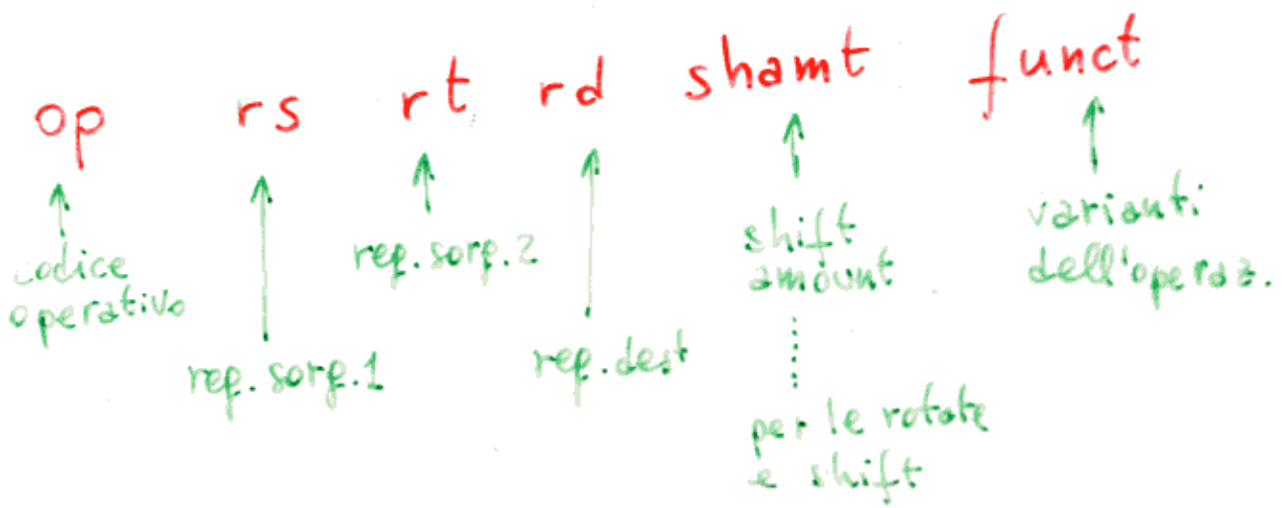
ALCUNE ISTRUZIONI DEL MIPS (COMPUTER COMPANY)

hp: - 32 registri visibili
- $0 sempre uguale a zero

istruzioni LOGICHE/ARITMETICHE (classe R)

	sintassi	semantica
somma	add $1, $2, $3	$1 = $2 + $3
sottrazione	sub $1, $2, $3	$1 = $2 - $3
prod. logico	and $1, $2, $3	$1 = $2 and $3
somma log.	or $1, $2, $3	$1 = $2 or $3
confronto di minoranza	slt $1, $2, $3 (Set less than)	se ($2 < $3) allora $1 = 1, altr. $1 = 0

FORMATO ISTRUZIONE CLASSE R



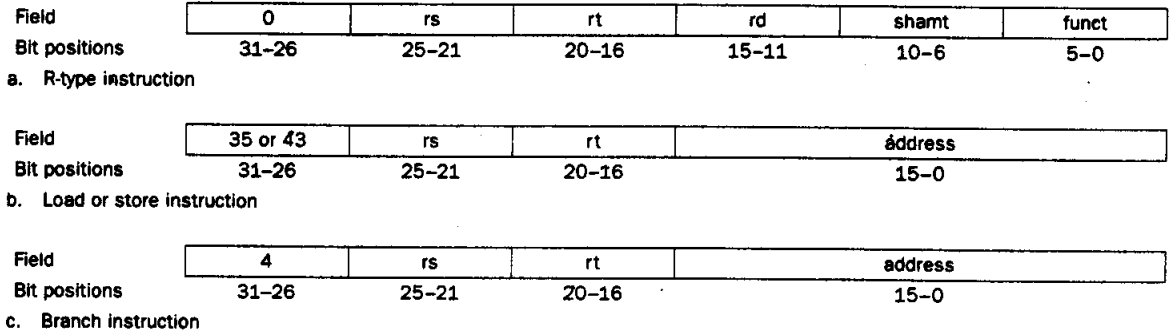


FIGURE 5.16 The three instruction classes (R-type, load and store, and branch) use two different instruction formats. The jump instructions use another format, which we will discuss shortly. (a) Instruction format for R-format instructions, which all have an opcode of 0. These instructions have three register operands: rs, rt, and rd. Fields rs and rt are sources, and rd is the destination. The ALU function is in the funct field and is decoded by the ALU control design in the previous section. The R-type instructions that we implement are `add`, `sub`, `and`, `or`, and `slt`. The shamt field is used only for shifts; we will ignore it in this chapter. (b) Instruction format for load (opcode = 35_{ten}) and store (opcode = 43_{ten}) instructions. The register rs is the base register that is added to the 16-bit address field to form the memory address. For loads, rt is the destination register for the loaded value. For stores, rt is the source register whose value should be stored into memory. (c) Instruction format for branch equal (opcode = 4). The registers rs and rt are the source registers that are compared for equality. The 16-bit address field is sign-extended, shifted, and added to the PC to compute the branch target address.

p. esempio

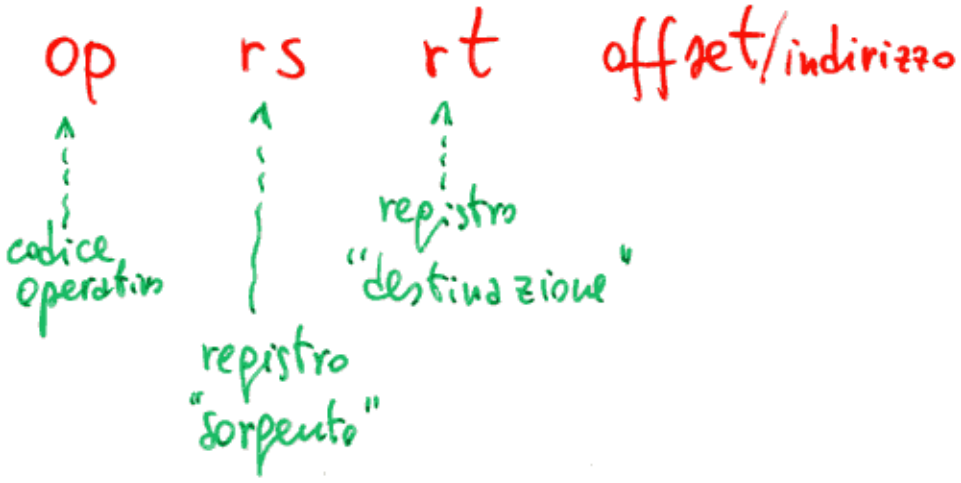
add $1, $2, $3 ----- $1 = $2 + $3

istruzioni CARICAMENTO - MEMORIZZAZIONE
SALTO CONDIZIONATO (classe I)

	sintassi	semantica
caricamento parola	lw $1, 100($2)	---- $1 = MEMORIA ($2+100)
memorizzazione parola	sw $1, 100($2)	---- MEMORIA ($2+100) = $1
salta se uguale	beq $1, $2, L	----- se ($1 == $2) vai a L

PC ← PC+4 + L

FORMATO ISTRUZIONE CLASSE I



lw

sw

beq

p. esempio

lw $1, 100($2) ---· $1 = MEMORIA ($2+100)

FASI ESECUZIONE ISTRUZIONE

add, sub, and, or

fetch → lettura registri → oper. ALU → scritt. reg.

slt

fetch → lettura registri → oper. ALU → scritt. reg. con 0 o con 1

lw

fetch → lettura registri → operaz. ALU → accesso memoria lettura → scritt. reg.

sw

fetch → lettura registri → oper. ALU → accesso memoria scrittura

beq

fetch → lettura registri → oper. ALU → eventuale salto a PC+4+ offset

Processore uniciclo

PROGETTO

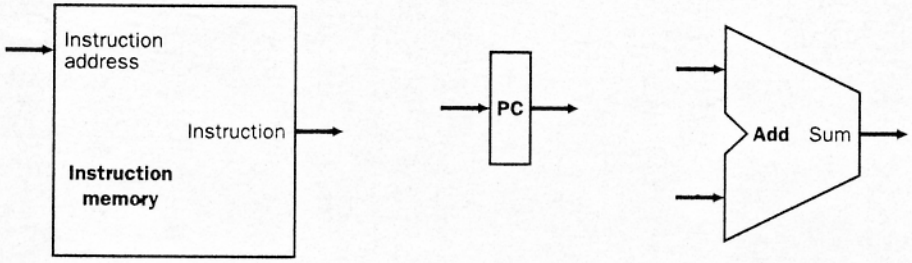
SCO-SCA

1 ciclo istruzione

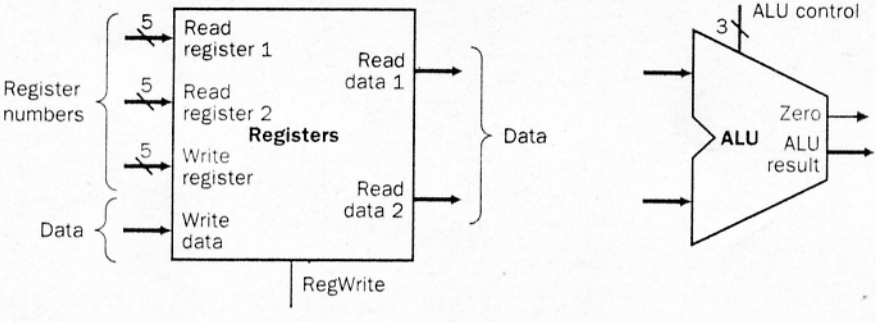
in

1 ciclo di CK

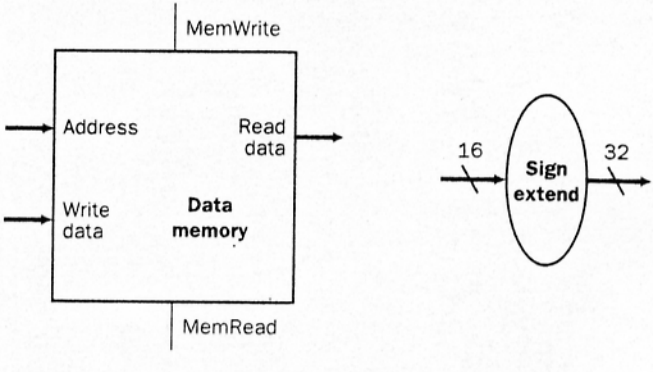
- PROGETTO SCA
- PROGETTO SCO → DATA-PATH



a. Instruction memory b. Program counter c. Adder

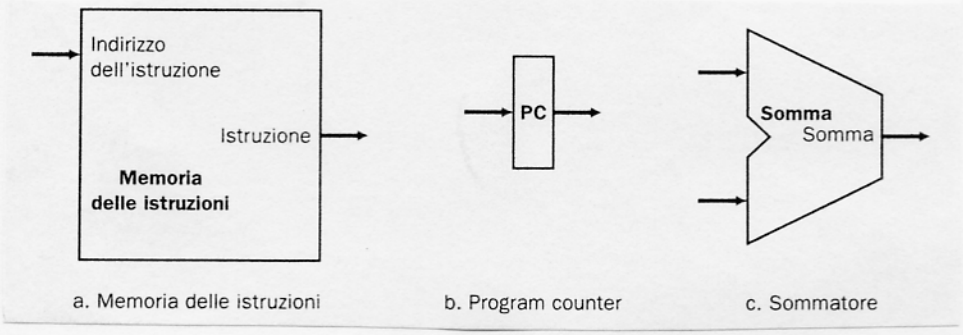


a. Registers b. ALU

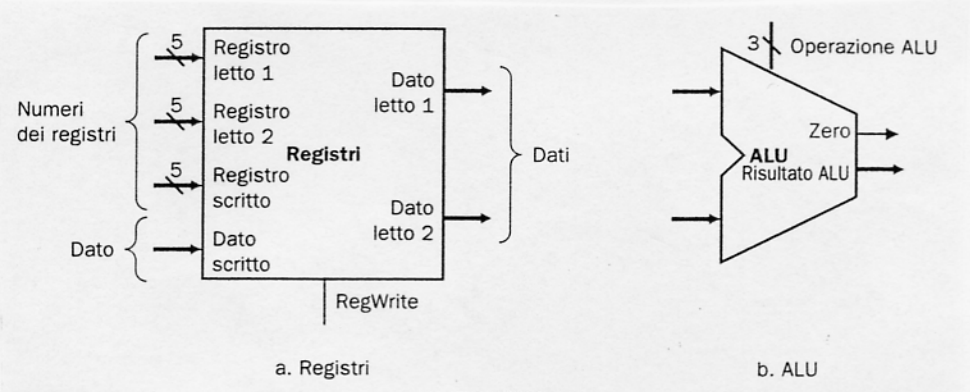


a. Data memory unit b. Sign-extension unit

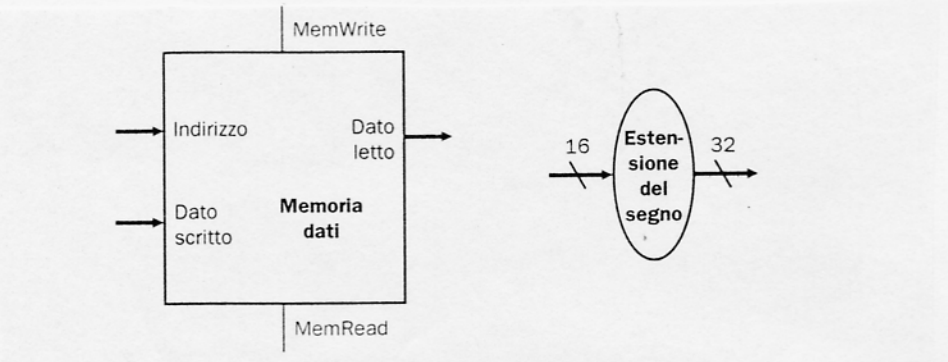
COMPONENTI BASE



a. Memoria delle istruzioni b. Program counter c. Sommatore



a. Registri b. ALU



a. Unità di memoria dei dati b. Unità di estensione del segno

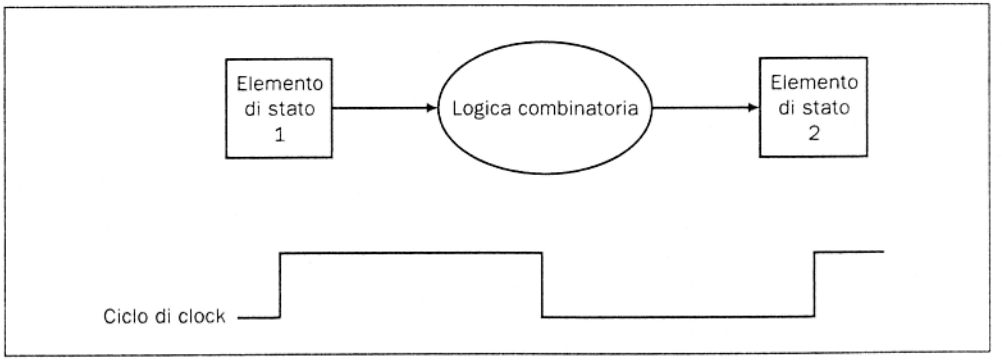


Figura 5.2 Logica combinatoria, elementi di stato e clock sono interdipendenti. In un sistema digitale sincrono il clock determina quando gli elementi di stato debbono scrivere i valori di ingresso nella loro memoria interna. Gli ingressi di un elemento di stato devono raggiungere uno stato stabile (ossia, devono aver raggiunto un valore destinato a non cambiare almeno fino al prossimo fronte del clock) prima che il fronte attivo del clock causi un aggiornamento dello stato; si assumerà che tutti gli elementi di stato, incluse le memorie, siano sensibili ai fronti.

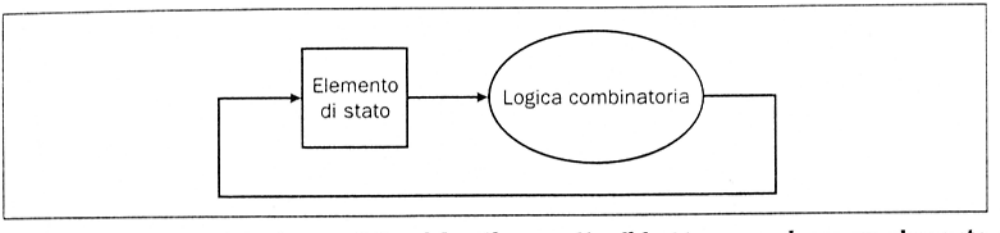
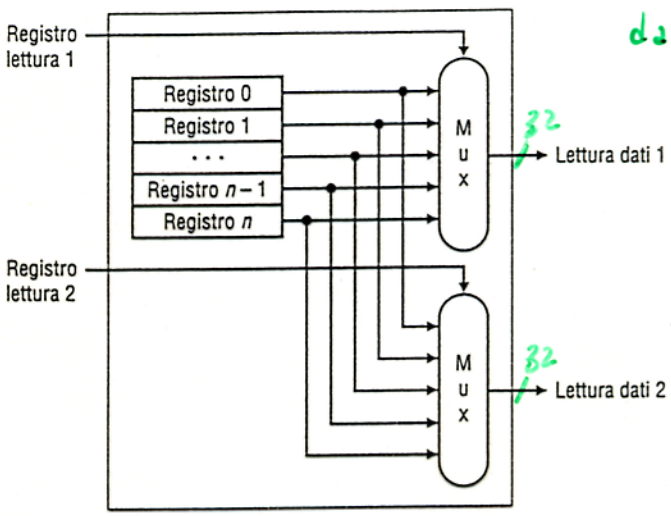


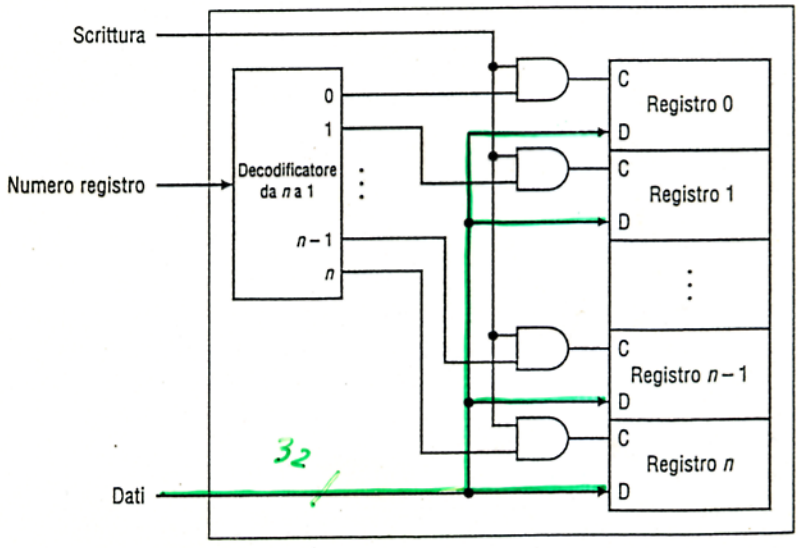
Figura 5.3 La metodologia sensibile ai fronti permette di leggere e scrivere un elemento di stato nello stesso ciclo di clock senza creare una condizione critica che potrebbe causare indeterminatezza nei valori dei dati. Il ciclo di clock deve avere una durata sufficiente a garantire che gli ingressi siano stabili al verificarsi del fronte attivo del clock. Non può verificarsi alcuna reazione ciclica del segnale in un solo ciclo di clock grazie all'aggiornamento sensibile ai fronti dell'elemento di stato: se fosse possibile un tale ritorno di segnale questo circuito non funzionerebbe correttamente. I progetti presentati in questo capitolo e nel prossimo si basano sulla metodologia di temporizzazione sensibile ai fronti e su strutture simili a quella indicata in figura.

ARCHITETTURA BANCO DEI REGISTRI



dati di 32 bit

FIGURA B.19 Sintesi di due porte di lettura, per un banco di *n* registri; occorrono due multiplatori, ciascuno in grado di selezionare tra *n* bus a 32 bit. L'etichetta del registro sottoposto a operazione funziona come segnale di selezione per il multiplatore. La sintesi della porta di scrittura è invece mostrata nella figura B.20.

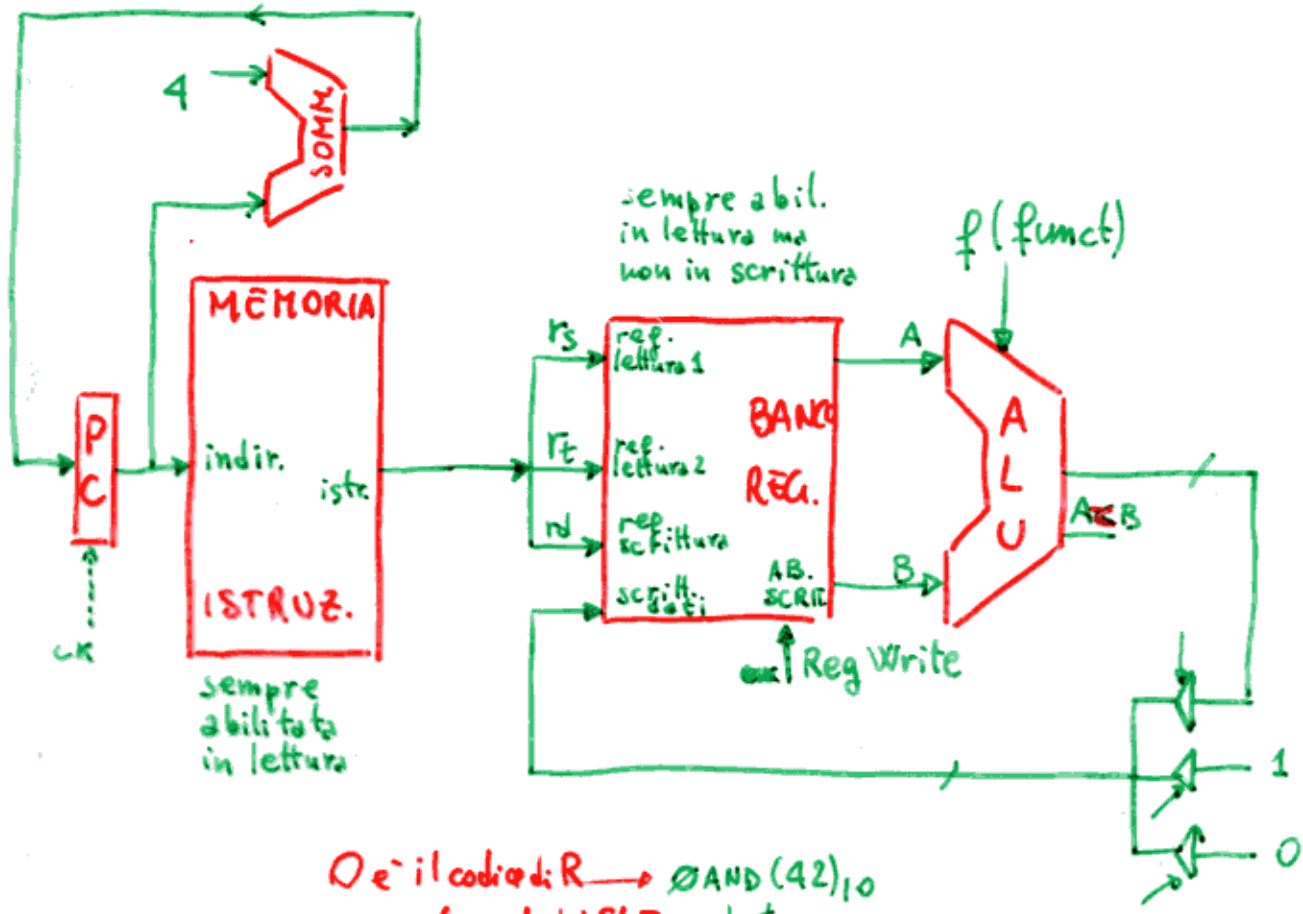


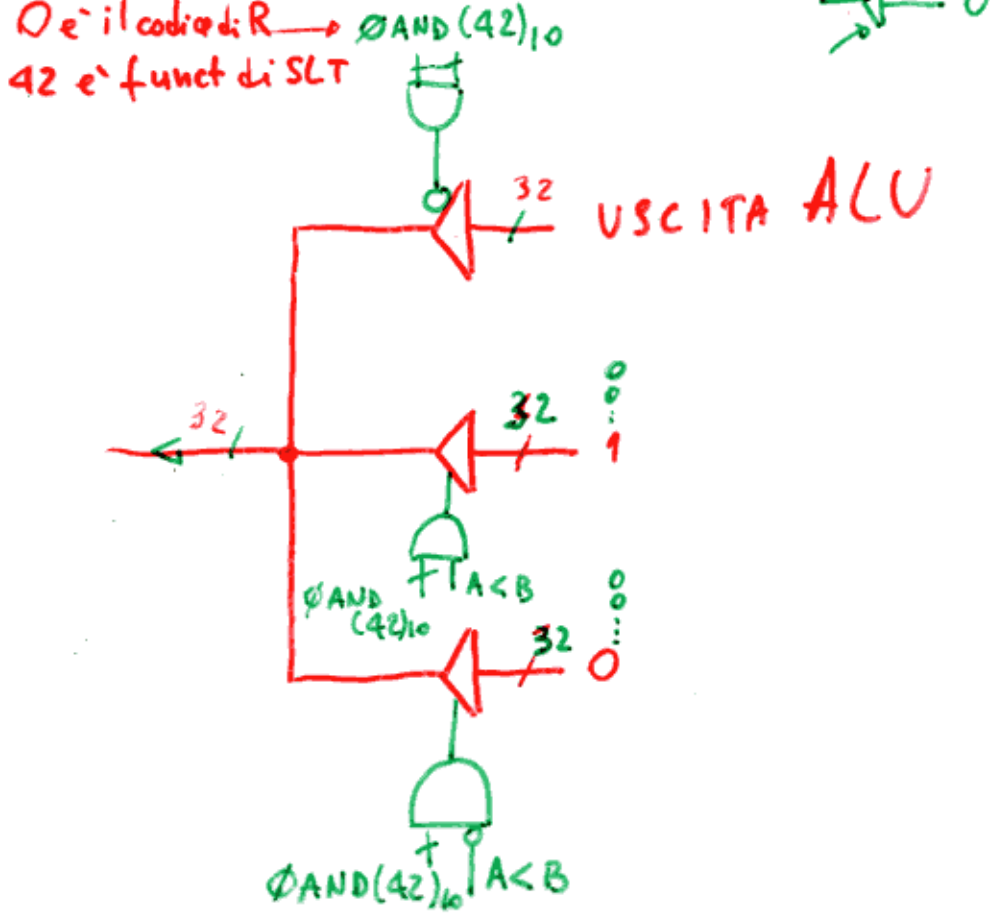
in ogni registro ci sono 32 F/F di tipo D

FIGURA B.20 Sintesi di una porta di scrittura per un banco dei registri; occorre un decodificatore, usato insieme al segnale di scrittura, per generare il segnale di clock da inviare ai bistabili componenti i registri. Tutti i tre gruppi di ingressi (l'etichetta del registro, il dato e il segnale di scrittura) sono soggetti a vincoli temporali, derivanti dai tempi di preparazione e di mantenimento, essenziali per garantire che il dato venga memorizzato nel banco in modo corretto.

ISTRUZIONI CLASSE R

FASI: ① FETCH ② LETTURA REGISTRI ③ OPERAZIONE ALU ④ SCRITTURA REGISTRO





ISTRUZIONI lw , sw

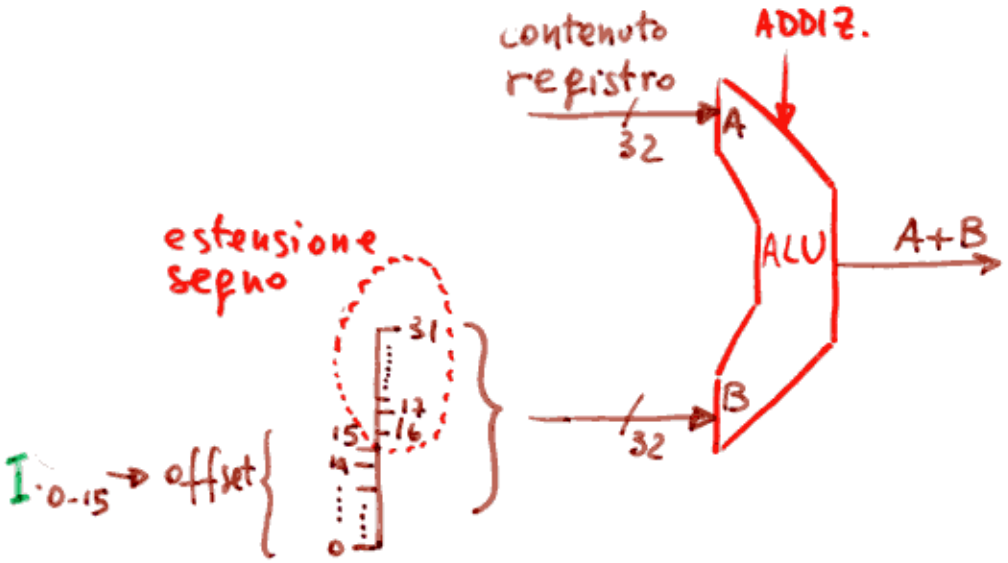
lw $1 , offset ($2) ---- $1 = memoria ($2+offset)

sw $1, offset ($2) ---- memoria ($2+offset) = $1

considerazioni

① ALU e registri di 32 bit

② offset è di 16 bit



se bit 15° = 1 ⟶ offset negativo

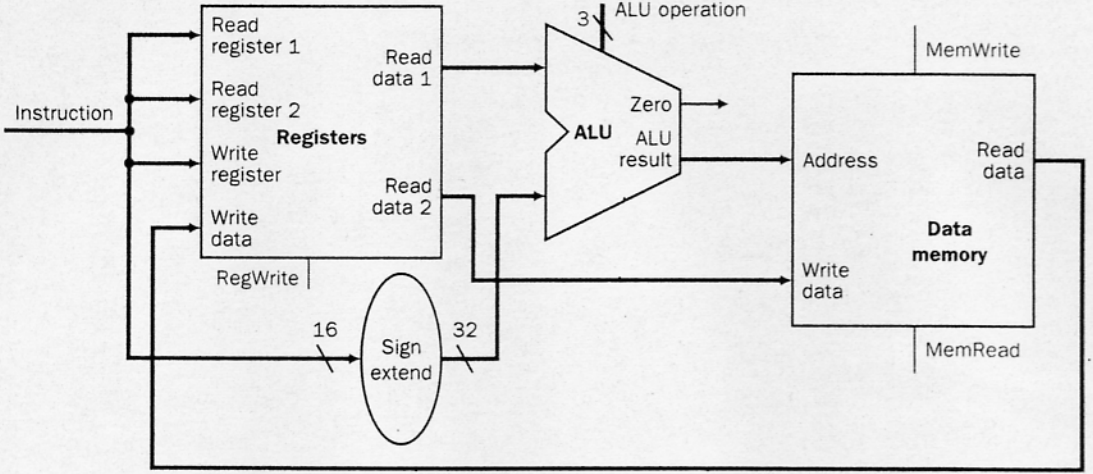


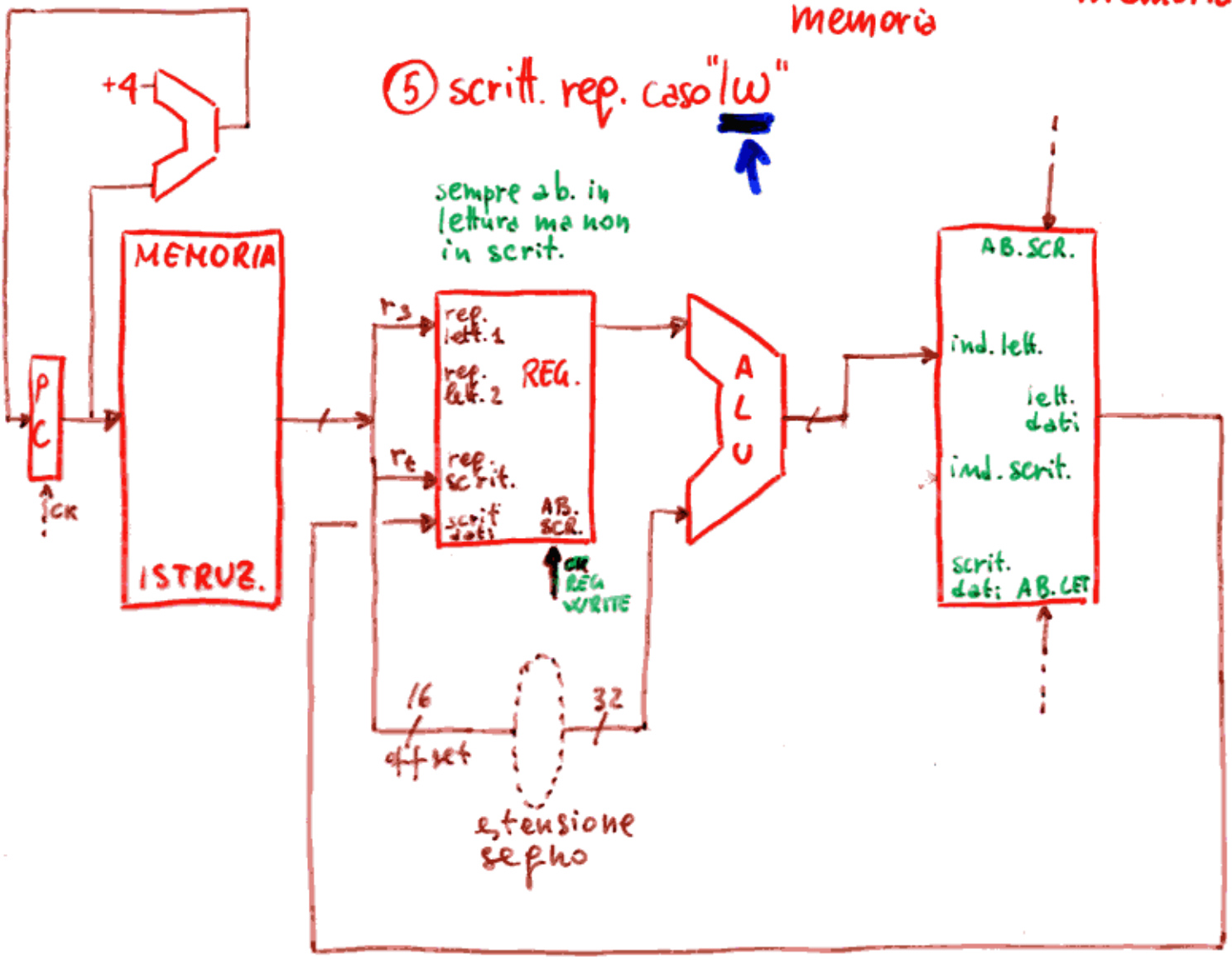
FIGURE 5.9 **The datapath for a load or store does a register access, followed by a memory address calculation, then a read or write from memory, and a write into the register file if the instruction is a load.**

---- continua

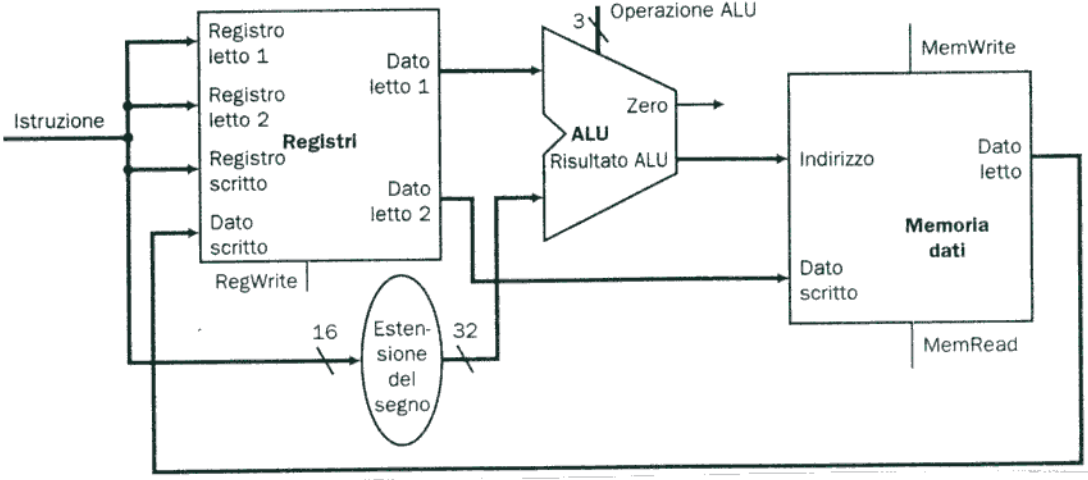
parte per "lw"

① fetch ② lettura registro ③ calcolo indirizzo memoria ④ accesso memoria

⑤ scritt. rep. caso "lw"



parte per "sw"


Istruzione
Registro letto 1
Registro letto 2
Registro scritto
Dato scritto
Registri
Dato letto 1
Dato letto 2
RegWrite
3
Operazione ALU
ALU
Risultato ALU
Zero
16
Estensione del segno
32
MemWrite
Indirizzo
Dato scritto
Memoria dati
Dato letto
MemRead

istruz. I - beq

beq $1, $2, offset_value ---- se ($1 == $2) allora

PC = PC + 4 + offset

considerazioni

① offset_value è di 16 bit

② ALU e PC sono di 32 bit

③ indirizzi da cui riprendere solo mult. pl di 4
"allineamento della word sulle stesse "righe" "
detto anche SPIAZZAMENTO DI PAROLA

trucco prendo offset "moltiplicato per quattro"
scorrimento a sinistra di 2 bit

④ PC lo trovo già incrementato di 4

Conviene quindi

PC = PC+4 + (offset_value)*4

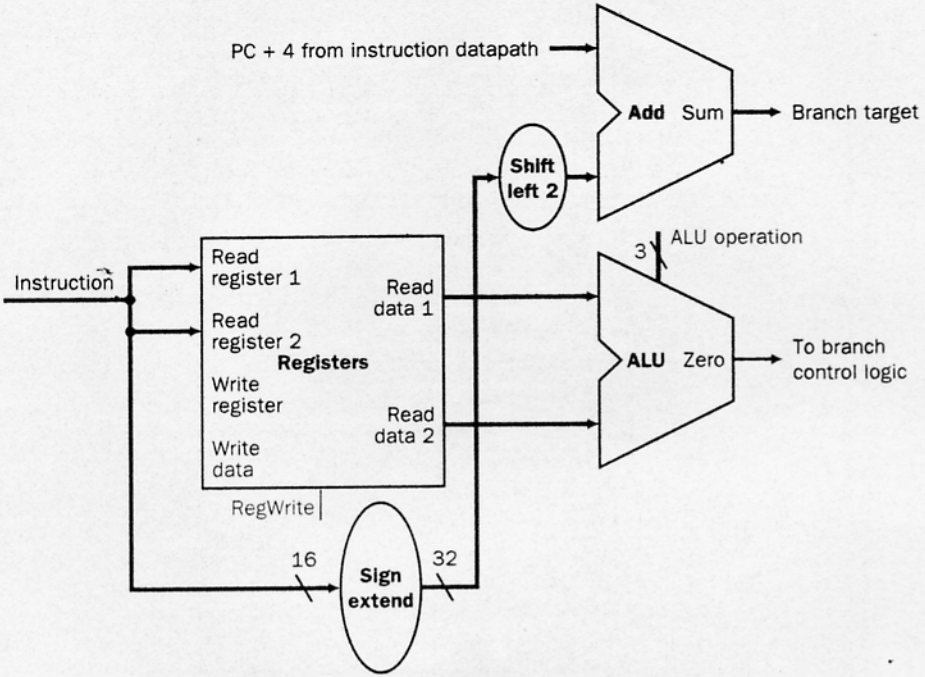
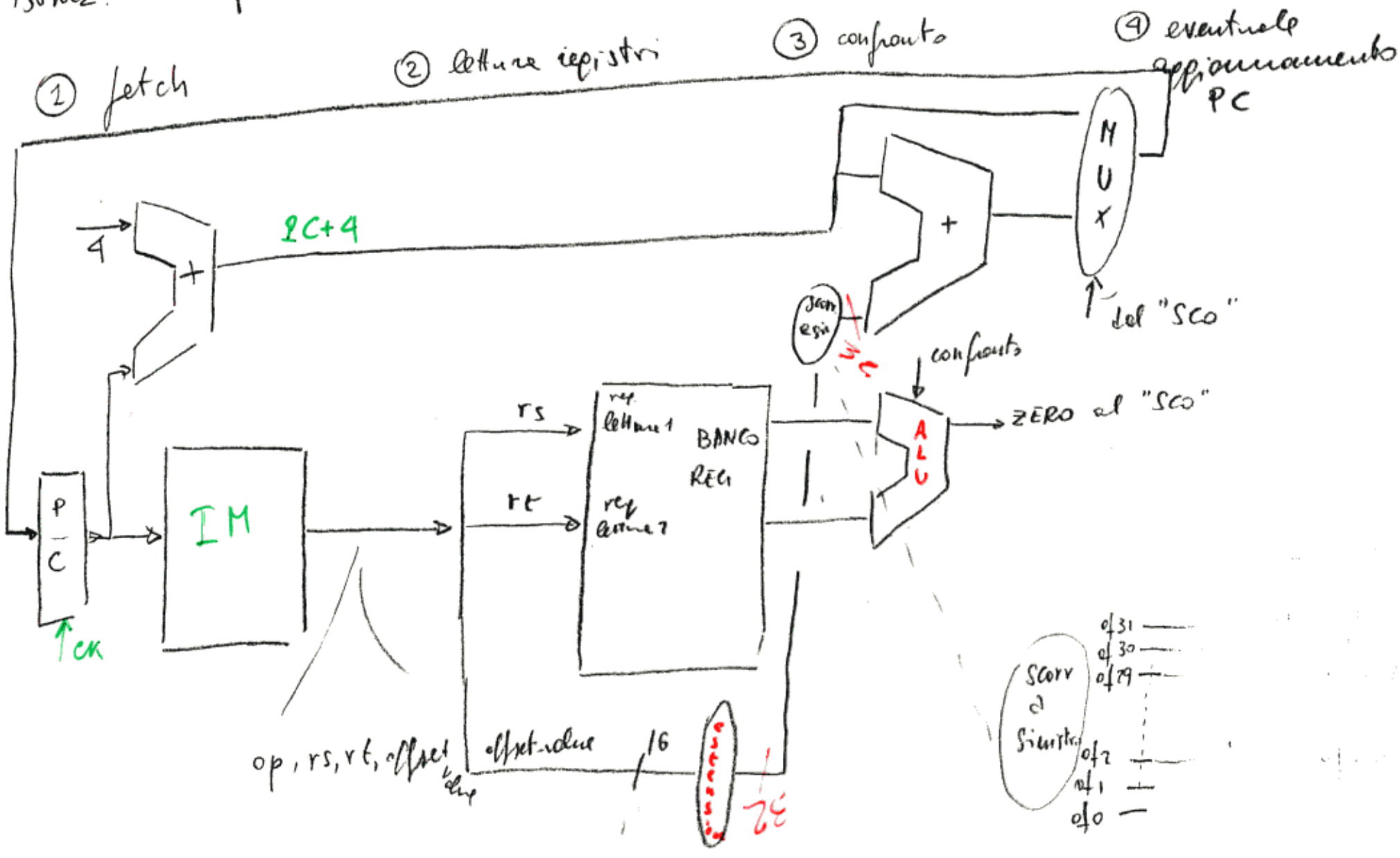


FIGURE 5.10 The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits. The unit labeled *Shift left 2* is simply a routing of the signals between input and output that adds 00_{two} to the low-order end of the sign-extended offset field; no actual shift hardware is needed, since the amount of the "shift" is constant. Since we know that the offset was sign-extended from 16 bits, the shift will throw away only "sign bits." Control logic is used to decide whether the incremented PC or branch target should replace the PC, based on the Zero output of the ALU.

istruz. I-beq


① fetch
② lettura registri
③ confronto
④ eventuale aggiornamento PC
4
+
PC+4
+
MUX
del "SCO"
jump reg
confronto
ALU
ZERO al "SCO"
rs
rt
reg address
BANCO REGISTRI
reg address
P C
IM
CK
op, rs, rt, offset
offset_value
16
32
scorr a sinistra
of31
of30
of29
of2
of1
of0

scorrimento a sinistra di 2 posizioni

17 bit 15 bit

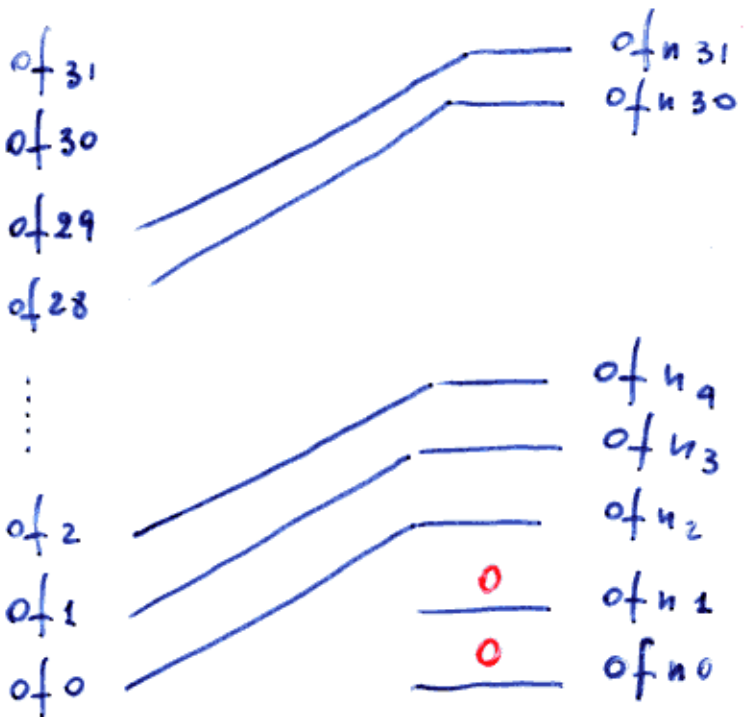
offset già esteso 00...0 1 1 0 ... 0 1

offset già esteso × 4 (100)

00...0 1 1 0 ... 0 1 ×
1 0 0 =

00...0 1 1 0 ... 0 1 0 0

15 bit 17 bit



ASSEMBLAGGIO SCA

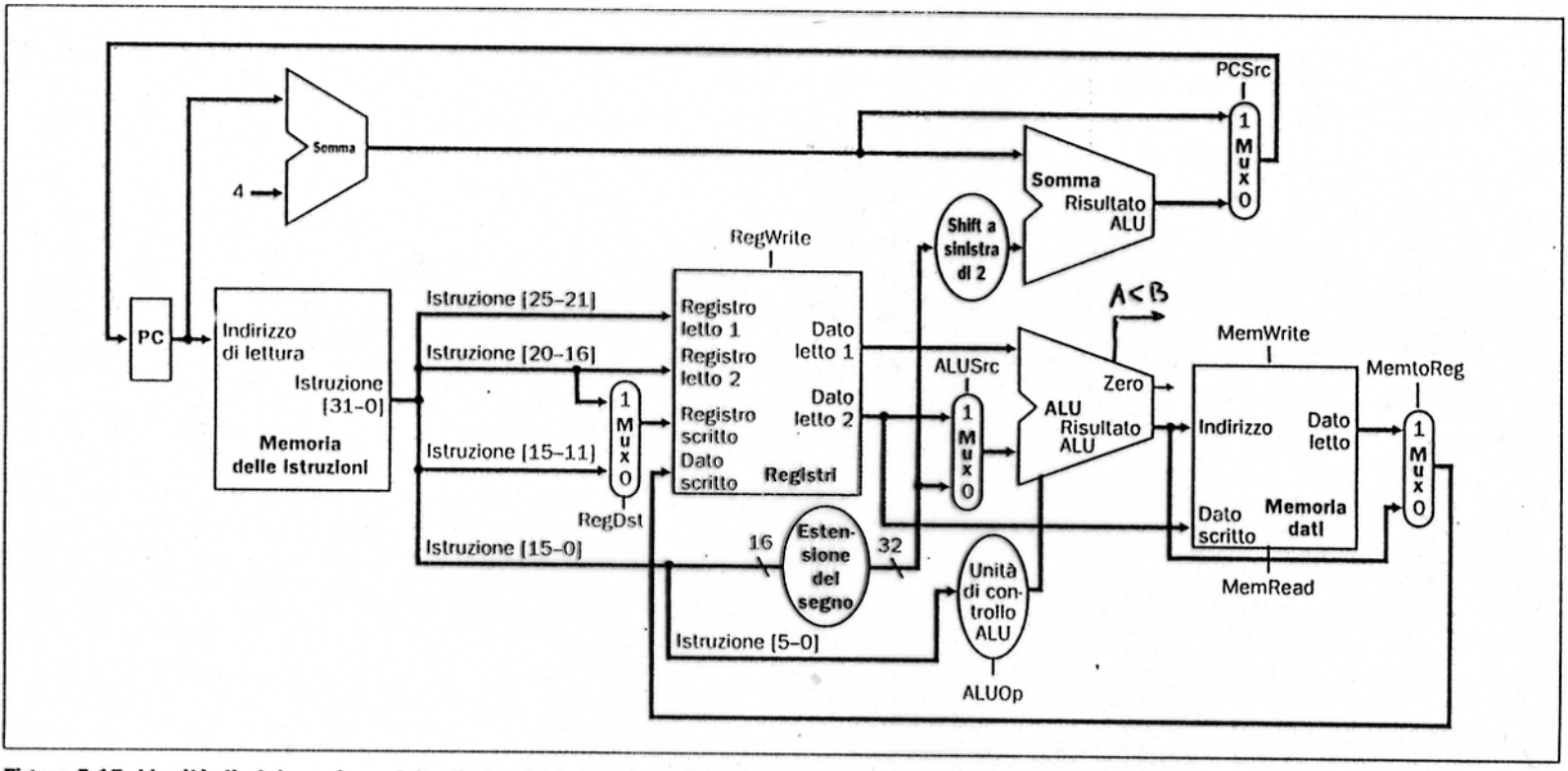
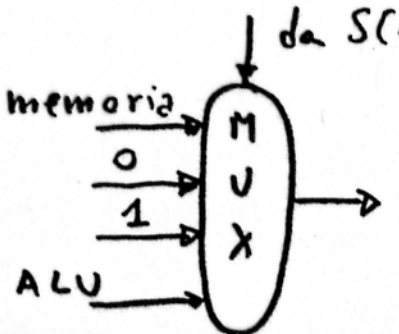


Figura 5.17 L'unità di elaborazione della figura 5.13 completa di tutti i multiplexer necessari e con l'identificazione di tutti i segnali di controllo. I segnali di controllo sono indicati con il colore ed è stata aggiunta anche l'unità di controllo della ALU. PC non necessita di segnali di controllo in quanto viene scritto al termine di ogni ciclo di clock; la logica di controllo dei salti determina se viene scritto PC incrementato o l'indirizzo di destinazione del salto.

ingressi
0 e 1 per
SLT



CONTROLLO ALU

Ingresso di controllo della ALU	Funzione
000	AND
001	OR
010	somma
110	sottrazione
111	set on less than

Istruzioni	Codice operativo istruzione	Codice della funzione	Segnali controllo ALU
	$O_5 O_4 O_3 O_2 O_1 O_0$	$F_5 F_4 F_3 F_2 F_1 F_0$	O_{P2} O_{P1} O_{P0}
somma	0 0 0 0 0 0	1 0 0 0 0 0	0 1 0
sottraz.	0 0 0 0 0 0	1 0 0 0 1 0	1 1 0
AND	0 0 0 0 0 0	1 0 0 1 0 0	0 0 0
OR	0 0 0 0 0 0	1 0 0 1 0 1	0 0 1
set on less than	0 0 0 0 0 0	1 0 1 0 1 0	1 1 1
lw	1 0 0 0 1 1	- - - - - -	0 1 0
sw	1 0 1 0 0 1	- - - - - -	0 1 0
branch equal	0 0 0 1 0 0	- - - - - -	1 1 0

p.e. $O_{P2} = \bar{O}_5 \bar{O}_4 \bar{O}_3 \bar{O}_2 \bar{O}_1 \bar{O}_0 F_5 \bar{F}_4 (\bar{F}_3 \bar{F}_2 F_1 \bar{F}_0 + F_3 \bar{F}_2 F_1 \bar{F}_0) +$

$+ \bar{O}_5 \bar{O}_4 \bar{O}_3 O_2 \bar{O}_1 \bar{O}_0$

SOLUZIONE AUTORI:

RICAVARE DUE SEGNALI DI CONTROLLO (ALUop)

DAL CODICE OPERATIVO

13b

Codice operativo istruzione	ALUOp	Operazione dell'istruzione	Codice della funzione	Azione dell'ALU desiderata	Ingresso di controllo dell'ALU
LW	00	Caricamento parola	XXXXXX	add	010
SW	00	Memorizzazione parola	XXXXXX	add	010
Branch equal	01	Salto condizionato	XXXXXX	subtract	110
Tipo R	10	Somma	100000	add	010
Tipo R	10	Sottrazione	100010	~~add~~ sub	110
Tipo R	10	AND	100100	and	000
Tipo R	10	OR	100101	or	001
Tipo R	10	Confronto di minoranza	101010	set-on-less-than	111

ALUOp		Codice della funzione						Operazione ALU
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

OP2

x 1 x x x x x x } x 1 xx x x x x

1 x x x 0 0 1 0 } 1 x xx x x 1 x

1 x x x 1 0 1 0 }

OP1

0 0 x x x x x x } 0x x x x x x x

x 1 x x x x x x

1 x x x 0 0 0 0 } 1x x x x 0 x x

1 x x x 0 0 1 0

1 x x x 1 0 1 0

OP0

1 x x x 0 1 0 1 } 1x x x x x x 1

1 x x x 1 0 1 0 } 1x x x 1 x x x

13d

ALUOp		Campi di codice della funzione					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
X	1	X	X	X	X	X	X
1	X	X	X	X	X	1	X

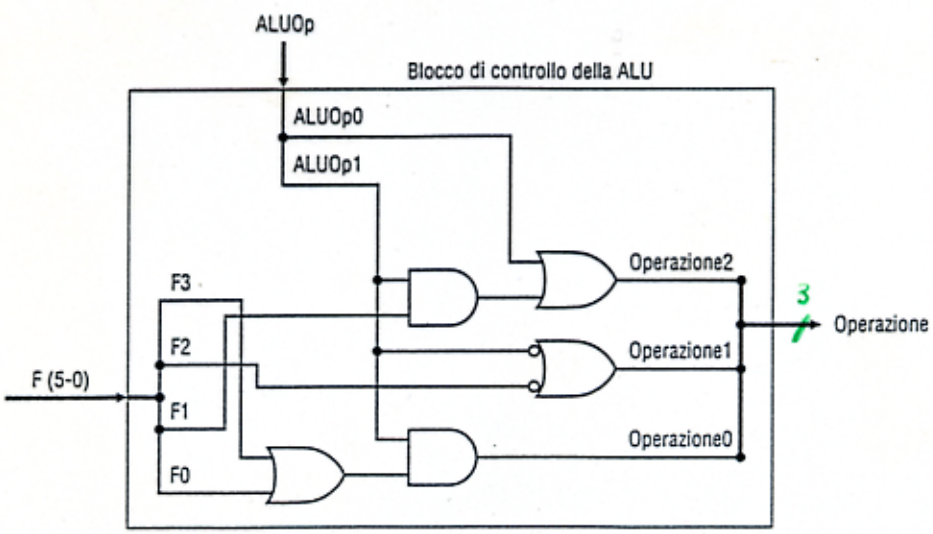
a. Tabella delle verità per Operazione2 = 1. (Questa tabella corrisponde al bit di sinistra del campo Operazione nella figura 5.16.)

ALUOp		Campi di codice della funzione					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	X	X	X	X	X	X	X
X	X	X	X	X	0	X	X

b. Tabella delle verità per Operazione1 = 1.

ALUOp		Campi di codice della funzione					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	X	X	X	X	X	X	1
1	X	X	X	1	X	X	X

c. Tabella delle verità per Operazione0 = 1.



CONTROLLO ALU