Corso di Laurea in Informatica
Architettura degli elaboratori
a a 2019-2020

Architettura degli Elaboratori
2019-2020

Datapath multi ciclo

Slide: Claudia Raibulet (commento in video Moodle: Claudio Ferretti)

Datapath

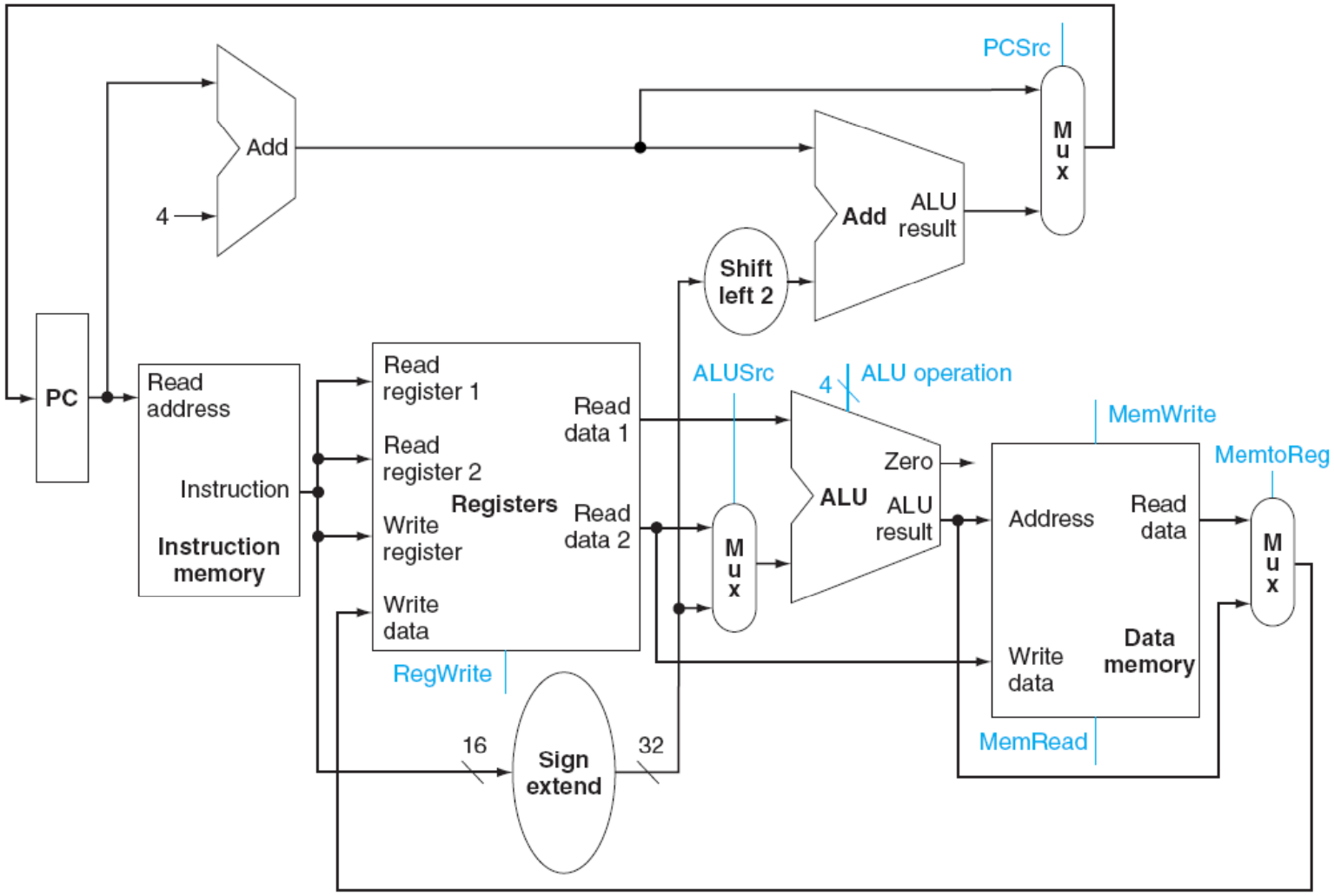


FIGURE 5.11 The simple datapath for the MIPS architecture combines the elements required by different instruction classes. This datapath can execute the basic instructions (load/store word, ALU operations, and branches) in a single clock cycle. An additional multiplexor is needed to integrate branches. The support for jumps will be added later.

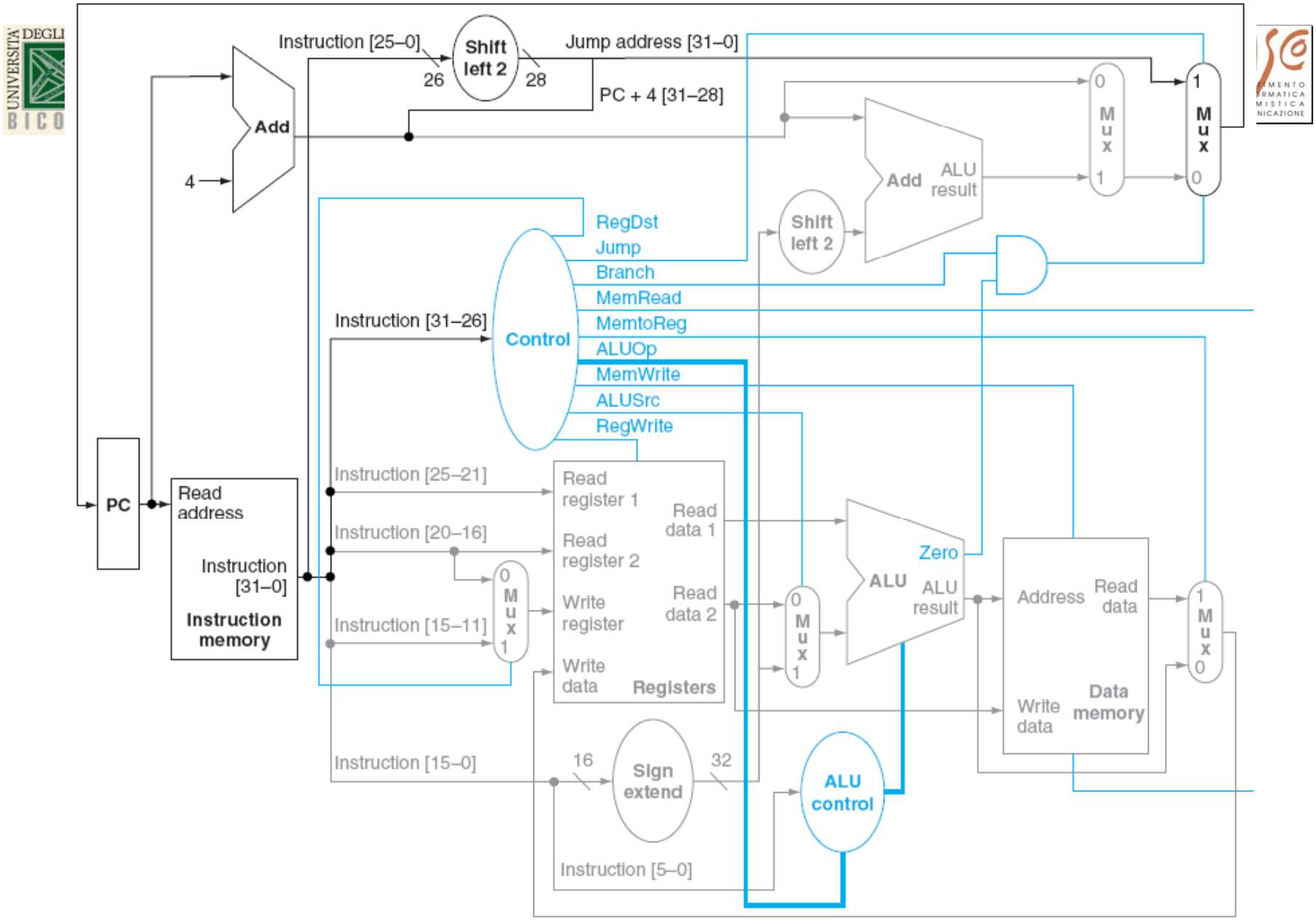


FIGURE 5.24 The simple control and datapath are extended to handle the jump instruction. An additional multiplexor (at the upper right) is used to choose between the jump target and either the branch target or the sequential instruction following this one. This multiplexor is controlled by the jump control signal. The jump target address is obtained by shifting the lower 26 bits of the jump instruction left 2 bits, effectively adding 00 as the low-order bits, and then concatenating the upper 4 bits of PC + 4 as the high-order bits, thus yielding a 32-bit address.

Svantaggi datapath singolo ciclo vs multi-ciclo

- Singolo ciclo
  - Ciclo di clock lungo (uguale al tempo necessario per eseguire l'istruzione piu' lunga)
  - Istruzioni potenzialmente piu' veloci sono rallentate
  - Unita' funzionali replicate (e.g., memoria, ALU)
- Multi-ciclo
  - Ciclo di lunghezza fissa piu' corto
  - Ogni istruzione viene eseguita in piu' cicli di clock
  - Istruzioni di tipo diverso – eseguite in un numero di cicli di clock diverso
  - Le unità funzionali vengono usate più volte durante lésecuzione della stessa istruzione in cicli di clock differenti -> meno replicazione
  - Si usano registri aggiuntivi per memorizzare i risultati parziali nell'esecuzione delle istruzioni

Datapath multi-ciclo

- Registri aggiuntivi:
  - memorizzano valori intermedi che vengono usati nel ciclo di clock successivo per continuare l'esecuzione della stessa istruzione:
  - IR - Instruction Register
  - MDR - Memory Data Register
  - A, B - Registri tra Register File e l'ingresso ALU
  - ALUout - L'output della ALU
- Riutilizzo di unita' funzionali:
  - ALU usata non solo per le operazioni aritmetico-logiche ma anche per calcolare l'indirizzo dei salti e incrementare il PC
  - Memoria usata sia per leggere le istruzioni che per leggere/scrivere i dati

Datapath multi-ciclo

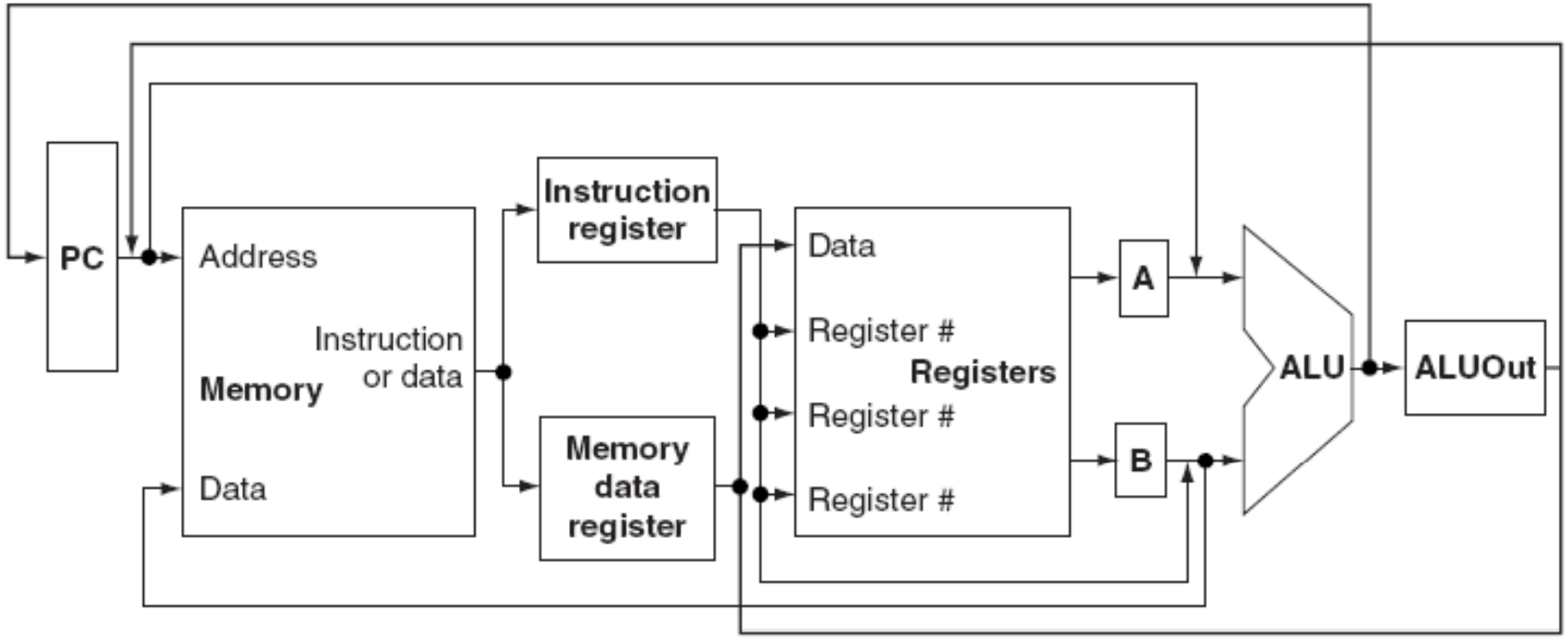
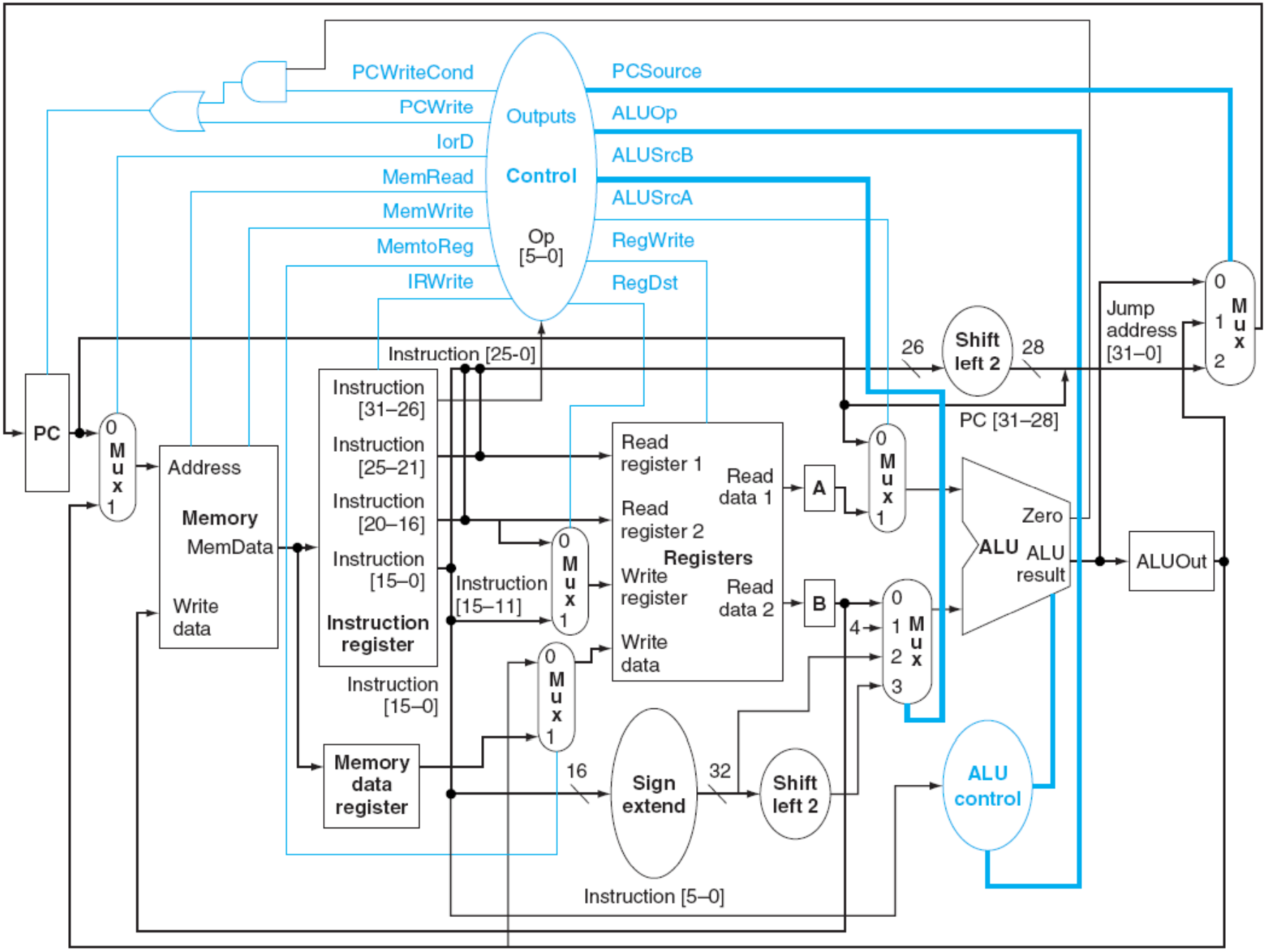


FIGURE 5.25 The high-level view of the multicycle datapath. This picture shows the key elements of the datapath: a shared memory unit, a single ALU shared among instructions, and the connections among these shared units. The use of shared functional units requires the addition or widening of multiplexors as well as new temporary registers that hold data between clock cycles of the same instruction. The additional registers are the Instruction register (IR), the Memory data register (MDR), A, B, and ALUOut.


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction [25-0]
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
0
1
2
3
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

Passi per eseguire le istruzioni

- Ogni istruzione si esegue in piu' passi
- Ogni passo si esegue in un ciclo di clock (abbastanza corto)
- Importante il bilanciamento della quantita' di lavoro in ogni passo
- Una unita' funzionale viene usata solo una volta durante lo stesso ciclo di clock
- Al termine di ogni ciclo di clock i valori intermedi vengono memorizzati nei registri addizionali e restano disponibili per il ciclo successivo

Passi per eseguire le istruzioni

- Fetch: fetch dell'istruzione E incremento PC
- Decode: decodifica dell'istruzione E lettura registri E calcolo dell'indirizzo per un eventuale branch
- Execute: eseguire le operazioni relative a R_type OPPURE calcolo di memoria OPPURE completa branch OPPURE completa jump
- Execute: completa R_type OPPURE accesso alla memoria
- Execute: scrittura registro (solo per l'istruzione lw)

- L'istruzione piu' lunga si esegue in 5 passi
- L'istruzione piu' corta si esegue in 3 passi

Passo 1: Fetch – per tutte le istruzioni

- Operazioni

IR <- M [PC]
PC <- PC+4

- Segnali di controllo

- Per leggere della memoria: MemRead
- Per scrivere IR: IRWrite
- Per indicare l'indirizzo da dove leggere dalla memoria: IorD
- Per incrementare il PC: ALUSrcA, ALUSrcB, ALUop
- Per salvare il nuovo valore del PC in PC: PCWrite

Passo 2: Decode – per tutte le istruzioni

- Operazioni

  A <- Reg[IR[25:21]]

  B <- Reg[IR[20:16]]

  ALUOut <- PC+(sign-extend (IR[15:0]) << 2

  - Segnali di controllo

  - Per il calcolo di un eventuale indirizzo di branch: ALUSrcA, ALUSrcB, ALUOp

Passo 3: Execute – per le istruzioni lw e sw

- Operazioni

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

  - Segnali di controllo
  - Per il calcolo dell' indirizzo di memoria per lw o sw: ALUSrcA, ALUSrcB, ALUOp

Passo 3: Execute – per le istruzioni R-Type aritmetico-logiche

- Operazioni

  ALUOut <- A op B

  - Segnali di controllo
  - Per il calcolo artimetico o logico: ALUSrcA, ALUSrcB, ALUOp

Passo 3: Execute – per le istruzioni beq

- Operazioni

  if A == B then PC <- ALUOut

  - Segnali di controllo

  - Per la comparazione tra A e B: ALUSrcA, ALUSrcB, ALUOp
  - Per scrivere PC: PCWriteCond, PCSource

Passo 3: Execute – per le istruzioni jump

- Operazioni

  PC <- (PC[31:28], IR[25:0]<<2)

  - Segnali di controllo
  - Per scrivere PC: PCWite, PCSource

Passo 4: Execute – per le istruzioni lw e sw

- Operazioni

  MDR <- M[ALUOut]

OPPURE

  M[ALUOut] <- B

- Segnali di controllo

- Per indicare l'indirizzo di memoria: IorD
- Per leggere dalla memoria (lw): MemRead
- Per scrivere nella memoria (sw): MemWrite

Passo 4: Execute – per le istruzioni aritmetico-logiche

- Operazioni

  Reg[IR[15:11]] <- ALUOut

  - Segnali di controllo

  - Per poter scrivere nel Register File: RegWrite
  - Per indicare il registro da scrivere: RegDest
  - Per scrivere il valore nel ALUOut: MemToReg

Passo 5: Execute – per l'istruzione lw

- Operazioni

  Reg[IR[20:16]] <- MDR

  - Segnali di controllo

  - Per poter scrivere nel Register File: RegWrite
  - Per indicare il registro da scrivere: RegDest
  - Per scrivere il valore dalla memoria: MemToReg

Tutte i passi delle istruzioni

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		

FIGURE 5.30 Summary of the steps taken to execute any instruction class. Instructions take from three to five execution steps. The first two steps are independent of the instruction class. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction class. The empty entries for the Memory access step or the Memory read completion step indicate that the particular instruction class takes fewer cycles. In a multicycle implementation, a new instruction will be started as soon as the current instruction completes, so these cycles are not idle or wasted. As mentioned earlier, the register file actually reads every cycle, but as long as the IR does not change, the values read from the register file are identical. In particular, the value read into register B during the Instruction decode stage, for a branch or R-type instruction, is the same as the value stored into B during the Execution stage and then used in the Memory access stage for a store word instruction.