Datapath multi-cycle

Claudia Raibulet
claudia.raibulet@unimib.it

Datapath

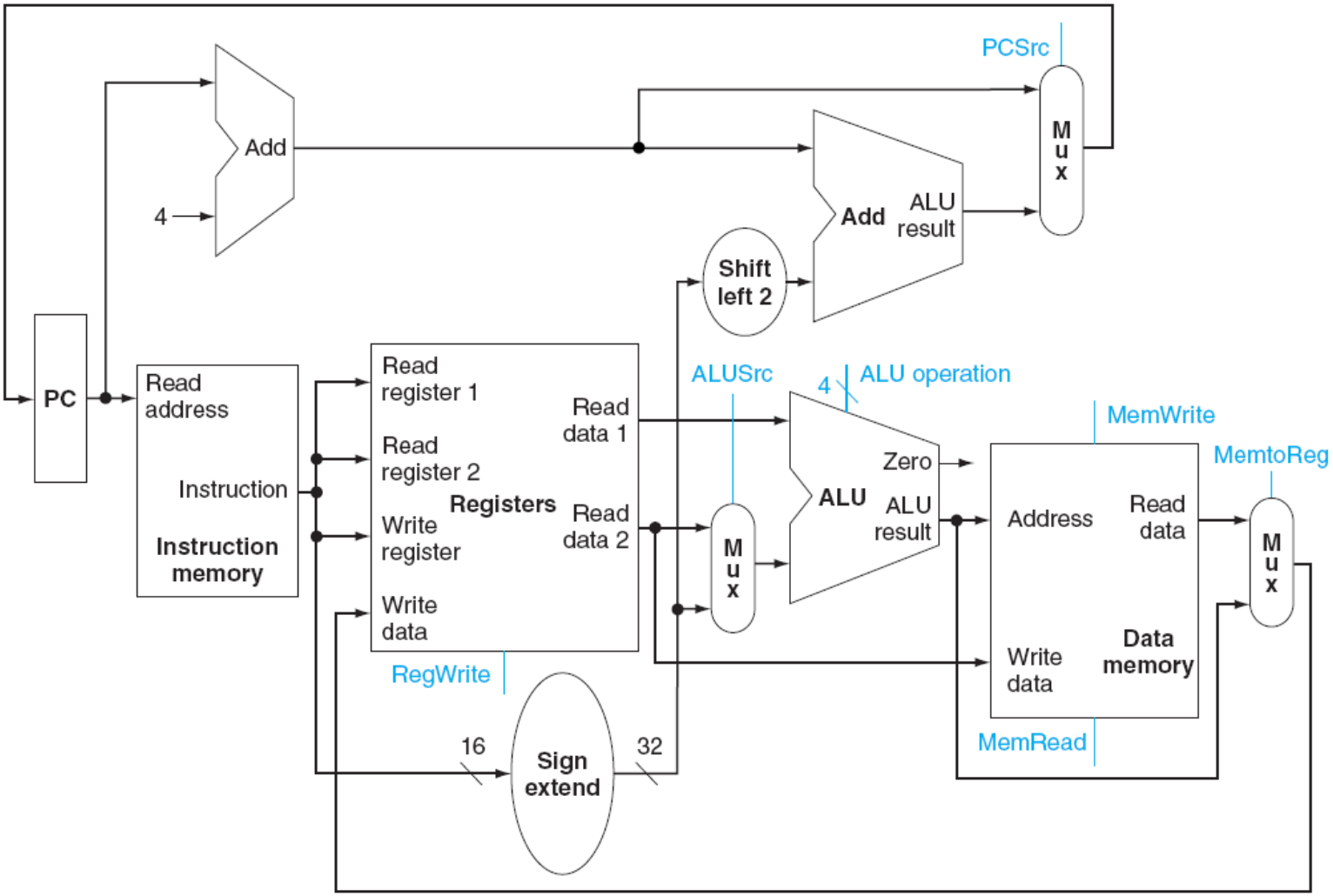


FIGURE 5.11 The simple datapath for the MIPS architecture combines the elements required by different instruction classes. This datapath can execute the basic instructions (load/store word, ALU operations, and branches) in a single clock cycle. An additional multiplexor is needed to integrate branches.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

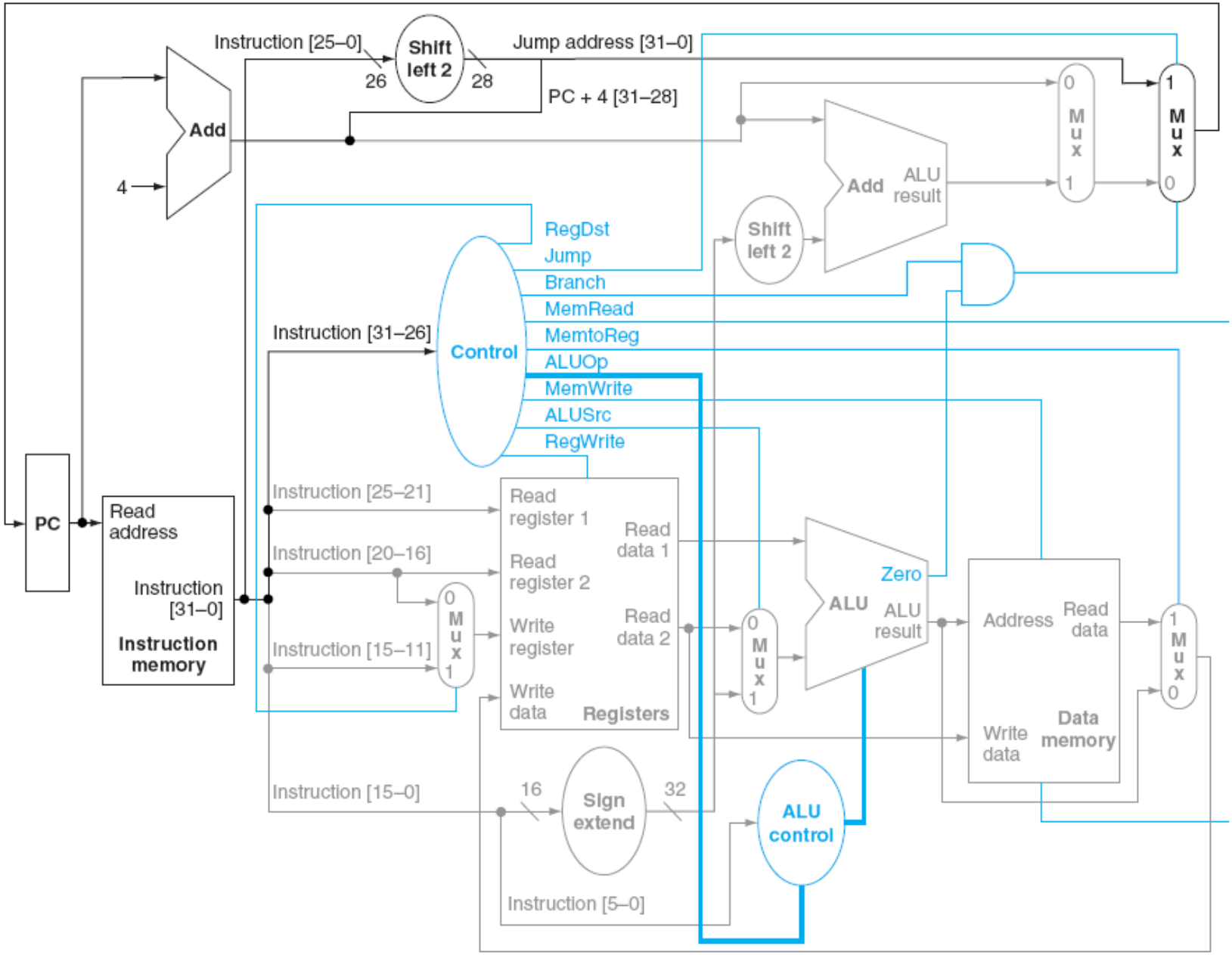


FIGURE 5.24 The simple control and datapath are extended to handle the jump instruction. An additional multiplexor (at the upper right) is used to choose between the jump target and either the branch target or the sequential instruction following this one. This multiplexor is controlled by the jump control 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.

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 l'esecuzione 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

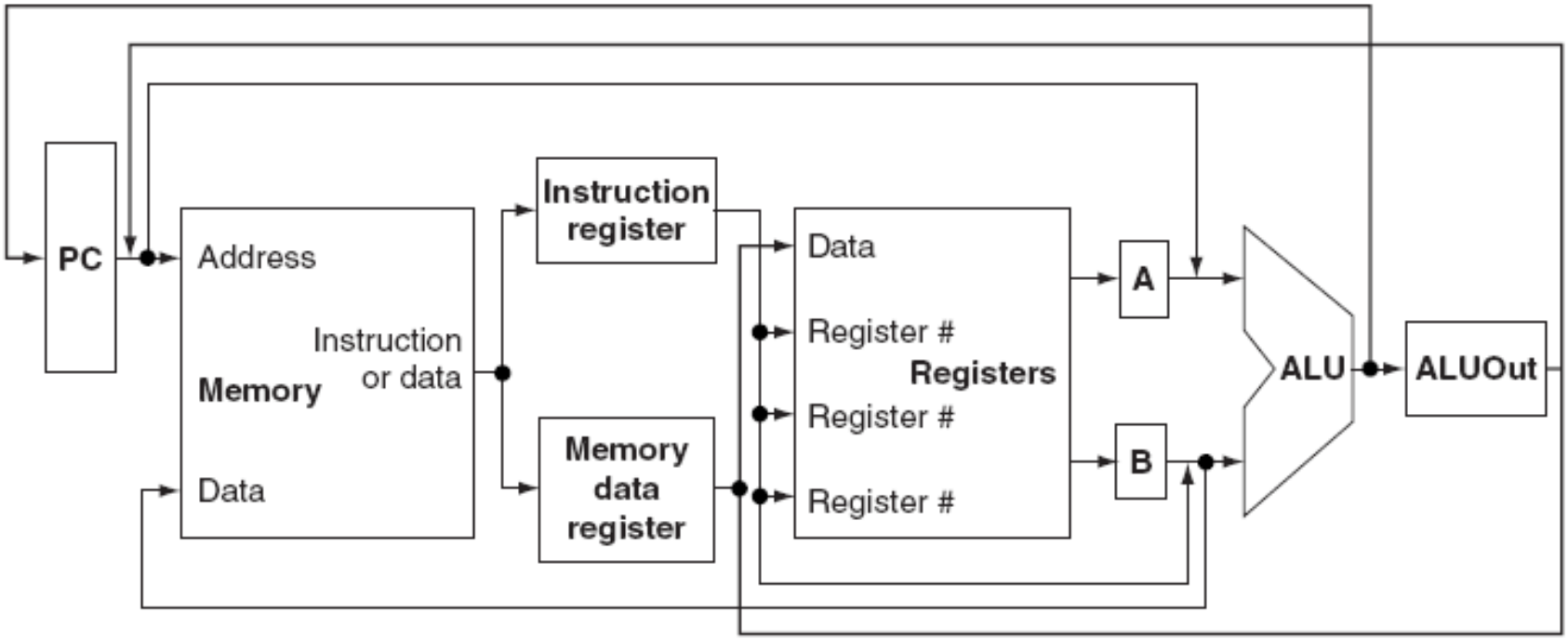
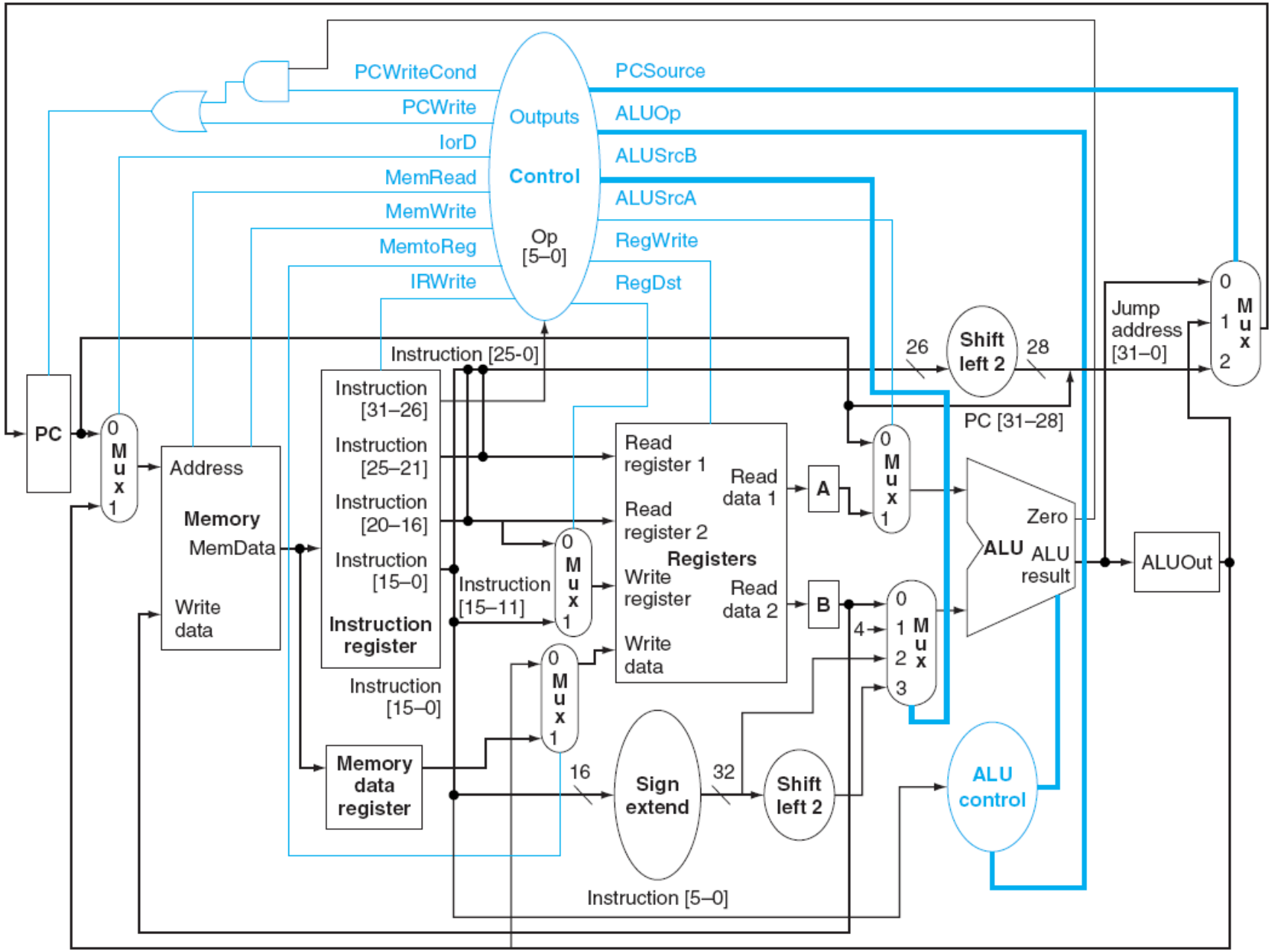


FIGURE 5.25 The high-level view of the multicycle datapath. This picture shows the key elements of the datapath: a shared memory unit, a single ALU shared among instructions, and the connections among these shared units. The use of shared functional units requires the addition or widening of multiplexors 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.

Fetch (I)


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
ALU
Zero
ALU
result
ALUOut
Memory
data
register
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 indirizzi 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: PCSource, 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: PCWrite, 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.

Segnali di controllo a 1 bit

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.

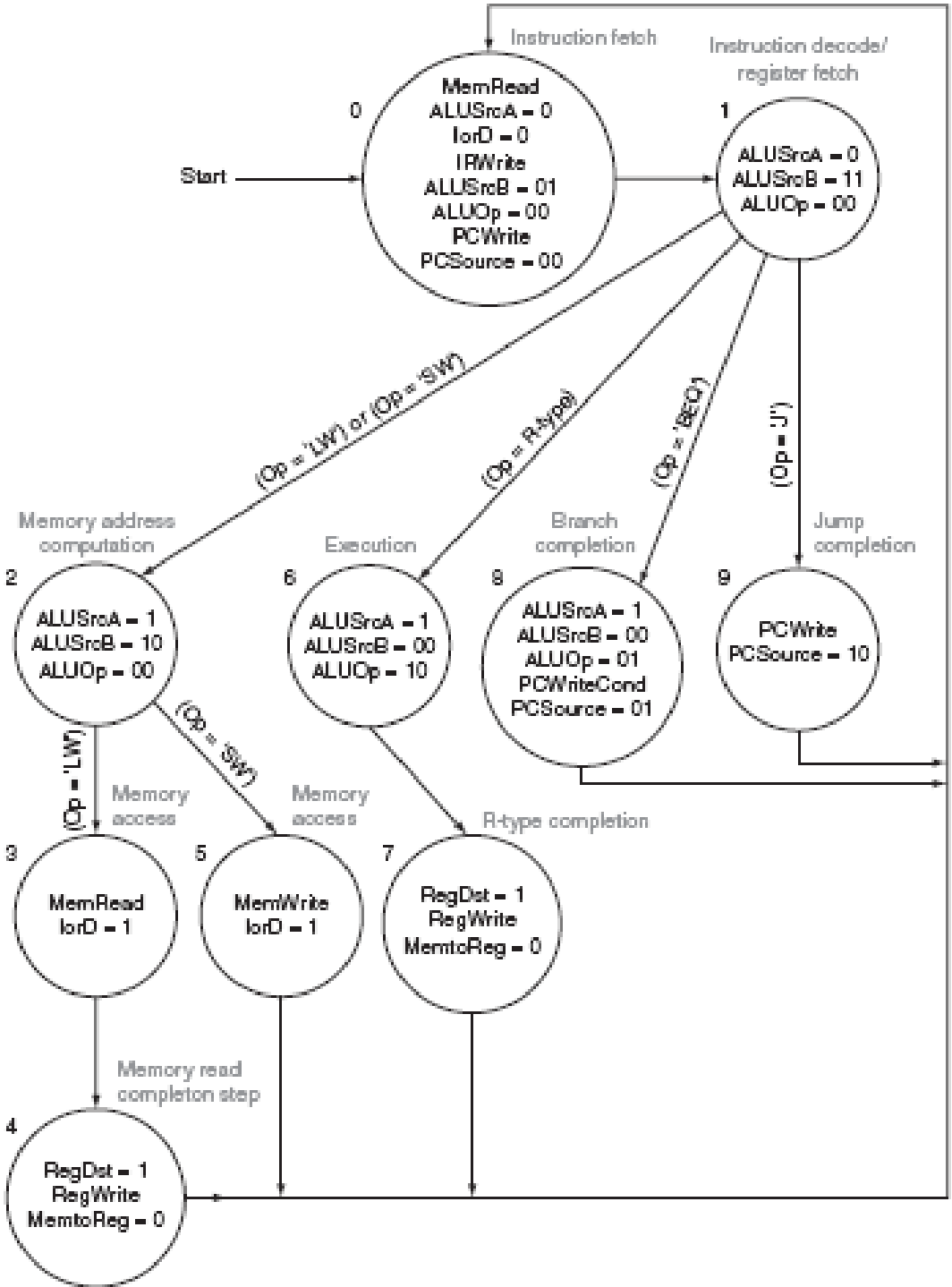
Segnali di controllo a 2 bit

Actions of the 2-bit control signals

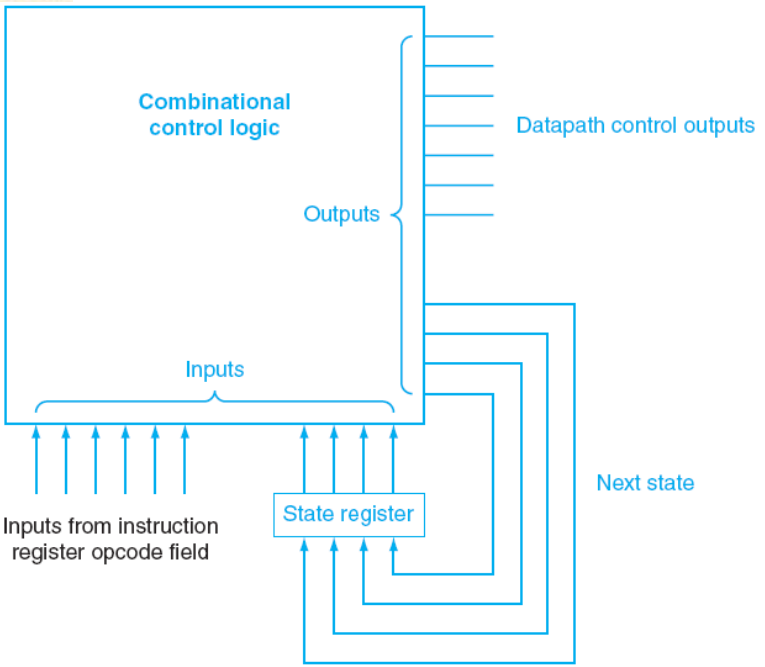
Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 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.

FIGURE 5.29 The action caused by the setting of each control signal in Figure 5.28 on page 323. The top table describes the 1-bit control signals, while the bottom table describes the 2-bit signals. Only those control lines that affect multiplexors have an action when they are deasserted. This information is similar to that in Figure 5.16 on page 306 for the single-cycle datapath, but adds several new control lines (IRWrite, PCWrite, PCWriteCond, ALUSrcB, and PCSource) and removes control lines that are no longer used or have been replaced (PCSrc, Branch, and Jump).

FSM


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(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

Circuito sequenziale per il controllo



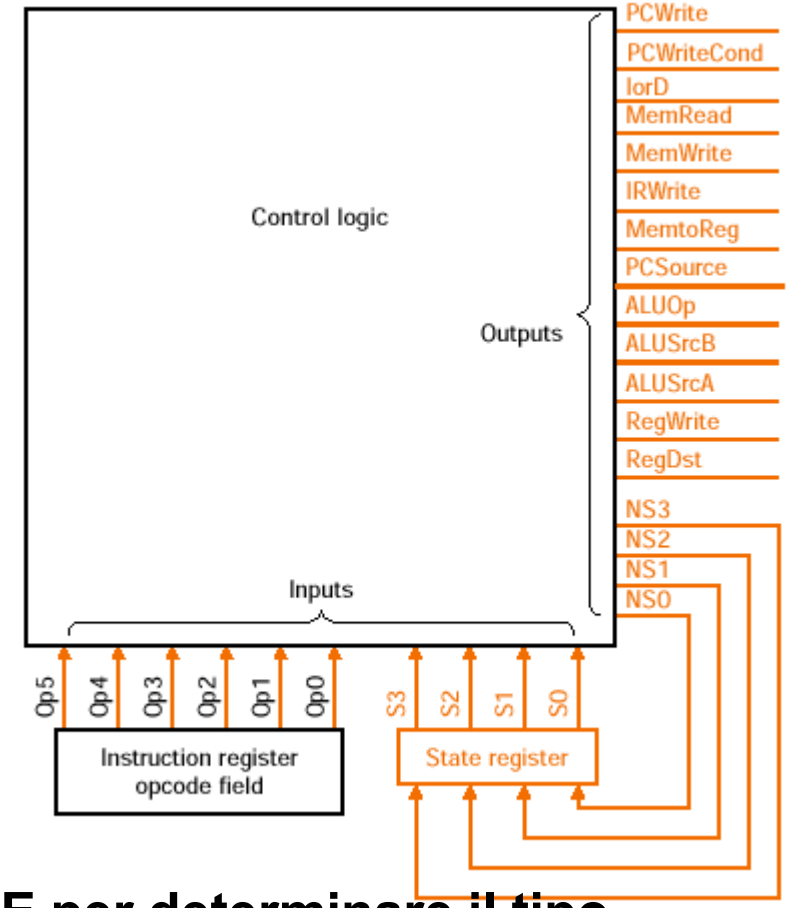


FIGURE 5.37 Finite state machine controllers are typically implemented using a block of combinational logic and a register to hold the current state. The outputs of the combinational logic are the next-state number and the control signals to be asserted for the current state. The inputs to the combinational logic are the current state and any inputs used to determine the next state. In this case, the inputs are the instruction register opcode bits. Notice that in the finite state machine used in this chapter, the outputs depend only on the current state, not on the inputs. The Elaboration above explains this in more detail.

- **Controllo a due livelli: si usa OPCODE per determinare il tipo dell'istruzione e per R_type si usa anche il FUNC CODE**
- **State register memorizza lo stato corrente**
- **Blocco combinatorio (PLA) per il calcolo di NEXT_STATE e OUTPUT (memorizzate in ROM)**

Da leggere

- **Chapter 5: The processor: Datapath and Control – disponibile sul sito elearning del corso**