Computer Organization and Software Systems

BITS Pilani
Pilani Campus

LAKSHMIKANTHA G C
WILP & Department of CS & IS

Webinar 2

MIPS-Multicycle Implementation

BITS Pilani

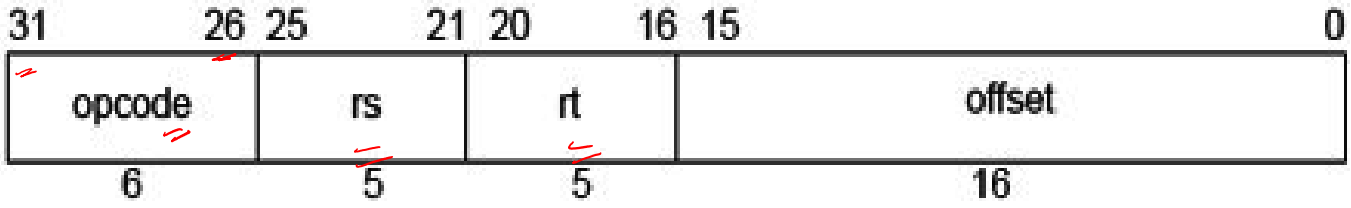
Pilani Campus

Session 6…

- MIPS Instruction set architecture
- Datapath for
  - fetching instructions
  - updating program counter
  - implementing R-type ALU operations
  - implementing data memory unit and the sign extension unit
  - datapath for implementing branch instructions
- Effect of control signals

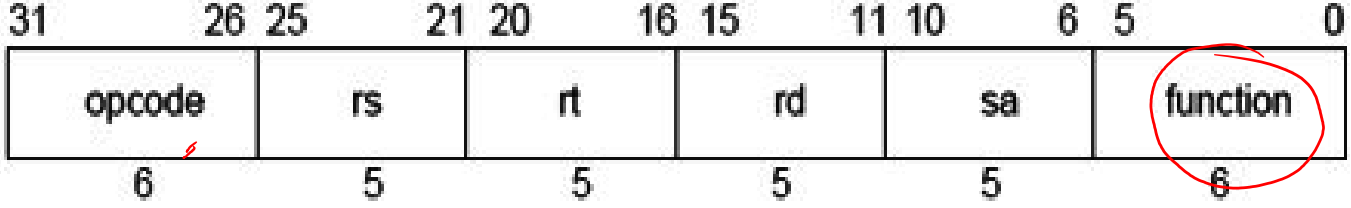
Fig.

I-Type (Immediate).



J-Type (Jump).

R-Type (Register).



Data Path

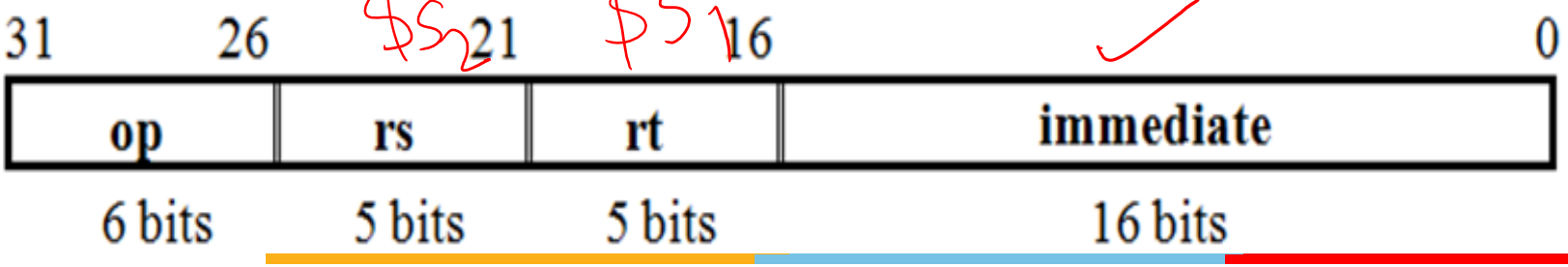
Effect of ALUop

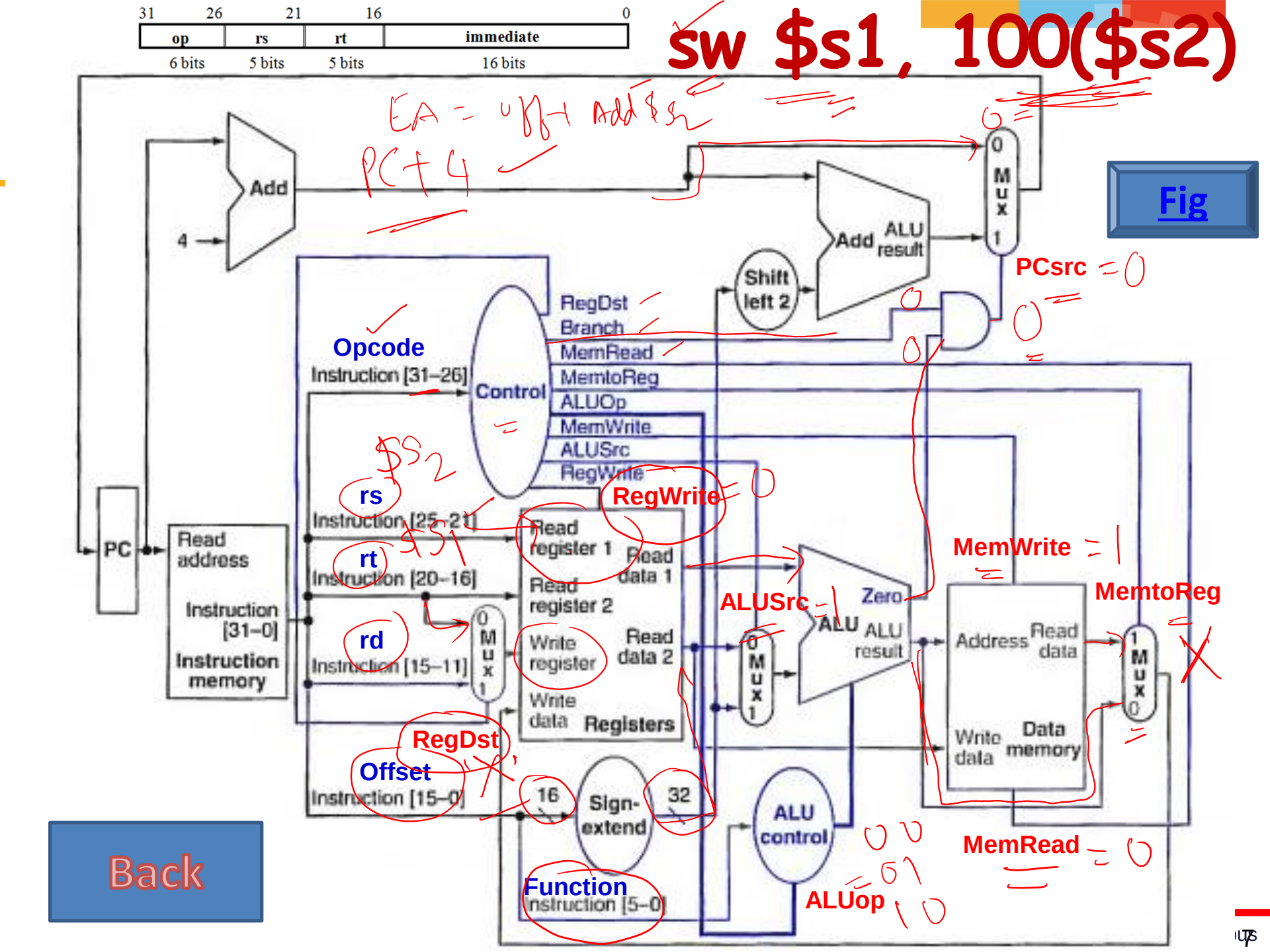
BACK

ALUop	Instructions
00	lw, sw
01	Beq
10	add, sub, and, or, slt

Execution steps: sw $s1, 100($s2)

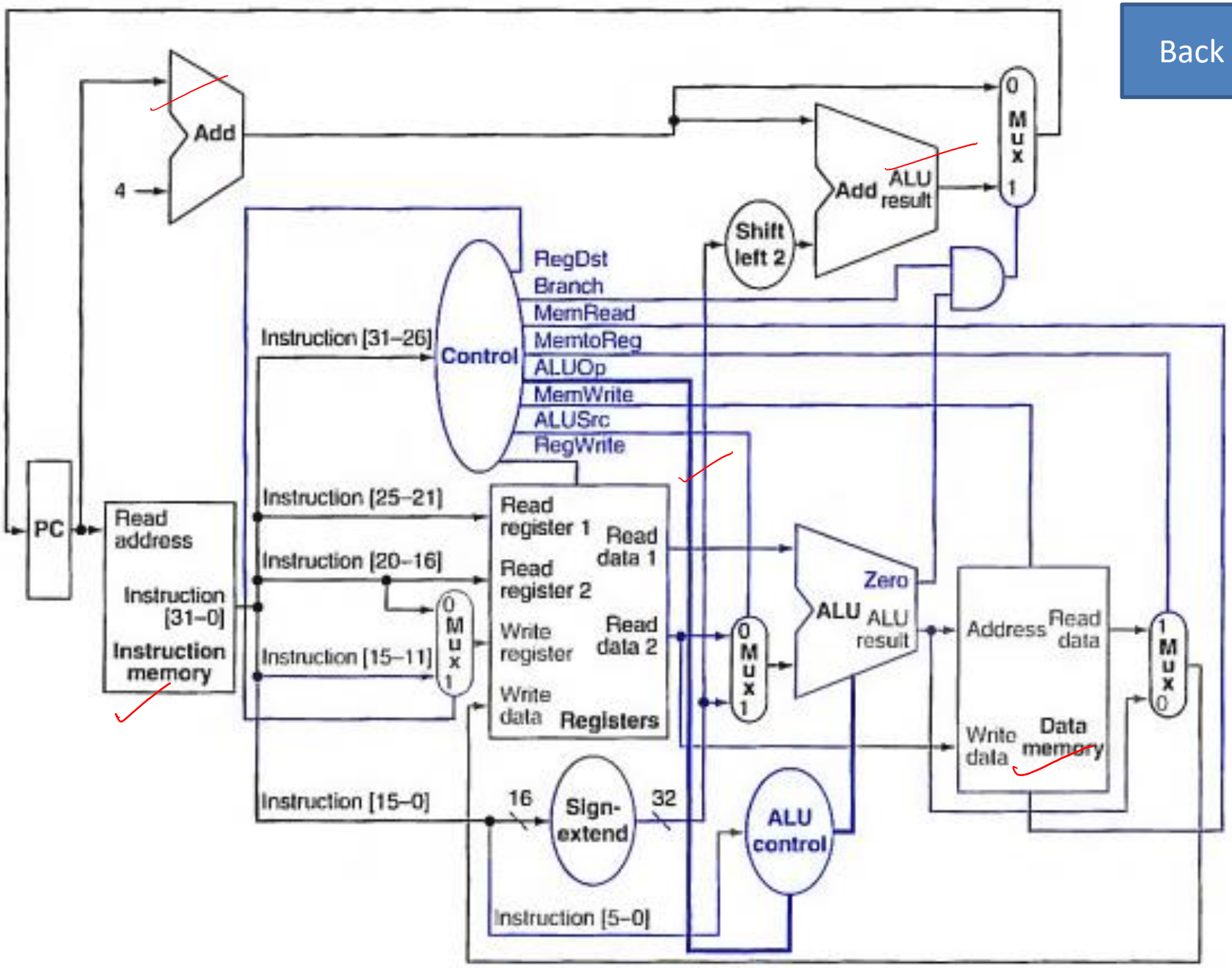
1. The instruction is fetched, and the PC is incremented
2. A register $s2 and $s1 value is read from the register file
3. The ALU computes the sum of the value read from the register s2 and the sign extended lower 16 bits of the instruction
4. the sum from the ALU is used as the address for the data memory and writes the contents of $s1 on to memory




31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
sw $s1, 100($s2)
EA = offset + Add $s2
PC + 4
Add
4
Add
ALU result
Shift left 2
M u x
0
1
PCsrc = 0
Fig
Opcode
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
$s2
rs
Instruction [25–21]
RegWrite = 0
PC
Read address
Instruction [31–0]
Instruction memory
rt
Instruction [20–16]
rd
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
MemWrite = 1
ALUSrc = 1
Zero
ALU
ALU result
MemtoReg
Address
Read data
Write data
Data memory
RegDst
Offset
Instruction [15–0]
16
Sign-extend
32
ALU control
MemRead = 0
Function
Instruction [5–0]
ALUop
00
01
10
Back

Home Work:
Execution steps:beq $s1,$s2, offset

1. The instruction is fetched, and the PC is incremented
2. Two registers $s1 and $s2 are read from the register file
3. The ALU performs a subtract on the data values read from the register file. The value of PC + 4 is added to the sign extended, lower 16 bits of the instruction (offset) sifted left by two; the result is the branch target address
4. The zero result from the ALU is used to decide which adder result to store into the PC

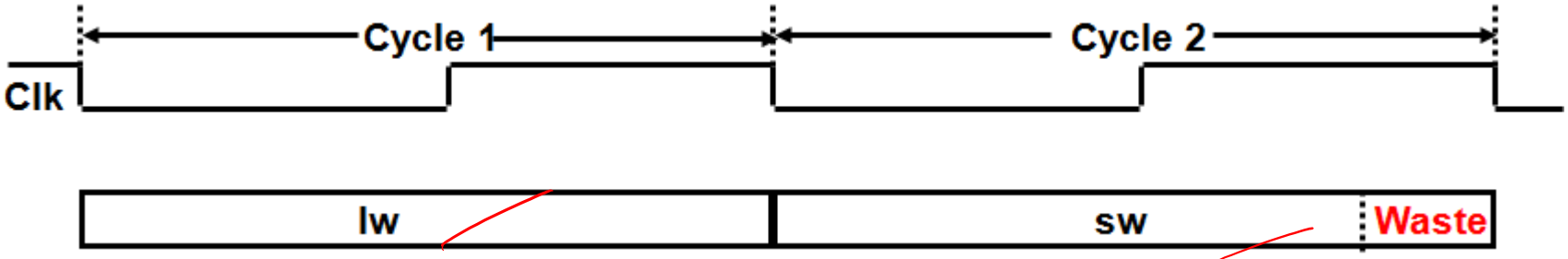

Add
4
Add
ALU result
Shift left 2
M u x
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory

Back

Single Cycle Advantages & Disadvantages

CPI = 1

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instruction
- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle but is simple and easy to understand



Multicycle datapath approach

- also called multiple clock cycle implementation
- an instruction is executed in multiple clock cycles
- Advantage : hardware sharing and ability to allow instructions to take different numbers of clock cycles
- Break up instructions into steps where each step takes a cycle while trying to
  - balance the amount of work to be done in each step
  - restrict each cycle to use only one major functional unit
- Not every instruction takes the same number of clock cycles

Single cycle vs multicycle

Single Cycle

- Single cycle:
  - Two memory units – one for instruction and one for data
  - one ALU and two adders

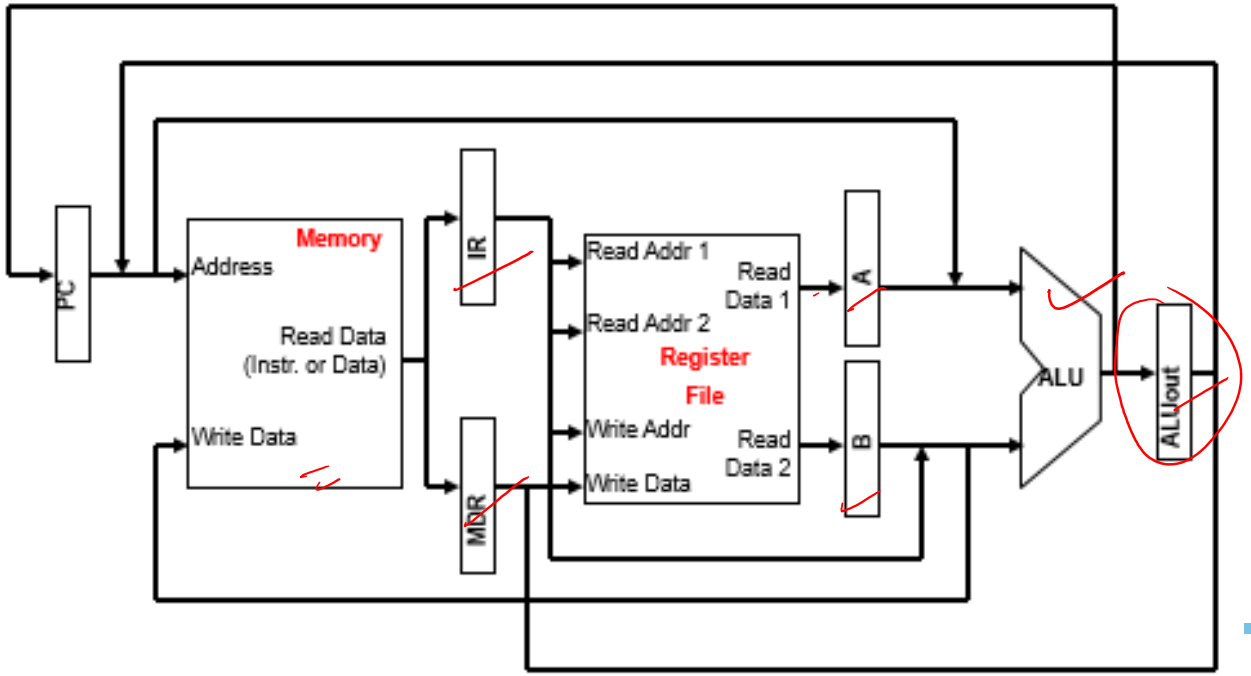
Multi Cycle

- Multicycle:
  - A single memory unit for both instructions and data
    - only one memory access per cycle
  - only a single ALU
    - only one ALU operation per cycle
  - one or more registers are added after every major functional unit to hold the output of that unit until the value is used in a subsequent clock cycle

Temporary registers

Back

- IR → Instruction Register
- MDR → Memory Data Register
- A,B→ used to hold the register operand values read from the register file
- ALUout → holds the output of the ALU



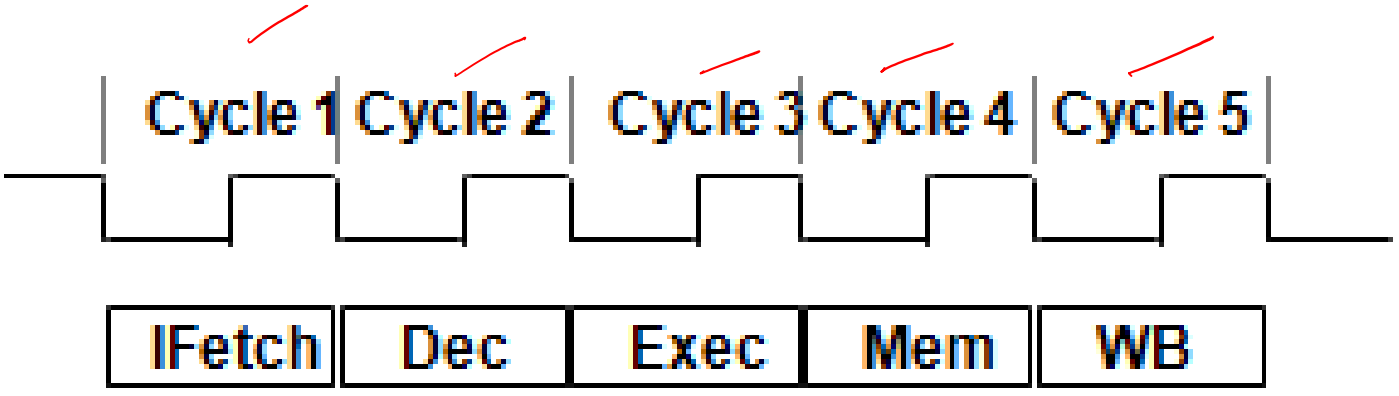
Contd…

Important note:

1. all data that is used in subsequent clock cycles must be stored in inter stage registers
2. Data used by subsequent instruction is stored in programmer visible registers (i.e., register file, PC, or memory)
3. All the registers except IR do not need a write control signal
4. Expand the multiplexer units

The Five Steps

- Step 1 : IF (Instruction Fetch)
- Step 2 : ID (Instruction Decode)
- Step 3 : EX (Execute)
- Step 4 : MEM (Memory)
- Step 5 : WB (Write Back)



Step1: IF (Instruction Fetch)

- Instruction Fetch and Update PC

  IR ← Memory [PC]

  PC ← PC + 4

Step 2: ID (Instruction Decode)

- Following operations are performed

(1) – read two registers corresponding to rs and rt fields

A ← Reg [IR [25:21]] → rs

B ← Reg [IR [20:16]] → rt

(2) – compute branch target address with ALU

ALUout ← PC + (sign-extend (IR[15-0]) << 2)

BEQ

Step 3: EX (Execute)

- Four operations are possible

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B)

   PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: MEM (Memory)

- Memory access or R type instruction completion step
  1. Memory Access

     MDR ← Memory [ALUout]

     or

     Memory [ALUout] ← B
  2. R-Type Instruction Completion

     Reg [IR [15: 11]] ← ALUout

Step 5: WB (Write Back)

- Memory read completion step
  - Reg [IR [20 : 16]] ← MDR

R-Type Instruction: add $S1, $S2, $S3

Step1: IFetch

IR ← Memory [PC]

PC ← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields

A ← Reg [IR [25:21]]

B ← Reg [IR [20:16]]

– compute branch target address with ALU

ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B

2. R-Type Instruction Completion

   Reg [IR [15: 11]] ← ALUout

Step 5: WB

Memory read completion step

– Reg [IR [20 : 16]] ← MDR

CPI = 4 CC

lw Instruction: lw $S1, 100($S2)

Step1: IFetch

IR ← Memory [PC]
PC ← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields
A ← Reg [IR [25:21]]
B ← Reg [IR [20:16]]
– compute branch target address with ALU
ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference
   ALUout ← A + sign-extend (IR[15:0])
2. Arithmetic and logical instruction
   ALUout ← A op B
3. Branch
   if (A == B) PC ← ALUout
4. Jump
   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access
   MDR ← Memory [ALUout]
   or
   Memory [ALUout] ← B
2. R-Type Instruction Completion
   Reg [IR [15: 11]] ← ALUout

Step 5: WB

Memory read completion step
– Reg [IR [20 : 16]] ← MDR

CPI = 5 CC

sw Instruction: sw $S1, 100($S2)

Step1: IFetch

IR ← Memory [PC]
PC ← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields
  A ← Reg [IR [25:21]]
  B ← Reg [IR [20:16]]
– compute branch target address with ALU
  ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference
   ALUout ← A + sign-extend (IR[15:0])
2. Arithmetic and logical instruction
   ALUout ← A op B
3. Branch
   if (A == B) PC ← ALUout
4. Jump
   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access
   MDR ← Memory [ALUout]
   or
   Memory [ALUout] ← B
2. R-Type Instruction Completion
   Reg [IR [15: 11]] ← ALUout

Step 5: WB

Memory read completion step
– Reg [IR [20 : 16]] ← MDR

CPI = 4 CC

beq Instruction: beq $S1, $S2, 100

Step1: IFetch

IR ← Memory [PC]
PC ← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields
A ← Reg [IR [25:21]]
B ← Reg [IR [20:16]]
– compute branch target address with ALU
ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference
   ALUout ← A + sign-extend (IR[15:0])
2. Arithmetic and logical instruction
   ALUout ← A op B
3. Branch
   if (A == B) PC ← ALUout
4. Jump
   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access
   MDR ← Memory [ALUout]
   or
   Memory [ALUout] ← B
2. R-Type Instruction Completion
   Reg [IR [15: 11]] ← ALUout

Step 5: WB

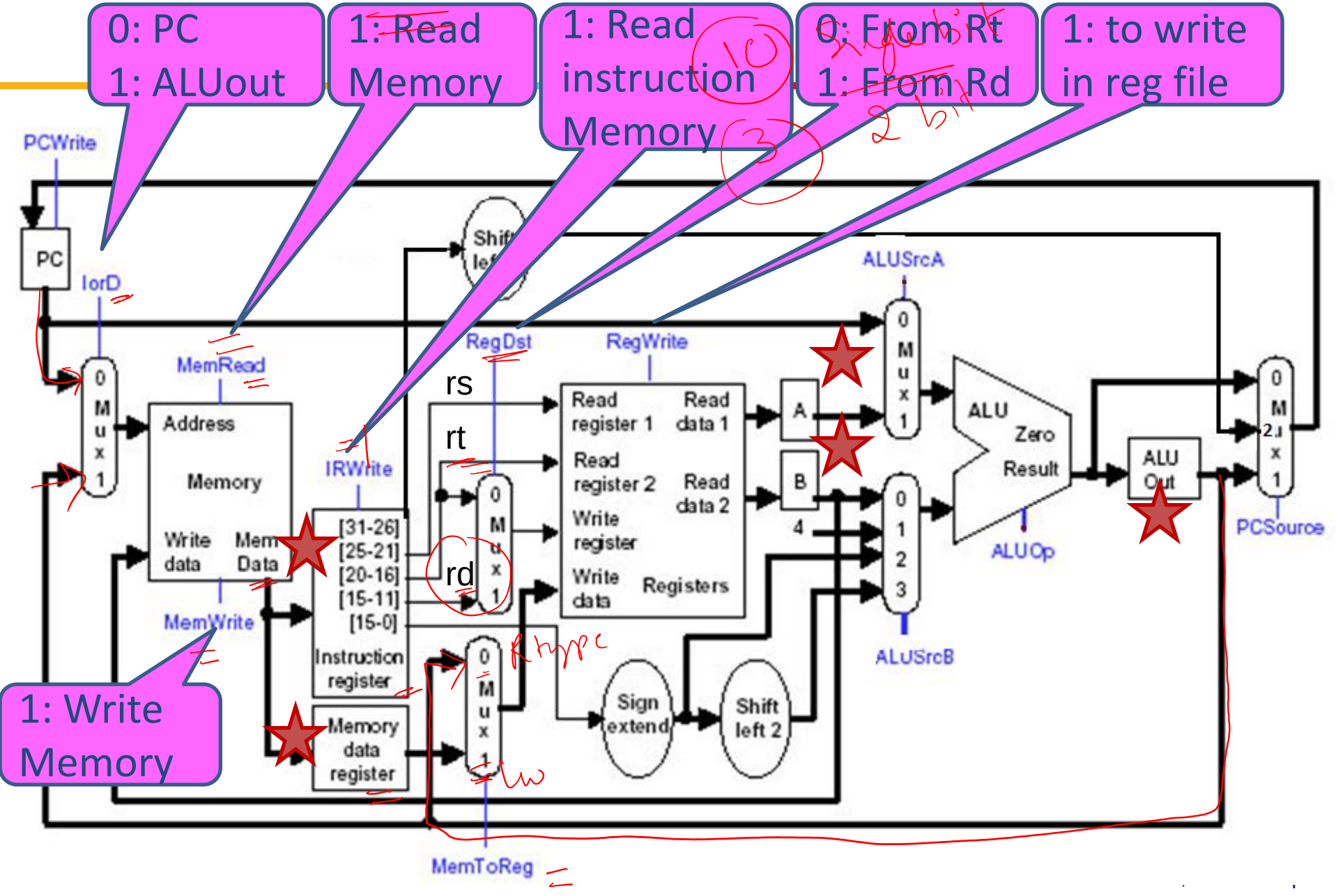
Memory read completion step
– Reg [IR [20 : 16]] ← MDR

CPI = 3CC

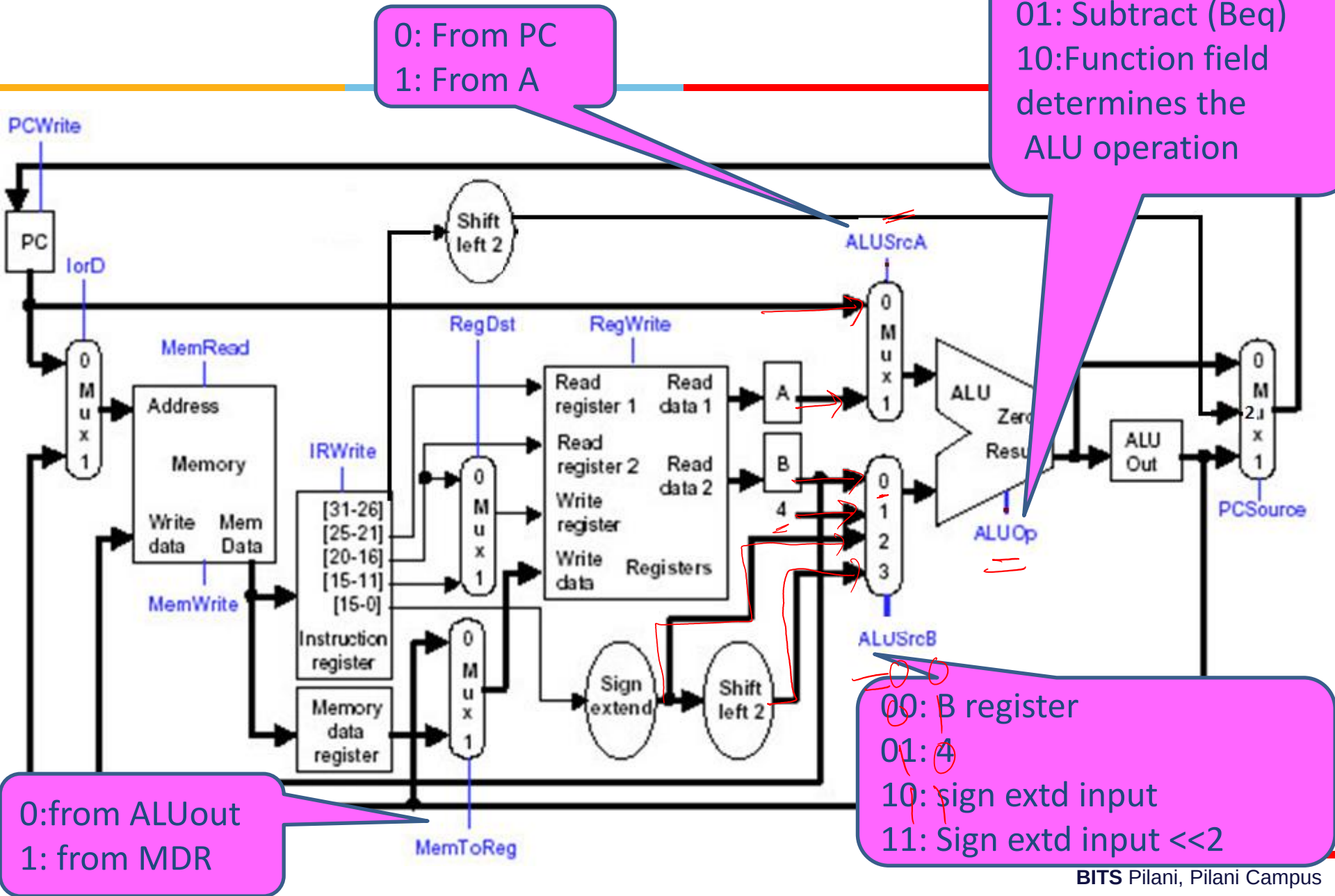
Summary

- R-Type: Require four cycles, CPI =4
  IF, ID, EX, WB
- Loads Require five cycles, CPI = 5
  IF, ID, EX, MEM, WB
- Store: Require four cycles, CPI = 4
  IF, ID, EX, MEM
- Branch: Require three cycles, CPI = 3
  IF, ID, EX

Complete Data Path (1/3)

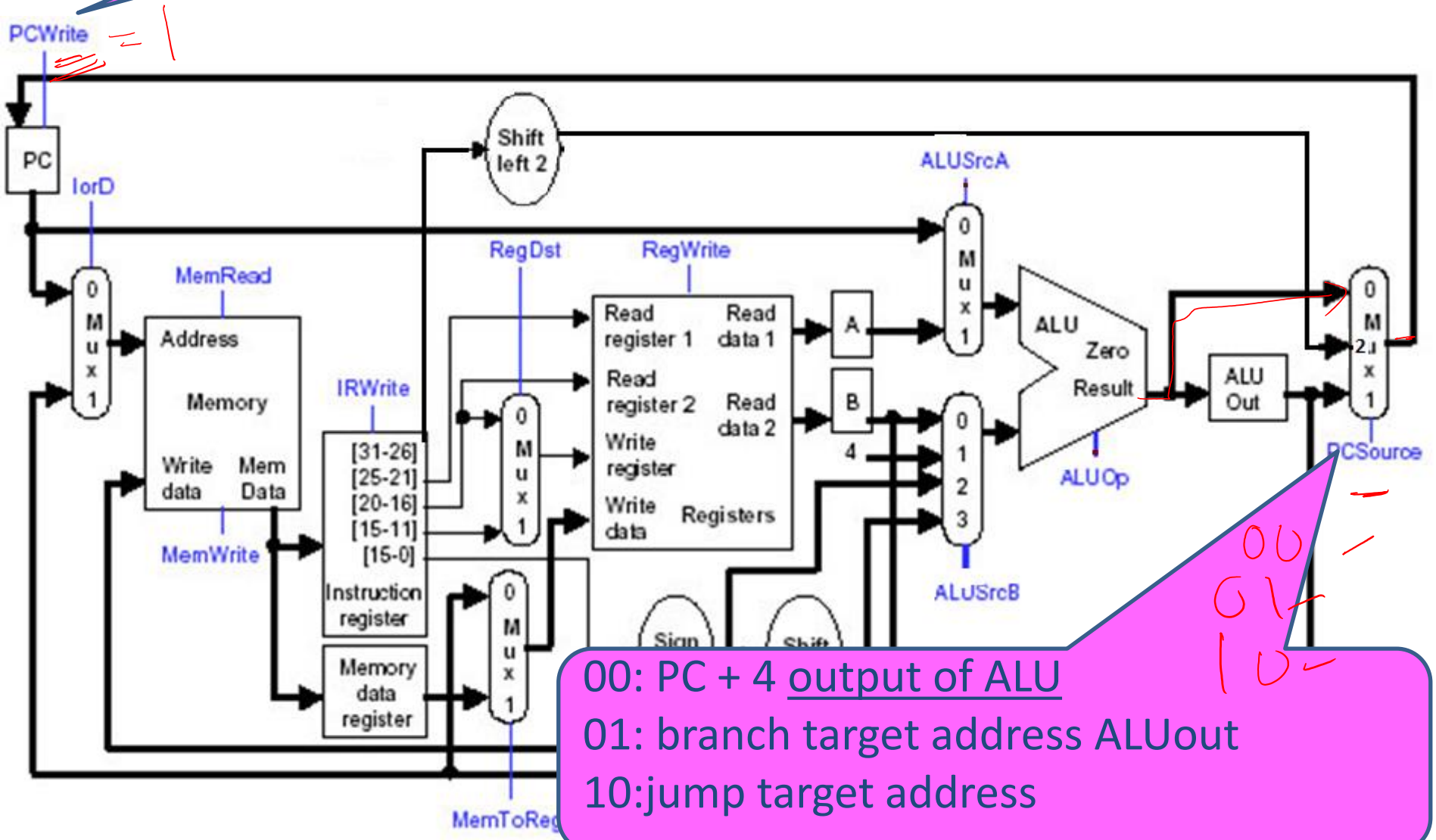


Complete Data Path (2/3)



Complete Data Path (3/3)

1: PC gets updated



00: PC + 4 output of ALU

01: branch target address ALUout

10:jump target address

Actions of the 1 bit control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2 bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are 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

Back

First Step

innovate achieve lead

FIG

- **Instruction Fetch : IFetch**
  - IR ← Memory [PC]:
    - 1 – MemRead
    - 1 – IRWrite
    - 0 – IorD
  - PC ← PC + 4 :
    - 0 - ALUSrcA
    - 01 - ALUSrcB
    - - ALUop
    - - PCSource
    - - PCWrite

0: PC
1: ALUout

0: First operand is in PC
1: First operand is in register A

Second operand
00: A register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

Contd…

innovate achieve lead

Fig

- **Instruction Fetch :**
  - IR ← Memory [PC]:
    - 1 – MemRead
    - 1 – IRWrite
    - 0 – IorD
  - PC ← PC + 4 :
    - 0 - ALUSrcA
    - 01 - ALUSrcB
    - 00 - ALUop
    - 00 - PCSource
    - 1 - PCWrite

00: Add operation
01: Subtract Operation
10:Function field of the instruction determines the ALU operation

00: PC + 4 output of ALU
01: branch target address ALUout
10:jump target address

Second Step

Fig

Instruction decode and re

- read two registers fields
- compute branch

A ← Reg [IR [2 21]]

B ← Reg [IR 0:16]]

ALUout ← P + (sign-e (IR[15-0]) << 2)

0: First operand is PC
1: First operand is register

Second operand
00: register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

0 ALUSrcA = 0

11 ALUSrcB = 11

00 ALUop = 00

00: Add
01: Subtract
10:Function field determines the ALU operation

Step 3

Execution, memory address computation, or branch completion

- Four operations are possible

1. Memory reference generation

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 3: Contd…

innovate achieve

Fig

1.Memory reference

ALUout ← A + sign-extend (IR[15:0])

- **1** – ALUSrcA

0: First operand is PC
1: First operand is register A

- **10** – ALUSrcB

Second operand
00: register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

- **00** – ALUop

00: Add operation
01: Subtract Operation
10:Function field of the instruction determines the ALU operation

Step 3: Contd…

Fig

2. Arithmetic and logical instruction

ALUout ← A op B

1 - ALUSrcA

0: First operand is PC
1: First operand is register

00 - ALUSrcB

00: register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

10 - ALUop

00: Add operation
01: Subtract Operation
10:Function field of the instruction determines the ALU operation

Step 3: Contd…

3. Branch

if (A == B) PC ← ALU

1 - ALUSrcA

00 - ALUSrcB

01 - ALUop

1 - PCWrite

01 - PCSource

0: First operand is PC
1: First operand is register

Fig

00: register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

00: Add operation
01: Subtract Operation(beq)
10:Function field of the instruction determines the ALU operation

00: PC + 4 output of ALU
01: branch target address ALUout
10:jump target address

Step 3: Contd…

innovate achieve lead

Fig

4. Jump

PC ← { PC[31:28], (IR [25:0] << 2)

10 – PCSource

1 – PCWrite

00: PC + 4 output of ALU
01: branch target address ALUout
10:jump target address

Step 4

- Memory access or R type instruction completion step
  - Two operations are possible
    - Memory reference

      MDR ← Memory [ALUout]

      or

      Memory [ALUout] ← B
    - Arithmetic and logical instruction

      Reg [IR [15: 11]] ← ALUout

Step 4 Contd…

FIG

Memory reference

lw

MDR ← Memory [ALUout]

1 MemRead

1 IorD

sw

Memory [ALUout] ← B

1 MemWrite

1 IorD

Step 4 contd…

Fig

Arithmetic and logical instruction

Reg [IR [15: 11]] ← ALUout

1 RegDst

1 RegWrite

0 MemtoReg

Step 5

Fig

Write back or Memory read completion step

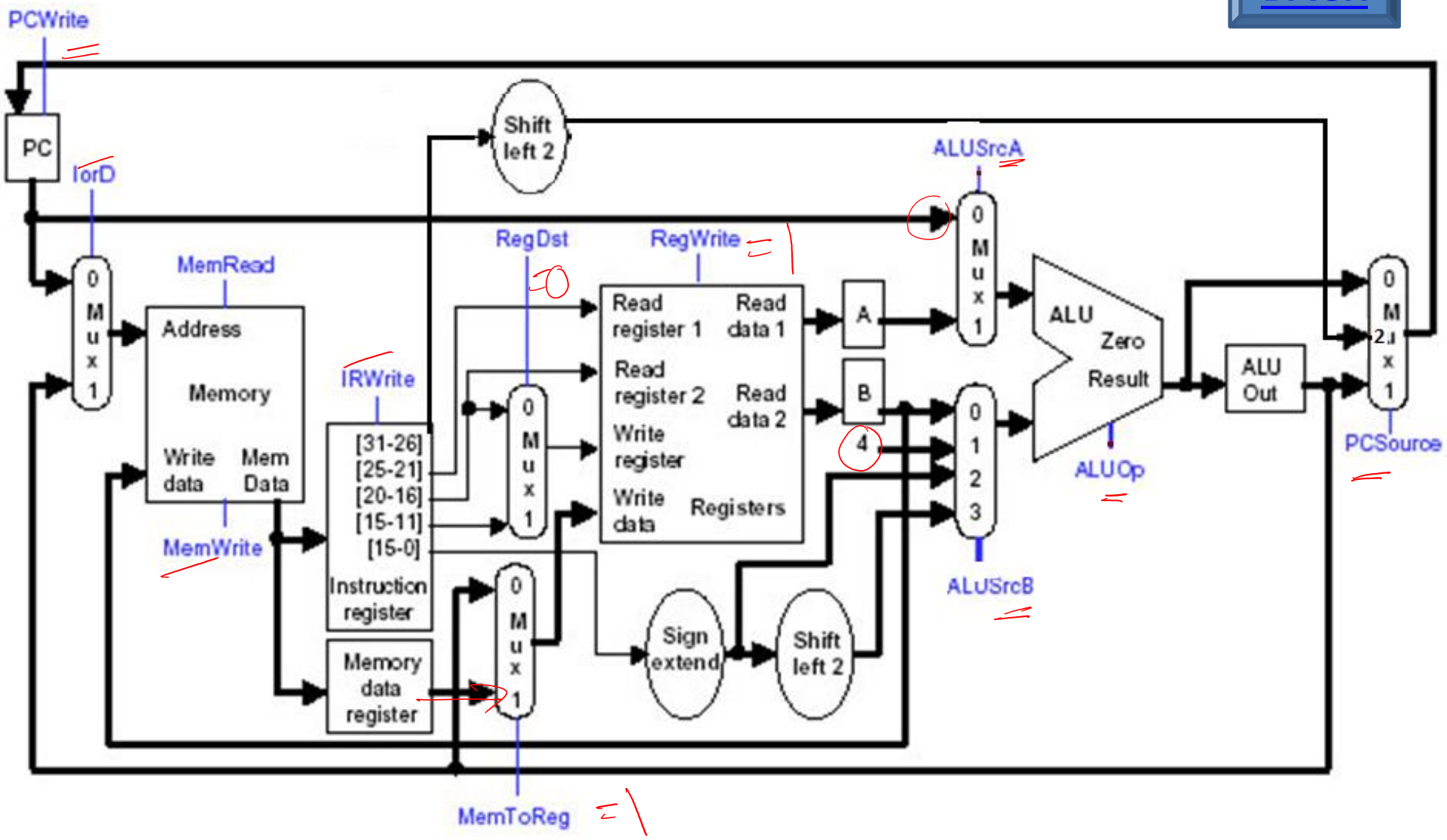
– Reg [IR [20 : 16]] ← MDR

MemtoReg 1

RegWrite 1

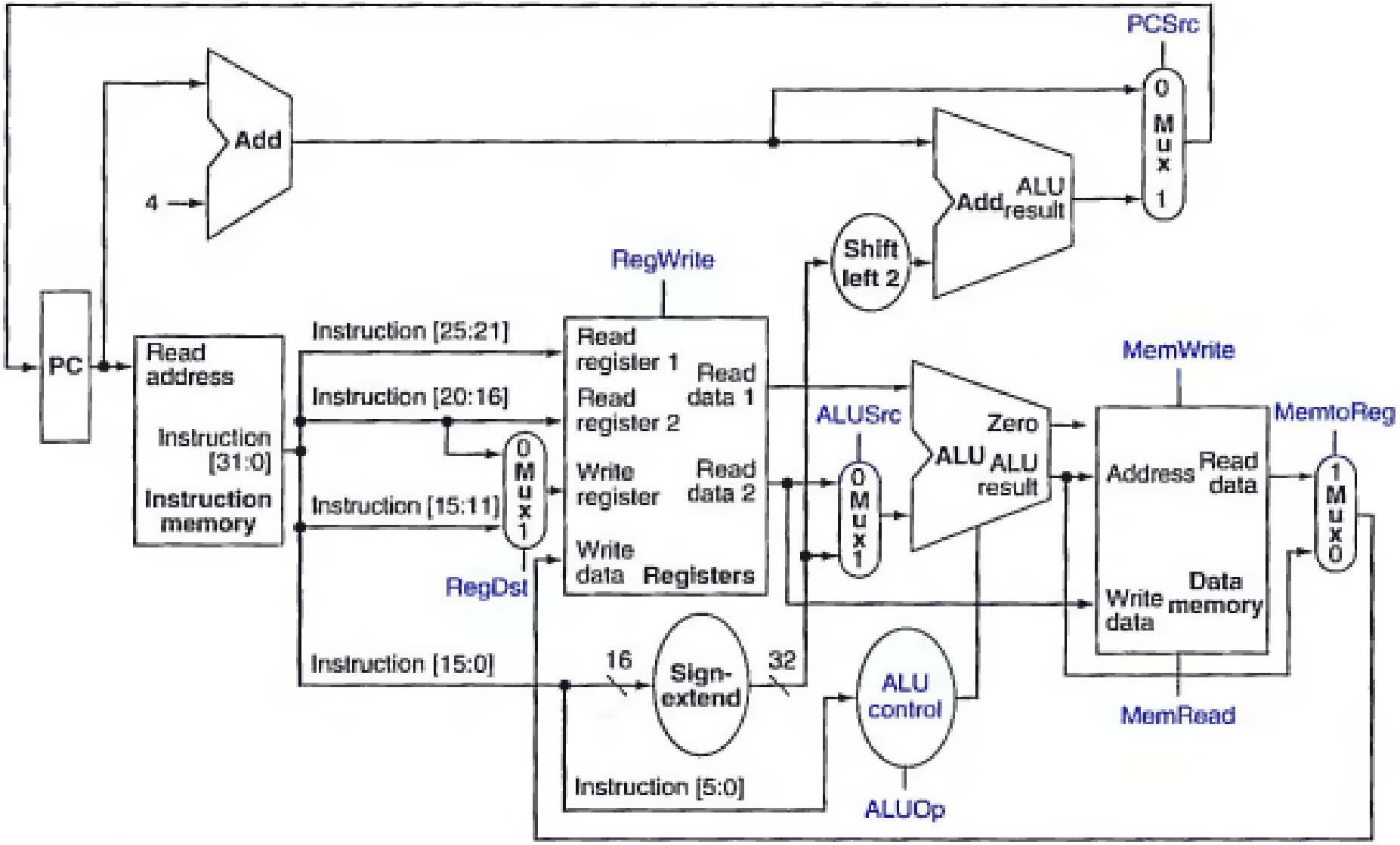
RegDst 0

BACK


PCWrite
PC
IorD
Shift left 2
ALUSrcA
RegDst
RegWrite
MemRead
Address
Memory
Write data
Mem Data
MemWrite
IRWrite
[31-26]
[25-21]
[20-16]
[15-11]
[15-0]
Instruction register
Memory data register
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
A
B
4
ALU
Zero
Result
ALUOp
ALU Out
ALUSrcB
Sign extend
Shift left 2
PCSource
MemToReg
0
M
u
x
1
0
1
2
3

Summary

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]),2'b00}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		



The datapath with all necessary multiplexors and all control lines identified.