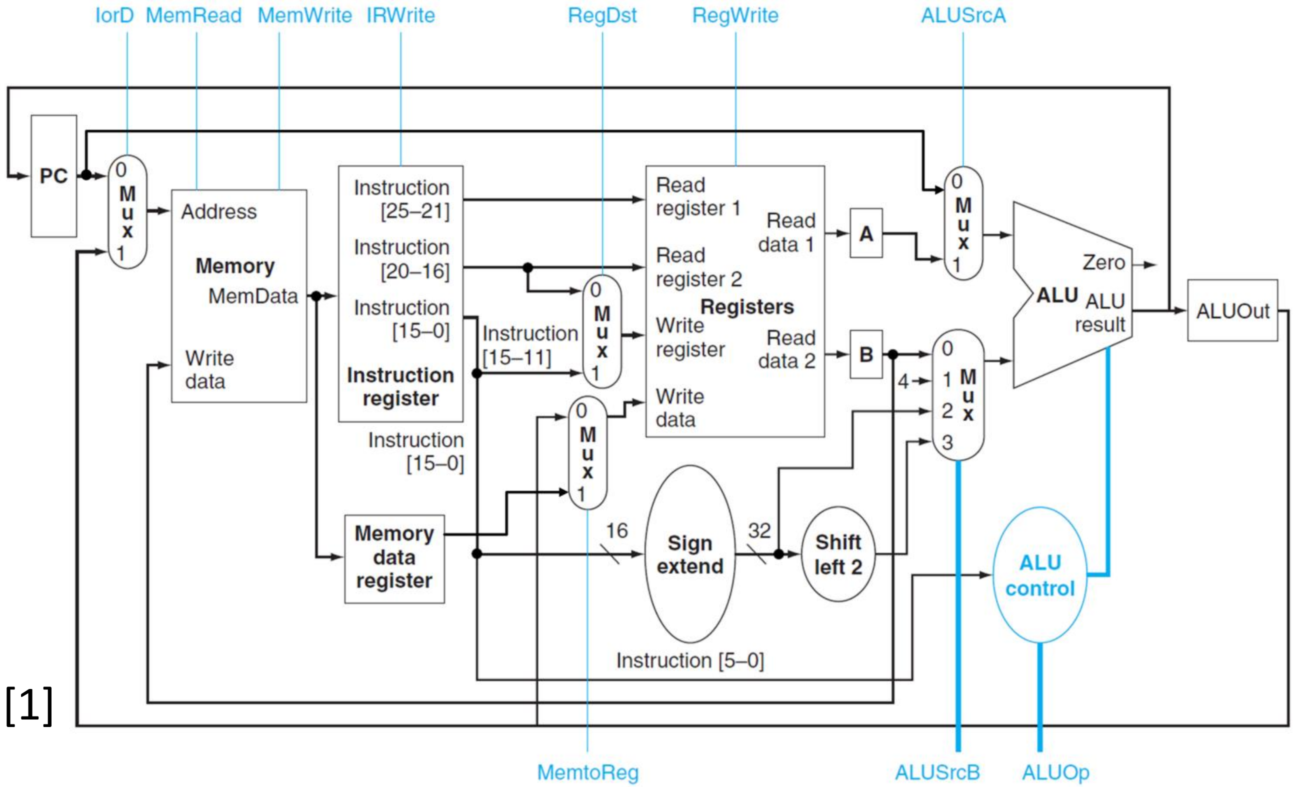
Concrete RTL:

T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt];
ORI & T2 →	ALUOut ← A | Z_Ext(imm);
ORI & T3 →	RF[rt] ← ALUOut;

	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	0	1	2	or
T3	x	0	0	0	0	0	1	x	x	x	x

Note: the operation is defined by the opcode field

Multi-Cycle CPU Design – Step 4



[1]

lw $rt, imm($rs)

Abstract RTL:

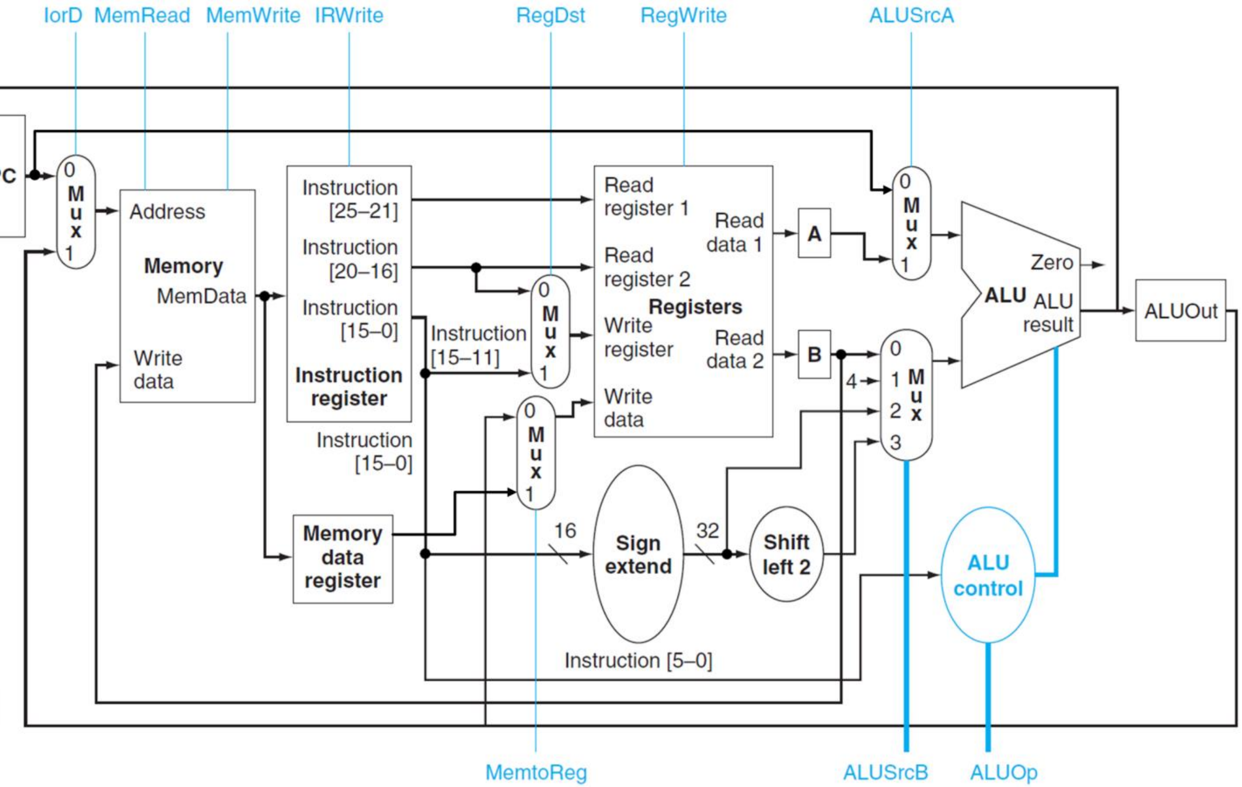
RF[rt] ← M[RF[rs] + S_Ext(imm)],
PC ← PC + 4

Concrete RTL:

T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt];
LW & T2 →	ALUOut ← A + S_Ext(imm);
LW & T3 →	MDR ← M[ALUOut]
LW & T4 →	RF[rt] ← MDR;

	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	1	1	2	add
T3	1	1	0	0	x	x	0	x	x	x	x
T4	x	0	0	0	0	1	1	x	x	x	x

Multi-Cycle CPU Design – Step 4



[1]

sw $rt, imm($rs)

Abstract RTL:

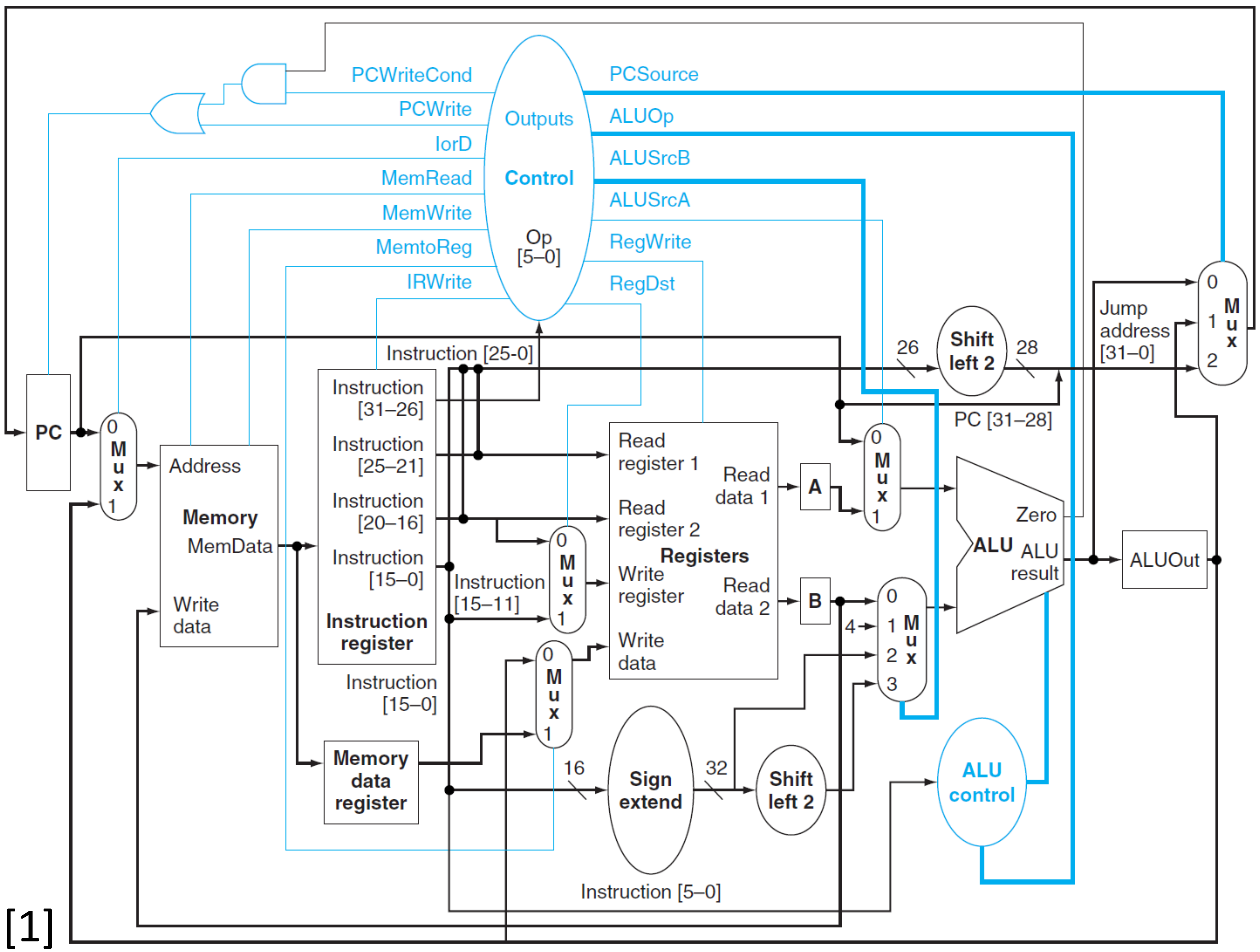
M[RF[rs] + S_Ext(imm)] ← RF[rt],
PC ← PC + 4

Concrete RTL:

T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt];
SW & T2 →	ALUOut ← A + S_Ext(imm);
SW & T3 →	M[ALUOut] ← B

	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op
T0	0	1	0	1	x	x	0	x	0	1	add
T1	x	0	0	0	x	x	0	x	x	x	x
T2	x	0	0	0	x	x	0	1	1	2	add
T3	1	0	1	0	x	x	0	x	x	x	x

Multi-Cycle CPU Design – Step 4



[1]

beq $rt, $rs, imm

Abstract RTL:

If(RF[rs] == RF[rt]) then

PC ← PC + 4 + S_Ext(imm) <<2

else

PC ← PC + 4

Concrete RTL:

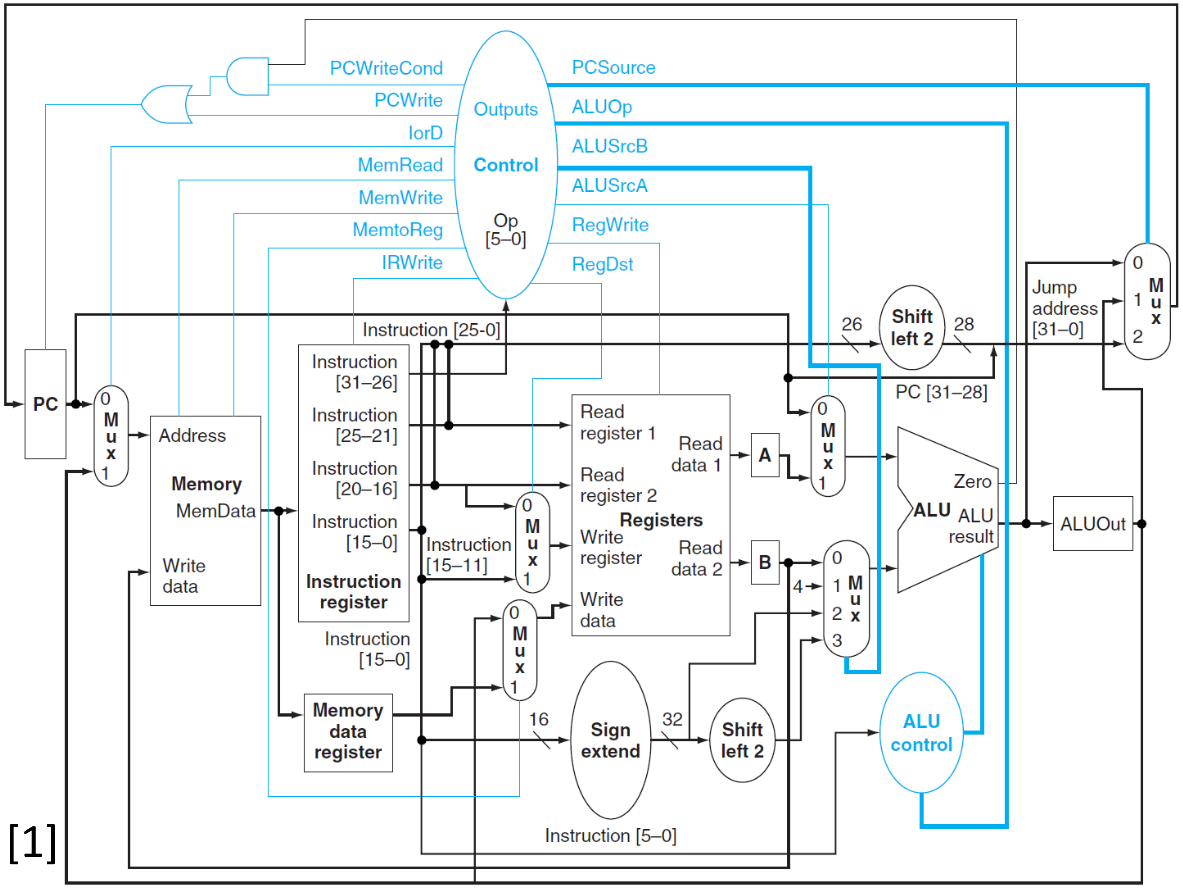
T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt];
BEQ & T2 →	ALUOut ← PC + S_Ext(imm) << 2;

BEQ & T3 (A == B) →

PC ← ALUOut;

Note: T1 and T2 can be executed in parallel, in the same clock period!

Multi-Cycle CPU Design – Step 4



[1]

beq $rt, $rs, imm

Abstract RTL:

If(RF[rs] == RF[rt]) then

PC ← PC + 4 + S_Ext(imm) <<2

else

PC ← PC + 4

Concrete RTL:

T0 →	IR ← M[PC], PC ← PC + 4;
T1 →	A ← RF[rs], B ← RF[rt], ALUOut ← PC + S_Ext(imm) << 2;

BEQ & T2 (A == B) →

PC ← ALUOut;

jmp → Homework

	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op	PC Src	PC WrCd	PC Wr
T0	0	1	0	1	x	x	0	x	0	1	add	0	0	1
T1	x	0	0	0	x	x	0	1	0	3	add	x	0	0
T2	x	0	0	0	x	x	0	x	1	0	sub	1	1	0

Multi-Cycle CPU Design – Summary

- Five Execution Phases
  - Instruction Fetch
  - Instruction Decode and Register Fetch
  - Execution, Memory Address Computation, or Branch / Jump Completion
  - Memory Access or Arithmetical – Logical instruction completion
  - Write-back
- Instructions take from 3 to 5 clock cycles
- In one clock cycle all operations are done in parallel, not sequential!
  - T0 → IR ← M[PC] and PC ← PC+4 are done simultaneously
- Between Clock T1 and Clock T2 the control unit will select the next step in accordance to the instruction type

Multi-Cycle CPU Design – Summary

Step / Cycle	Action for R-type instructions	Action for ORI instruction	Action for memory reference instructions	Action for branches	Action for jumps
T0: Instruction Fetch	IR ← M[PC] PC ← PC + 4				
T1: Instruction decode / register Fetch	A ← RF[IR[25:21]] B ← RF[IR[20:16]] ALUOut ← PC + S_Ext(IR[15:0] << 2				
T2: Execution, address computation, branch / jump completion	ALUOut ← A op B	ALUOut ← A OR Z_Ext(Imm16)	ALUOut ← A + S_Ext(IR[15:0])	If (A == B) PC ← ALUOut	PC ← PC[31:28]|| IR[25:0] << 2
T3: Memory access or R-type completion	RF[IR[15:11] ← ALUOut	RF[IR[20:16]] ← ALUOut;	LW: MDR ← M[ALUOut] SW: M[ALUOut] ← B		
T4: Memory read completion			LW: RF[IR[20:16]] ← MDR		

Multi-Cycle CPU Design – Control Signals

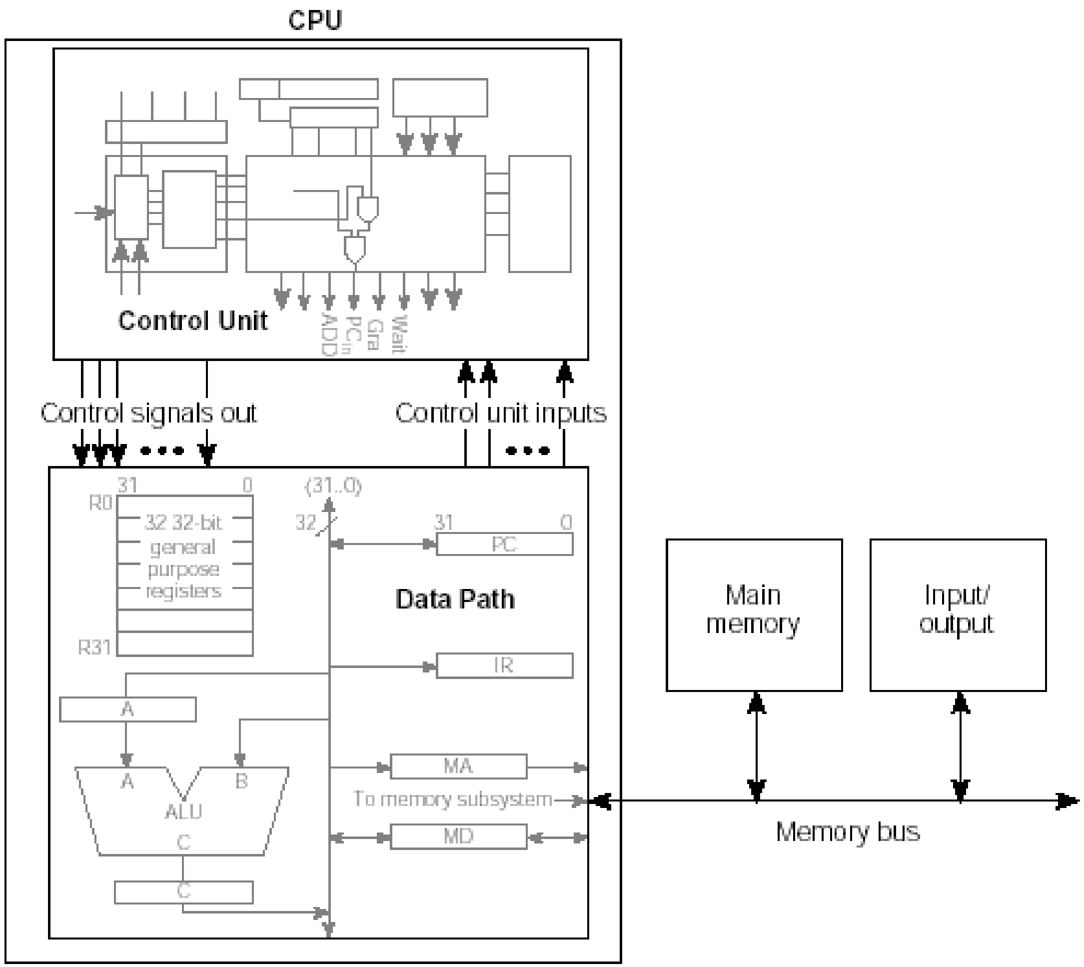
T	IorD	Mem Read	Mem Write	IR Write	Reg Dst	Mem toReg	Reg Write	Ext Op	ALU SrcA	ALU SrcB	ALU Op	PC Src	PC WrCd	PC Wr	
T0	0	1	0	1	x	x	0	x	0	1	add	0	0	1	IF
T1	x	0	0	0	x	x	0	1	1	3	add	x	0	0	ID
T2	x	0	0	0	x	x	0	x	1	0	fun	x	0	0	Ex R-T
T3	x	0	0	0	1	0	1	x	x	x	x	x	0	0	Wb R-T
T2	x	0	0	0	x	x	0	0	1	2	or	x	0	0	Ex ORI
T3	x	0	0	0	0	0	1	x	x	x	x	x	0	0	Wb ORI
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	Ex LW
T3	1	1	0	0	x	x	0	x	x	x	x	x	0	0	M LW
T4	x	0	0	0	0	1	1	x	x	x	x	x	0	0	Wb LW
T2	x	0	0	0	x	x	0	1	1	2	add	x	0	0	Ex SW
T3	1	0	1	0	x	x	0	x	x	x	x	x	0	0	M SW
T2	x	0	0	0	x	x	0	x	x	x	sub	1	1	0	Ex BEQ
T2	x	0	0	0	x	x	0	x	x	x	x	2	0	1	Ex J

Table 1: The Values of the Control Signals in each Clock Cycle

- Execution phases: IF, ID, Ex – Execute, M – Memory, Wb – Write back
- Instructions: R – R-type, LW – Load, SW – Store, BEQ – Branch , J – Jump , ORI – I-type
- ExtOp: 1/0 → 1 – arithmetic, 0 – logical operations

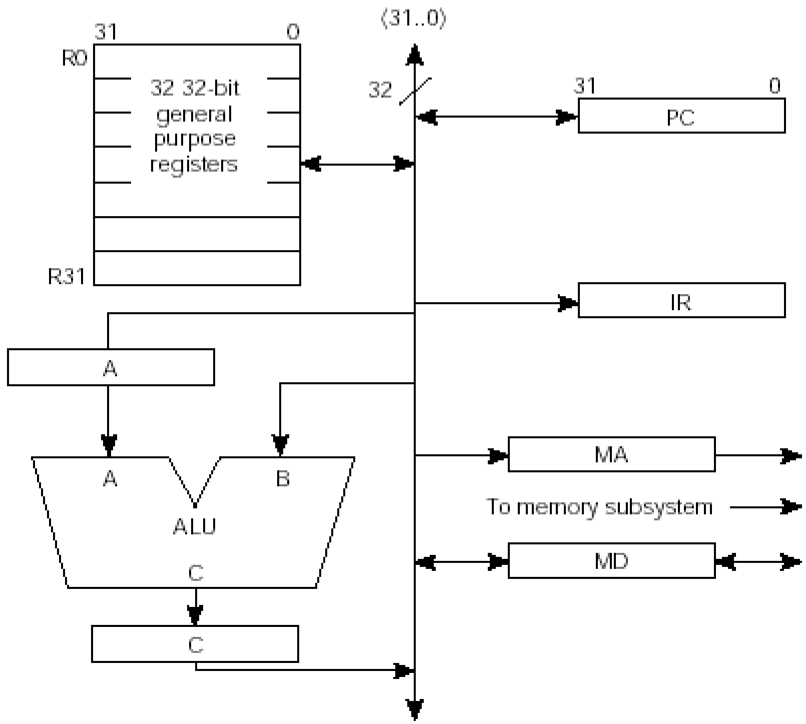
1-BUS Multi-Cycle MIPS

- 1-Bus Multi-Cycle Data-Path



Data-Path – Control Unit – Memory Interfacing

1-BUS SRC Multi-Cycle MIPS



1-BUS SRC (simple RISC Computer) Block Diagram [2]

add \$rd, \$rs, \$rt

Abstract RTL:

IF: IR ← M[PC], PC ← PC + 4;
Add: RF[rd] ← RF[rs] + RF[rt];

Concrete RTL:

IF: T0 → MA ← PC, C ← PC + 4;
T1 → MD ← M[MA], PC ← C;
T2 → IR ← MD;

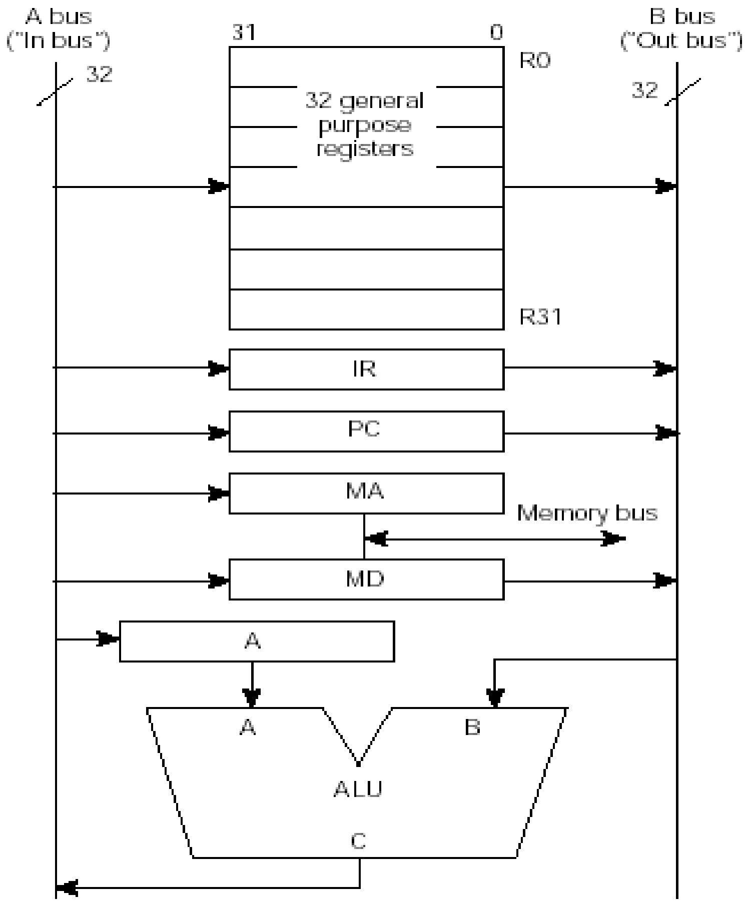
Ex: T3 → A ← RF[rs];
T4 → C ← A + RF[rt];
T5 → RF[rd] ← C

- Special ALU operation for PC + 4 in T0
- add takes 3 concrete RTs (T3, T4, T5)

ALU connected to the BUS through A and C registers

One bus connecting most registers allows many different RTs, but only one at a time → replaces multiplexors

2-BUS SRC Multi-Cycle MIPS



2-BUS SRC – Block Diagram [2]

- Bus A carries data going into registers
- Bus B carries data coming from registers
- ALU function C = B (Pass B)
    - is used for all simple register transfers

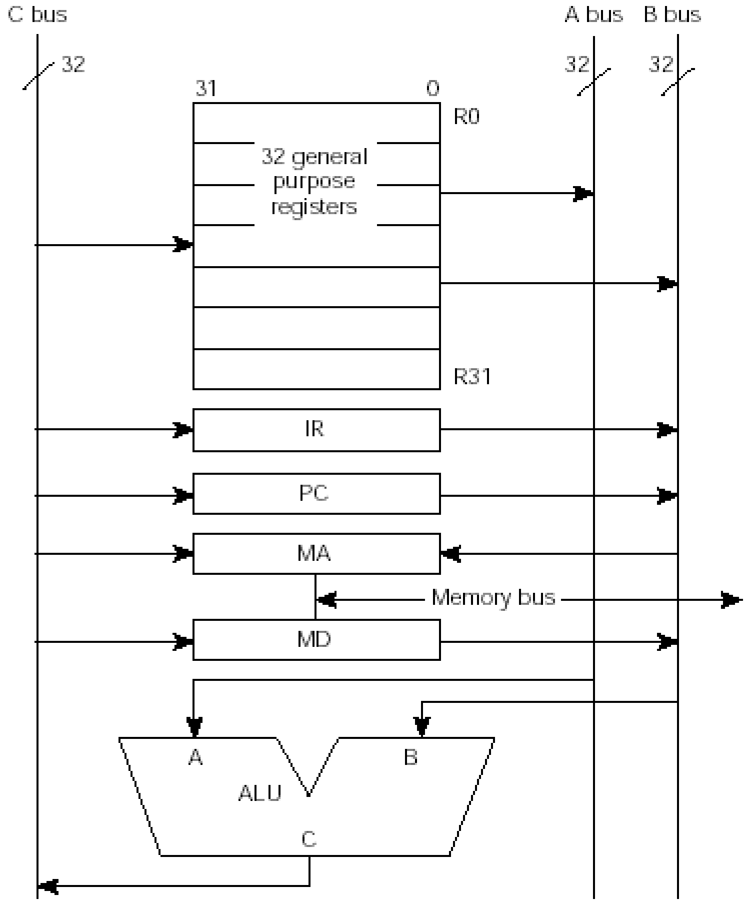
add $rd, $rs, $rt

Concrete RTL:

IF: T0 → MA ← PC;
T1 → PC ← PC + 4, MD ← M[MA];
T2 → IR ← MD;

Ex: T3 → A ← RF[rs];
T4 → RF[rd] ← A + RF[rt];

3-BUS SRC Multi-Cycle MIPS



3-BUS SRC – Block Diagram [2]

- A-bus is ALU operand 1
- B-bus is ALU operand 2
- C-bus is ALU output
- Note: MA input connected to the C and B-buses

add \$rd, \$rs, \$rt

Concrete RTL:

IF: T0 → MA ← PC, MD ← M[MA], PC ← PC + 4
T1 → IR ← MD;

Ex: T2 → RF[rd] ← RF[rs] + RF[rt];

- In step T0:
  - PC moves to MA over bus B and
  - goes through the ALU (INC 4 operation)
  - to reach PC again by way of bus C
- PC must be edge-triggered
- MA must be a transparent latch – the address is propagated to the memory unit in T0

Problems – Homework

- Implement other instructions for the Mux based multi-cycle MIPS CPU and for the bus based MIPS CPUs (1, 2 or 3 busses)
  - 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 Mux based multi-cycle MIPS CPU and for the bus based MIPS CPUs (1, 2 or 3 busses)
  - LWR, SWR (sums two registers to obtain the memory address)
  - LWA, SWA (uses a single register to obtain the memory address)

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".