Architettura di un elaboratore

Ciclo di un’istruzione, controllo word

Sandro Savino (sandro.savino@dei.unipd.it)
Department of Information Engineering, University of Padova

Argomenti:

- Controllo e microoperazioni
- Instruction Set Architecture
- Computer a ciclo signolo

Materiale:

- Capitolo 8

DIPARTIMENTO
DI INGEGNERIA
DELL’INFORMAZIONE

Organizzazione e controllo del datapath

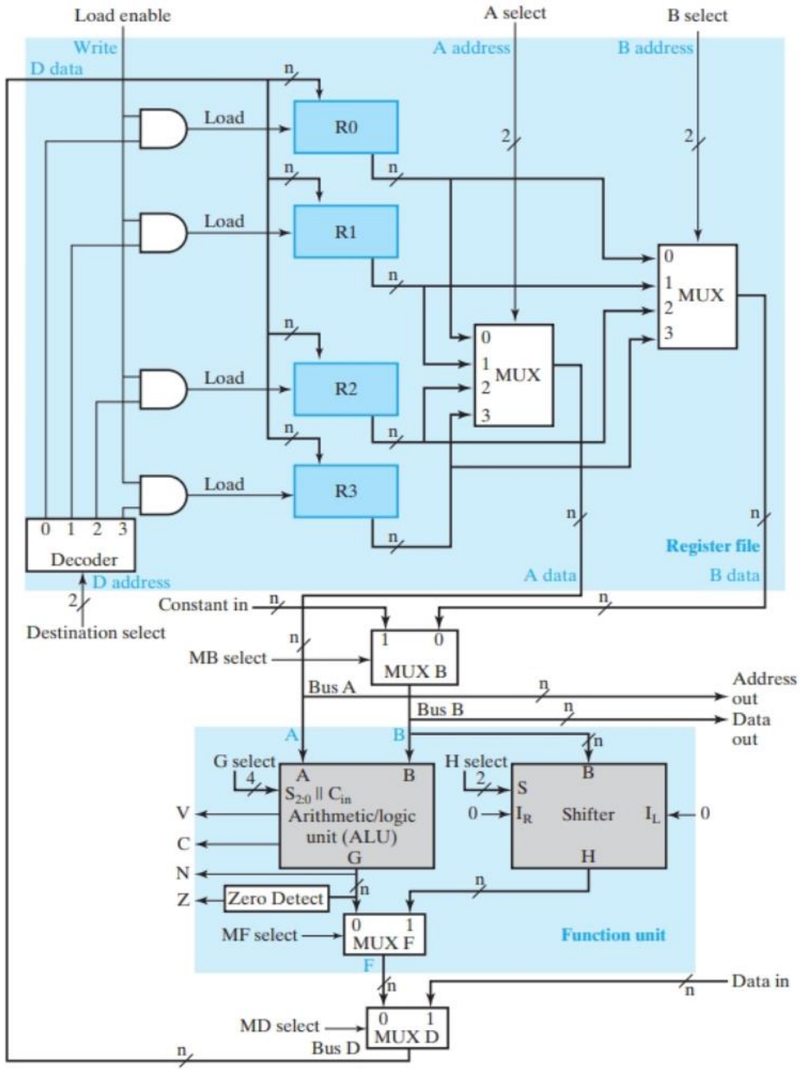


FIGURE 8-1
Block Diagram of a Generic Datapath

- Nel datapath, oltre alla function unit, abbiamo i registri
- i registri sono organizzati in un *register file*
- La dimensione del register file è $2^m \times n$, dove m è il numero di linee per i registri e n il numero di bits per registro
- La selezione di input, operazioni, output è eseguita tramite una *control word* che le identifica in maniera univoca
- Le operazioni vengono eseguite in un ciclo di clock

$2^2 = 4$ registri

Controllo del datapath

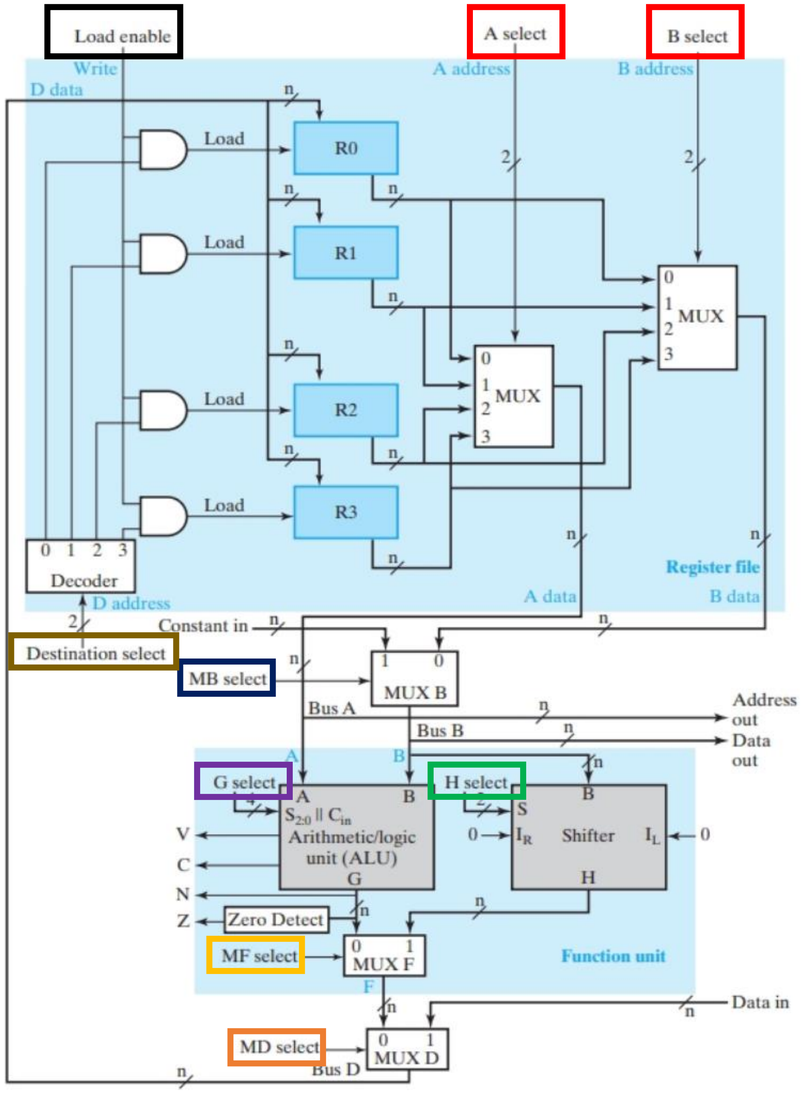


FIGURE 8-1
Block Diagram of a Generic Datapath

- A select, B select → selezionano quali registri utilizzare
- G select → seleziona l'operazione che l'ALU deve eseguire
- H select → seleziona se passare l'operatore B attraverso lo shifter o se eseguire un'operazione di shift
- MB select → seleziona se B proviene da una costante esterna
- MF select → seleziona se l'output proviene dall'ALU o dallo shifter
- MD select →selezione l'output della function unit oppure un dato esterno
- D select → seleziona il registro dove scrivere il risultato dell'operazione
- **Load enable** → attiva la scrittura nel register file

$2^2 = 4$ registri

Controllo del datapath

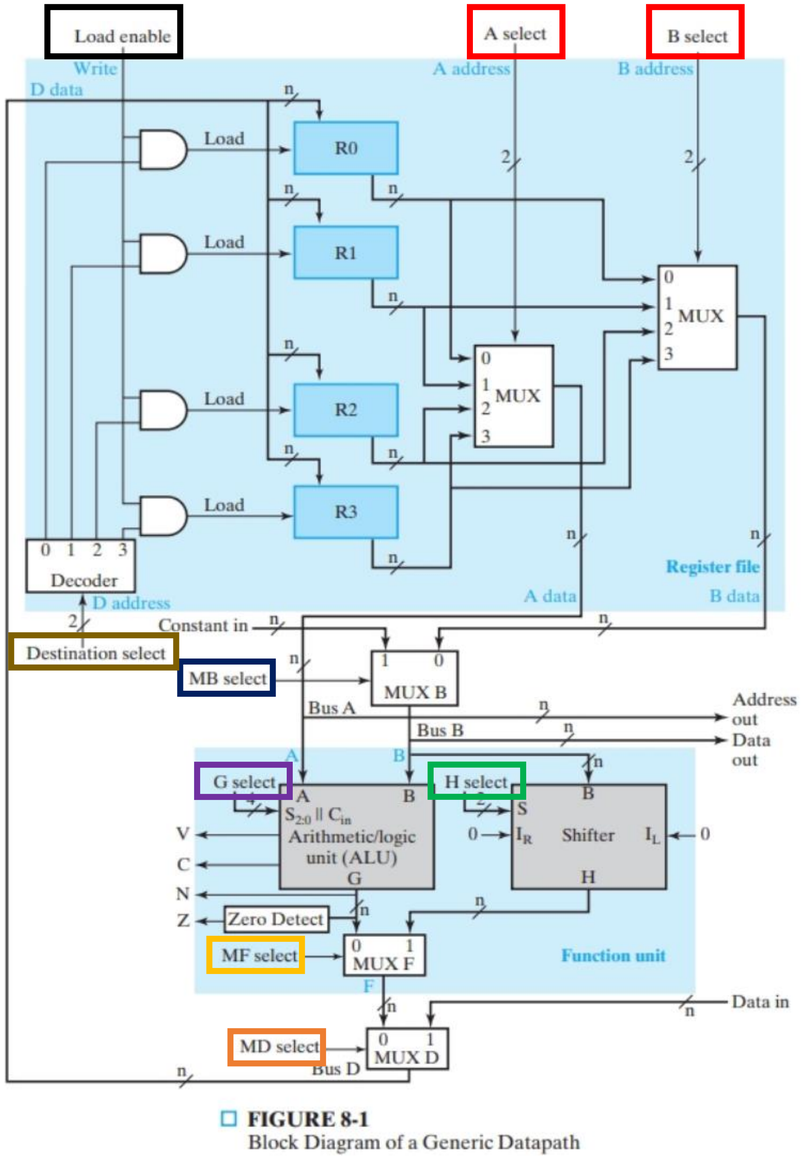
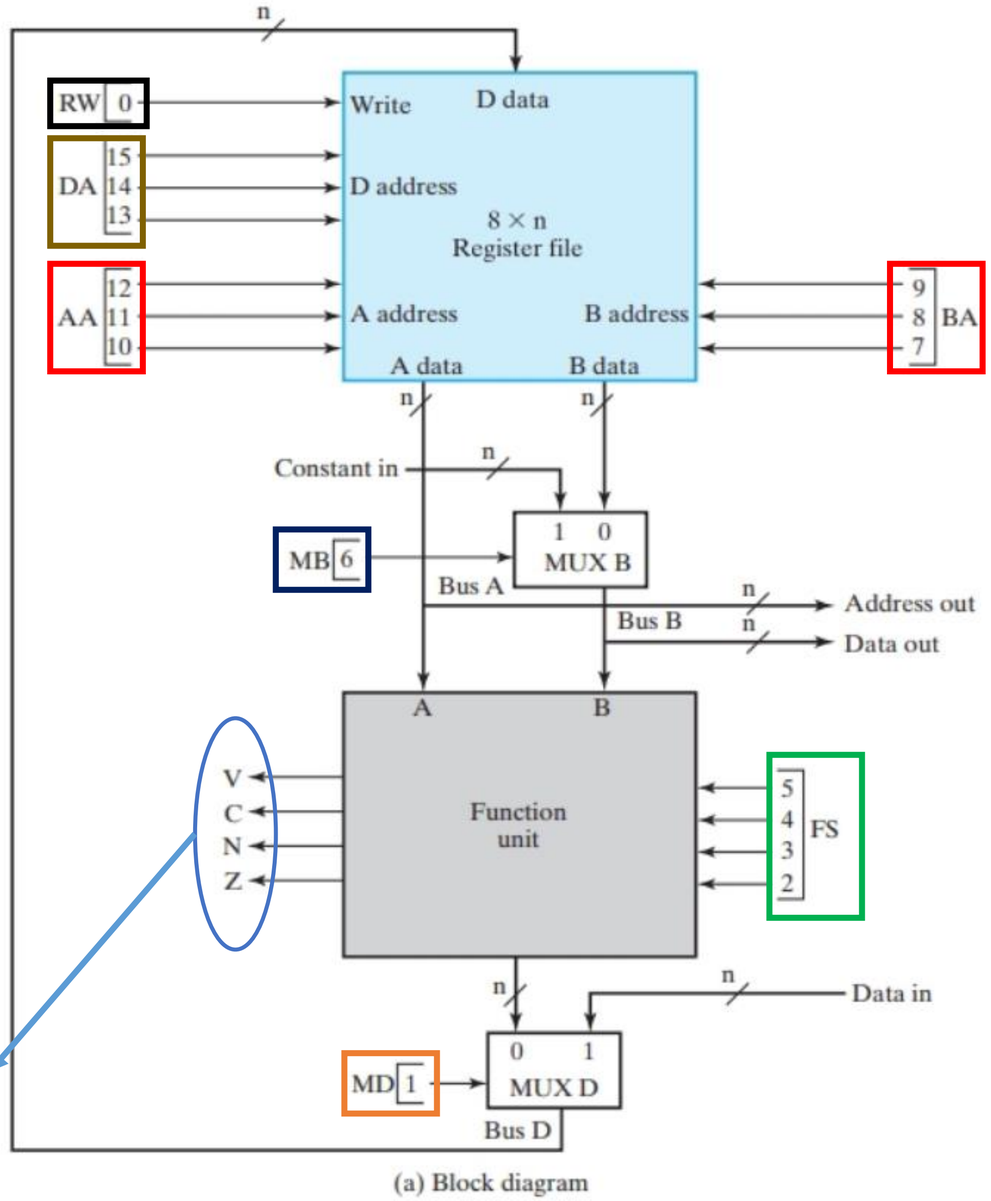


FIGURE 8-1
Block Diagram of a Generic Datapath

$2^2 = 4$ registri

L'operazione attuale ha causato:
V=1 overflow
C=1 carry
N=1 negativo
Z=1 zero

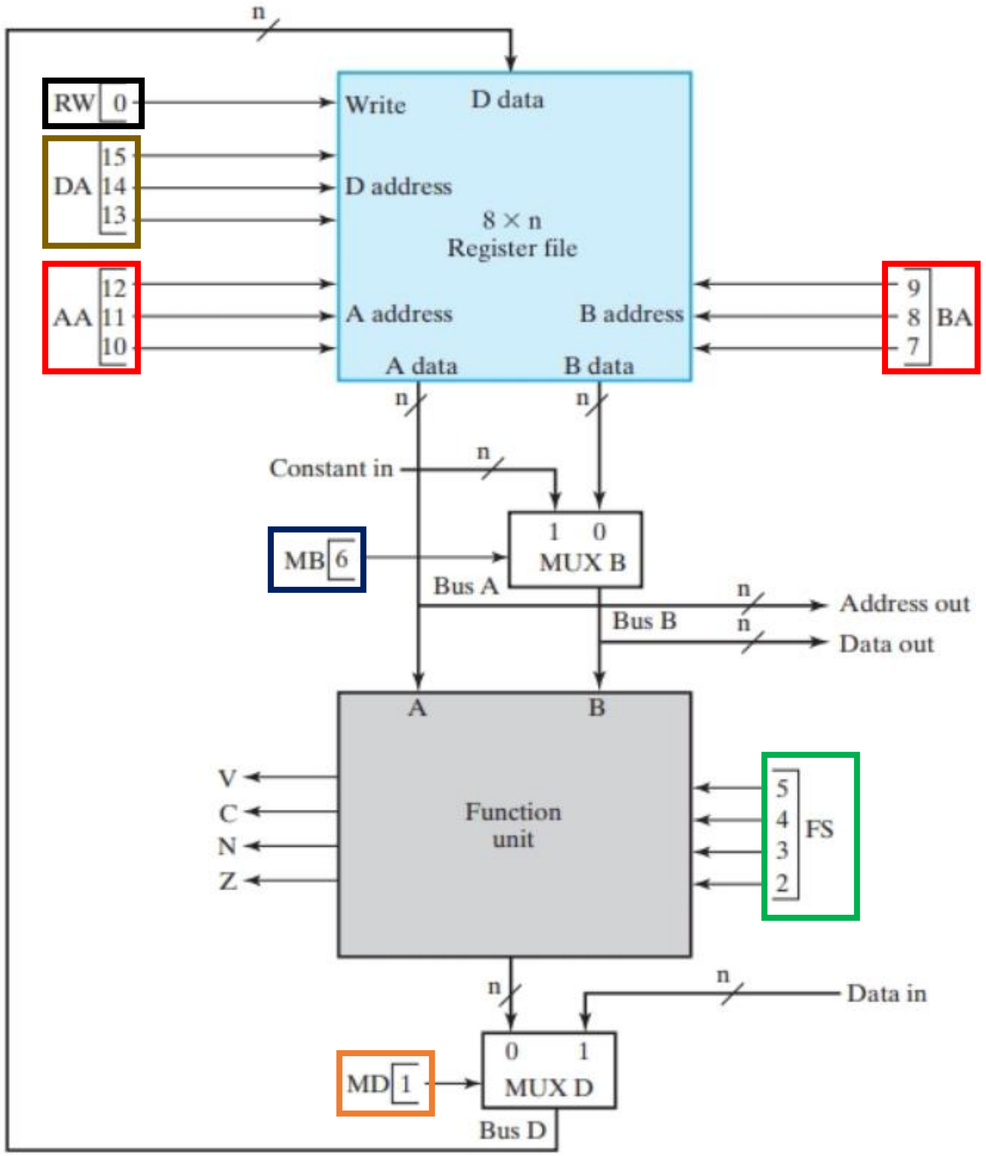


(a) Block diagram

$2^3 = 8$ registri

Controllo del datapath

- **DA** identifica l'indirizzo del registro di destinazione
- **AA** identifica l'indirizzo del registro per l'input A
- **BA** identifica l'indirizzo del registro per l'input B
- **MB** identifica se selezionare l'input B o una costante
- **FS** identifica l'operazione da eseguire (ALU o shifter)
- **MD** identifica l'output
- **RW** identifica se scrivere sul registro destinazione l'output

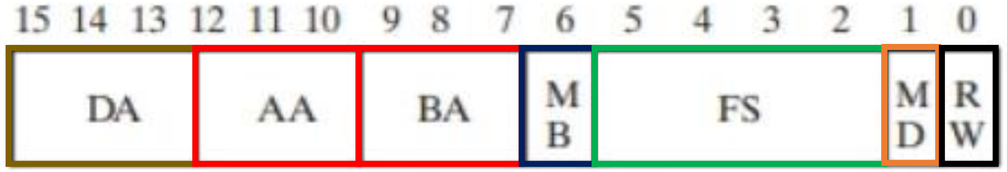


(a) Block diagram

$2^3 = 8$ registri

Control word

- **DA** identifica l'indirizzo del registro di destinazione
- **AA** identifica l'indirizzo del registro per l'input A
- **BA** identifica l'indirizzo del registro per l'input B
- **MB** identifica se selezionare l'input B o una costante
- **FS** identifica l'operazione da eseguire (ALU o shifter)
- **MD** identifica l'output
- **RW** identifica se scrivere sul registro destinazione l'output



(b) Control word

Control word:

- 16-bit binary control input
- 7 campi
- Identifica una microoperazione univoca

ALU: Arithmetic/Logic Unit

TABLE 8-2
Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

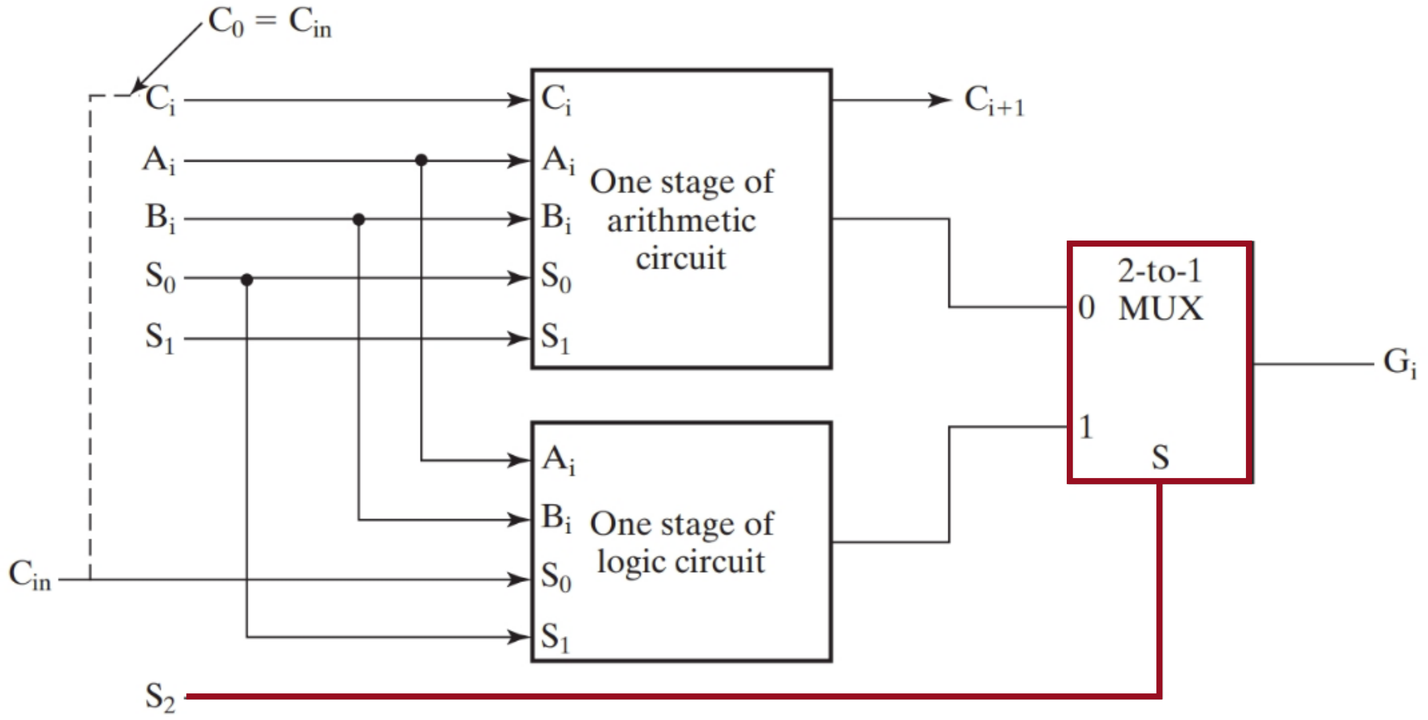


FIGURE 8-7
One Stage of ALU

ALU: Arithmetic/Logic Unit

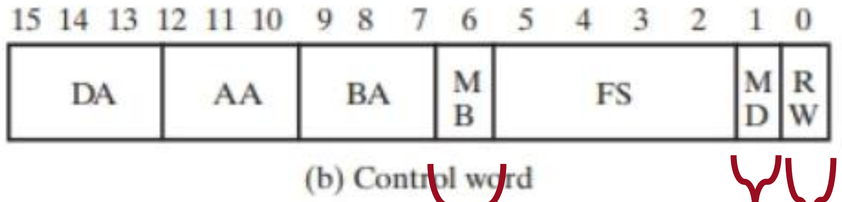
TABLE 8-2
Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

TABLE 8-4
***G* Select, *H* Select, and *MF* Select Codes Defined in Terms of *FS* Codes**

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

Codifica della control word



(b) Control word

TABLE 8-5
Encoding of Control Word for the Datapath

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
$R0$	000	Register	0	$F = A$	0000	Function	0	No Write	0
$R1$	001	Constant	1	$F = A + 1$	0001	Data in	1	Write	1
$R2$	010			$F = A + B$	0010				
$R3$	011			$F = A + B + 1$	0011				
$R4$	100			$F = A + \overline{B}$	0100				
$R5$	101			$F = A + \overline{B} + 1$	0101				
$R6$	110			$F = A - 1$	0110				
$R7$	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = \text{sr } B$	1101				
				$F = \text{sl } B$	1110				

9 bits

4 bits

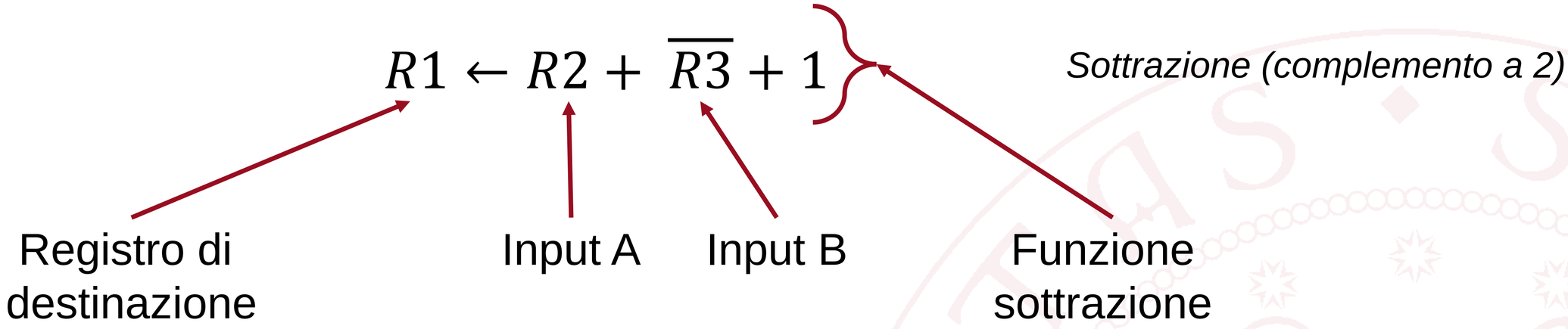
1 bit

1 bit

1 bit

Esempi di control word

Notazione simbolica delle microoperazioni:



Field:	DA	AA	BA	MB	FS	MD	RW
Symbolic:	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
Binary:	001	010	011	0	0101	0	1

Esempi di control word

TABLE 8-6
Examples of Microoperations for the Datapath, Using Symbolic Notation

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + I$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	_	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	_	_	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	_	_	$R3$	Register	_	_	No Write
$R4 \leftarrow$ Data in	$R4$	_	_	_	_	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Esempi di control word

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$							
$R4 \leftarrow$ sl R6							
$R7 \leftarrow R7 + 1$							
$R1 \leftarrow R0 + 2$							
Data out $\leftarrow R3$							
$R4 \leftarrow$ Data in							
$R5 \leftarrow 0$							

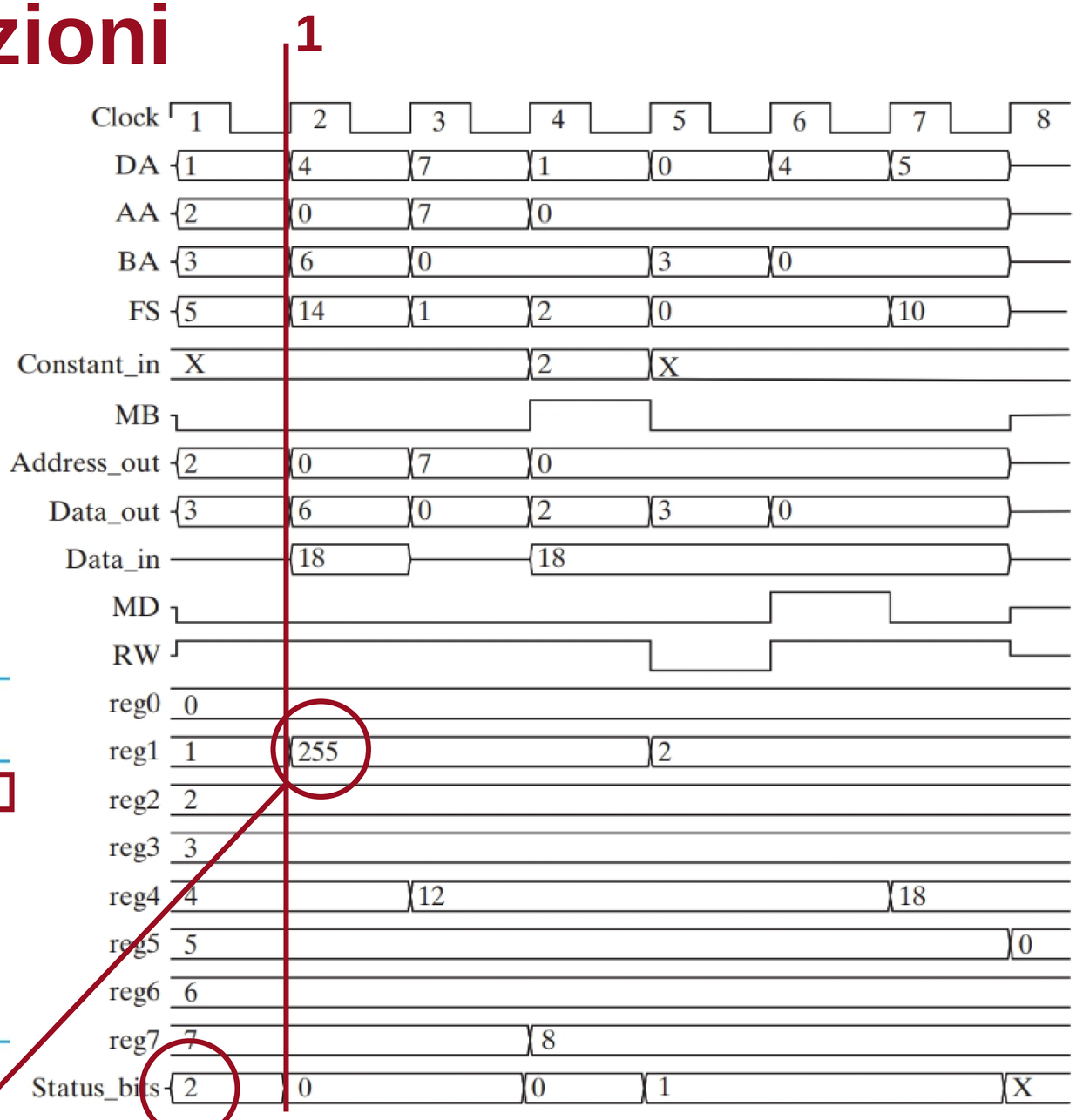
Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1



Risultato disponibile in R1 al prossimo ciclo di clock

FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

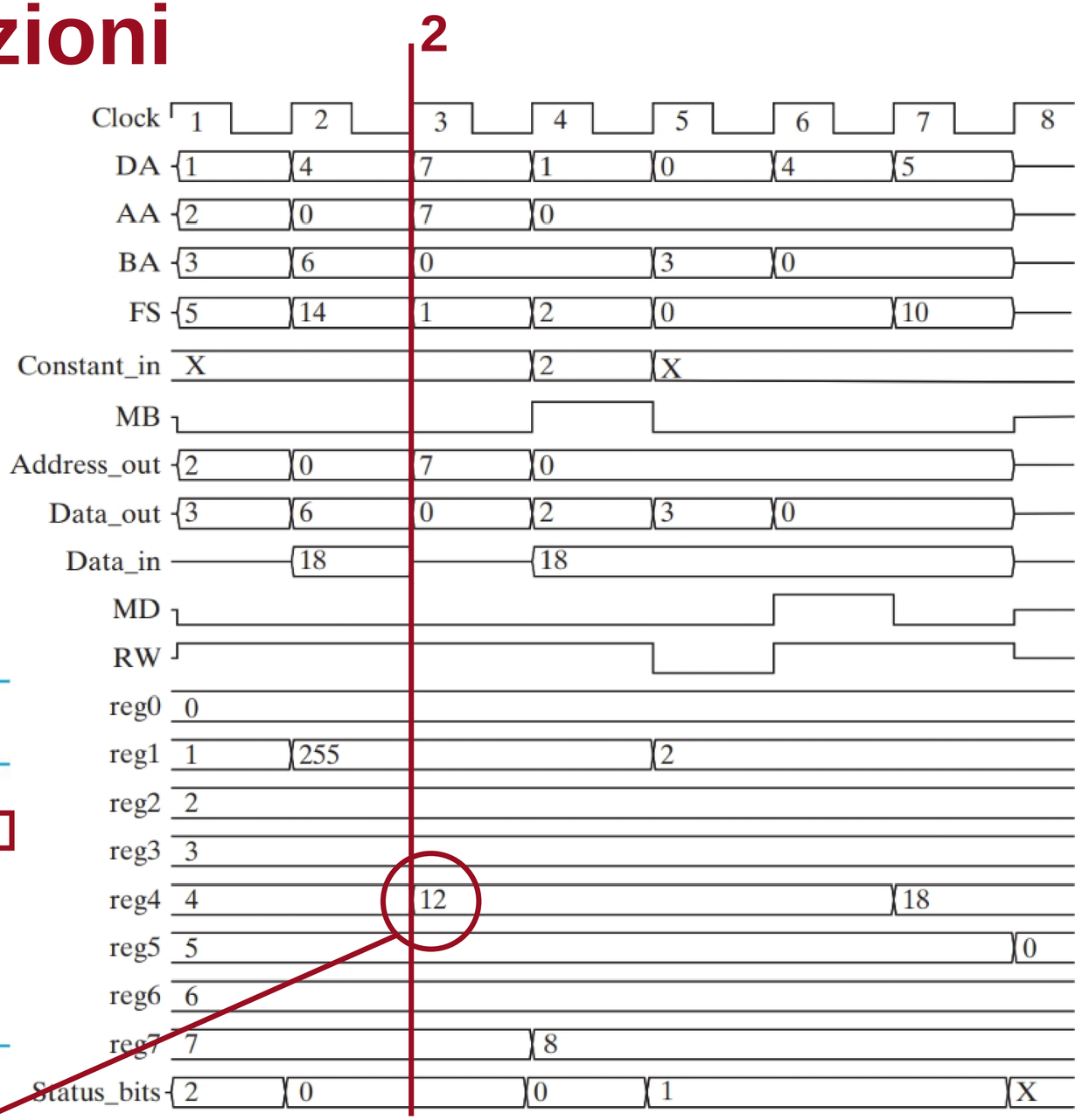
Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1



Risultato disponibile in R4 al prossimo ciclo di clock

FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

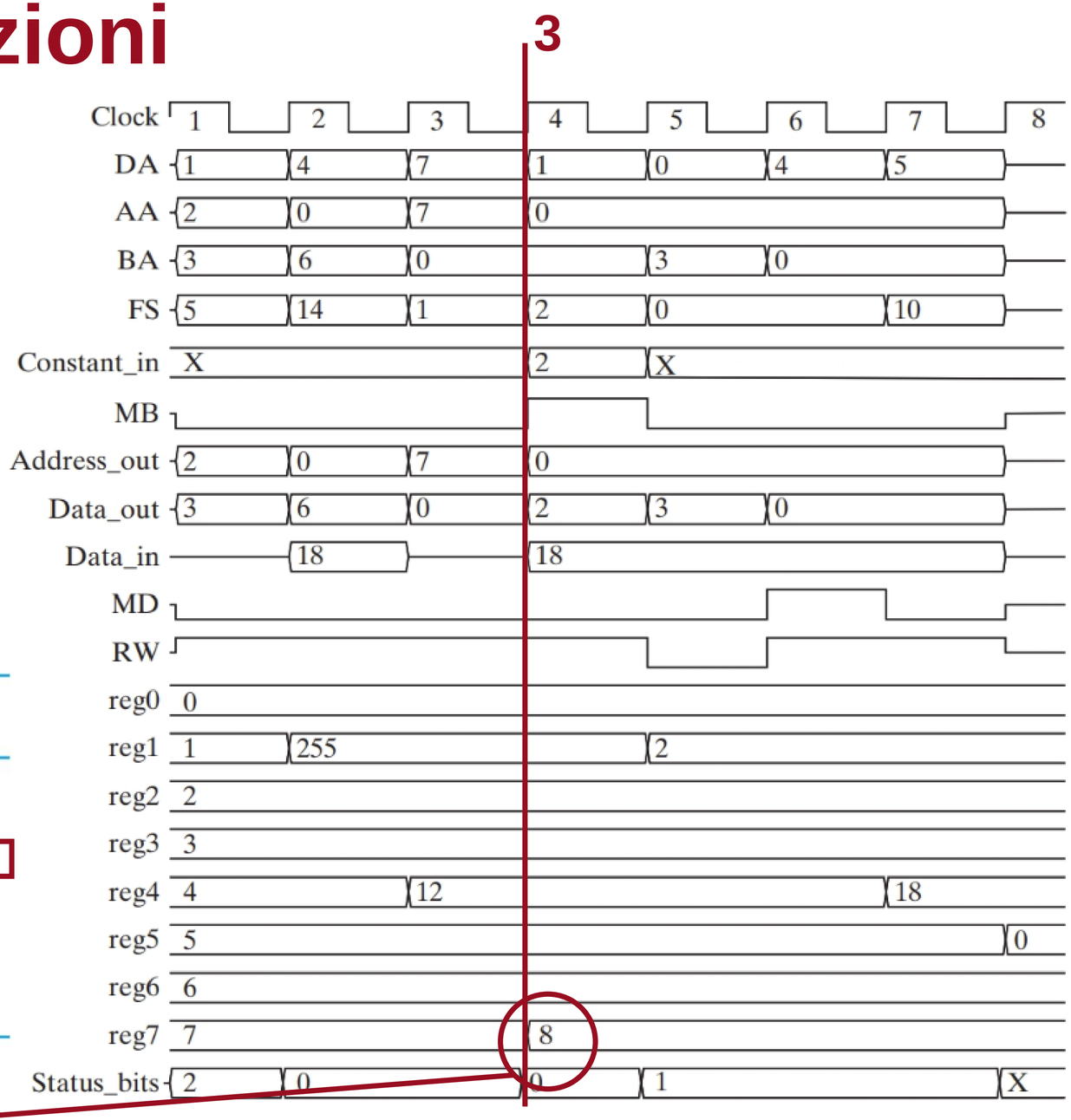
Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1



Risultato disponibile in R7 al prossimo ciclo di clock

FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

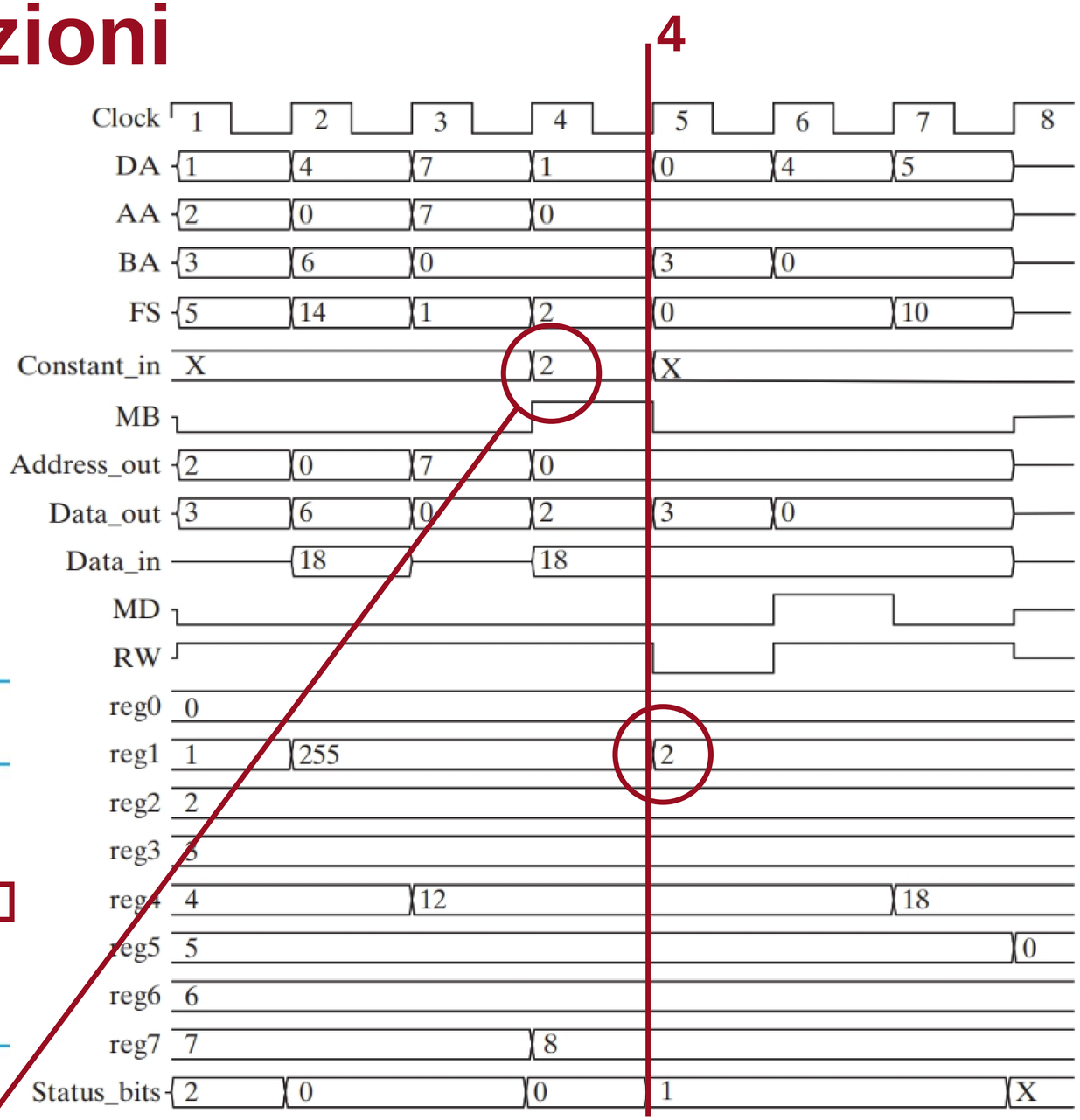


FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

Abbiamo usato la linea constant in il risultato è pronto nel prossimo ciclo di clock

Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

Scriviamo sulla linea Data out il valore di R3
la linea è azzerata nel prossimo ciclo di clock

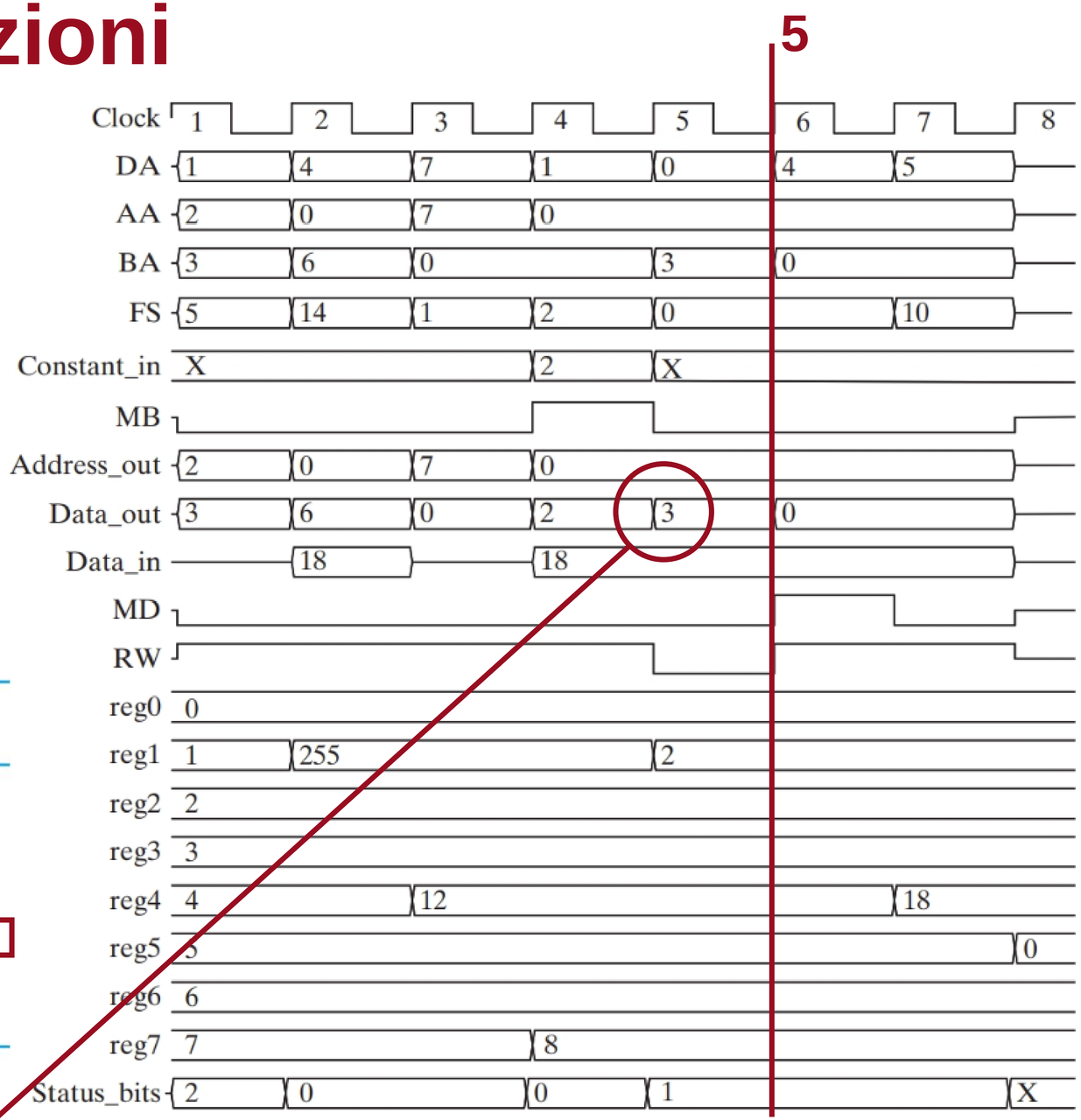


FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

Sequenza di microoperazioni

Assunti:

- Ogni registro contiene 8 bits
- Ogni registro contiene il suo numero in decimale
- Le operazioni seguenti sono eseguite in sequenza:

TABLE 8-7
Examples of Microoperations from Table 8-6, Using Binary Control Words

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	X	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

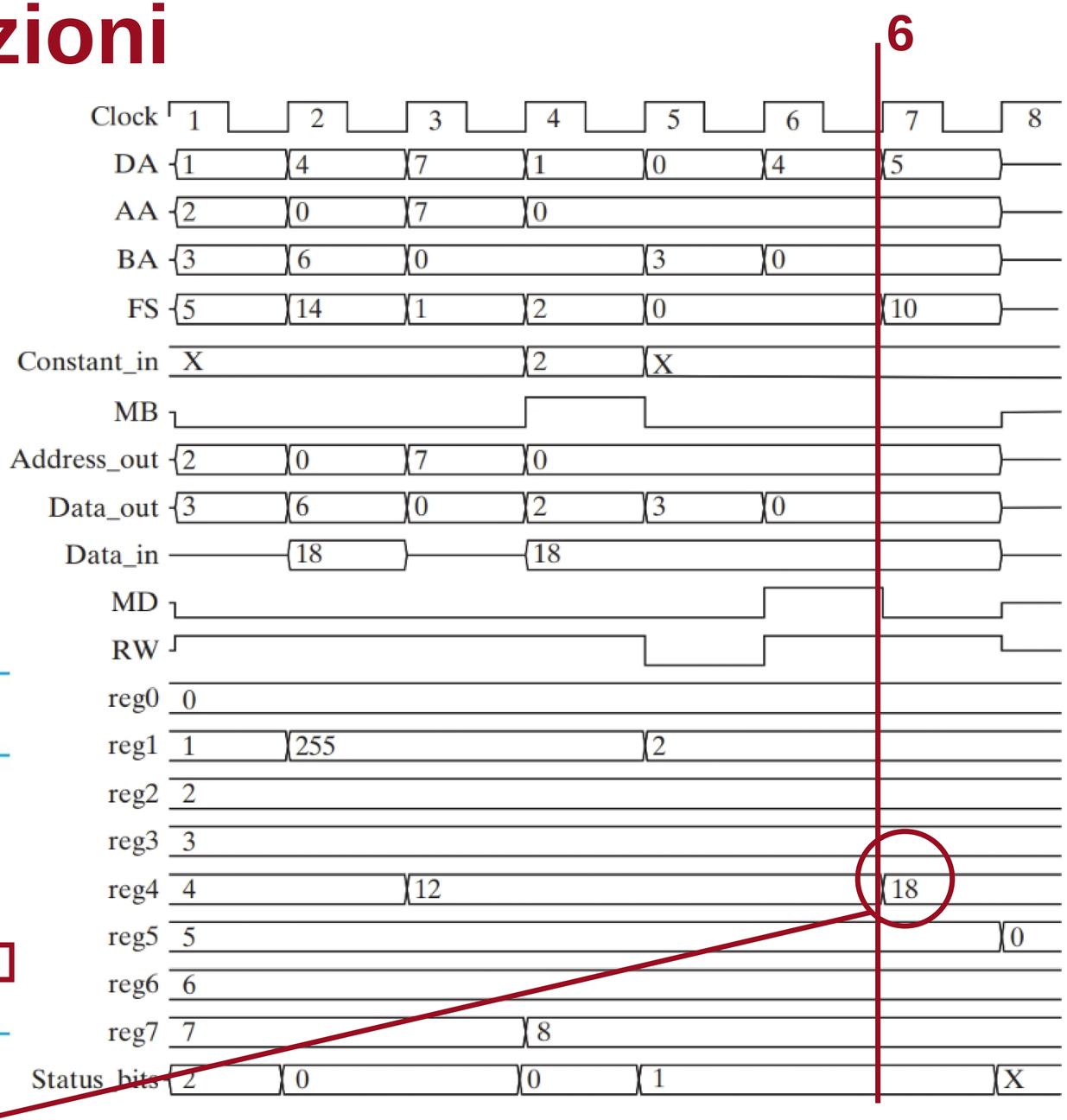
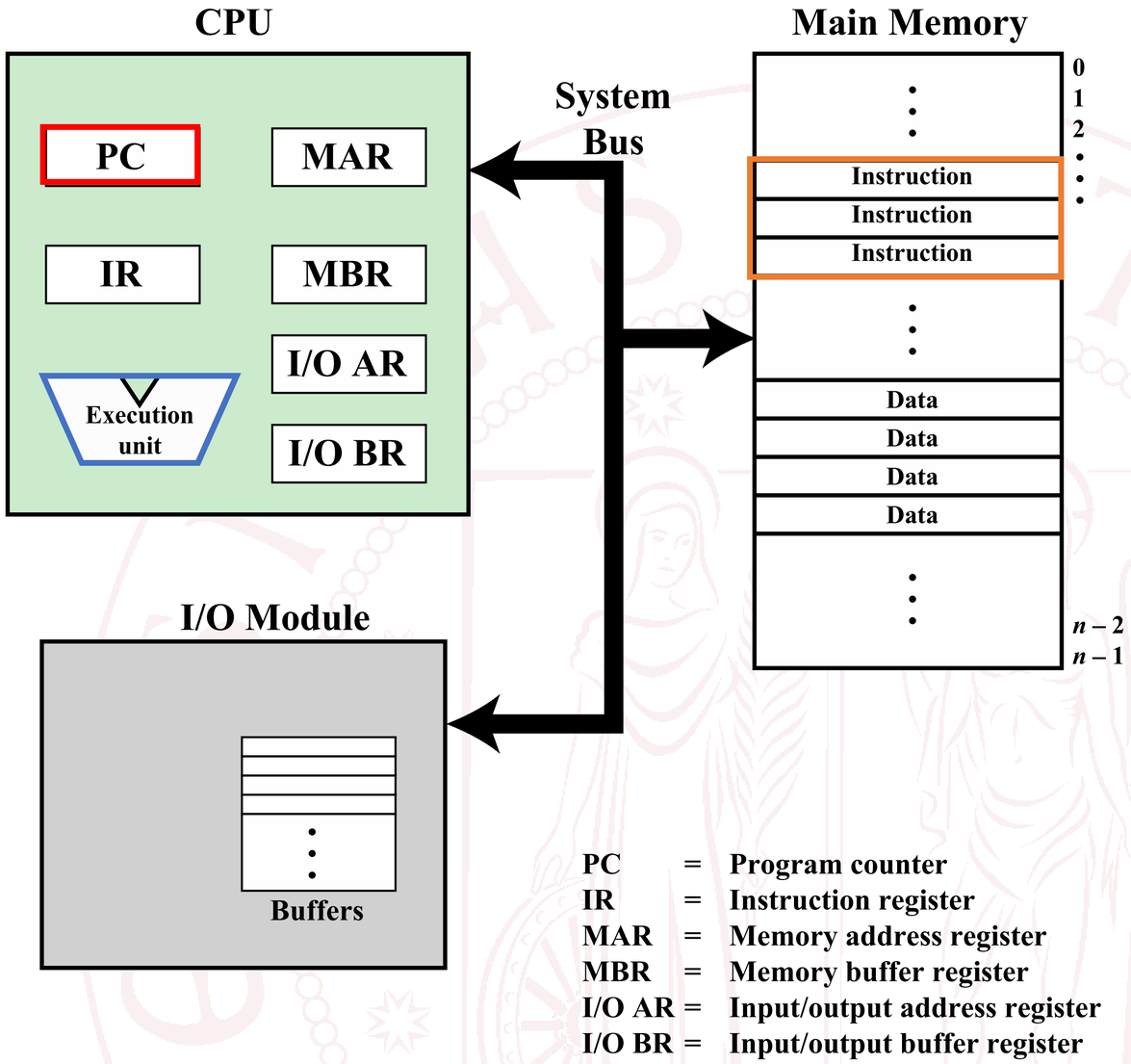


FIGURE 8-12
Simulation of the Microoperation Sequence in Table 8-7

Leggiamo il valore sulla linea Data in (che è 18) il valore è disponibile in R4 nel ciclo dopo

Istruzioni e microoperazioni

- In un sistema programmabile, una parte dell'input per il processore consiste in una sequenza di istruzioni
- Ogni istruzione determina:
  - L'operazione da eseguire
  - Quali operandi usare
  - Dove inserire il risultato dell'operazione
  - (La prossima istruzione da eseguire)
- La sequenza di istruzioni sono salvate in memoria e vengono caricate nei registri
- Per eseguire le istruzioni in sequenza si usa un Program Counter (PC) che punta all'indirizzo di memoria contenente la prossima istruzione
- La lista delle istruzioni compongono il programma



Instruction Set Architecture (ISA)

- Un’istruzione è una collezione di bit che indica l’esecuzione di una specifica operazione
- La collezione di istruzioni viene chiamata **Instruction Set**
- Una completa descrizione dell’instruction set determina un modello astratto del computer, chiamato anche **Instruction Set Architecture (ISA)**
- La più semplice ISA comprende tre componenti principali:
  - Storage resources
  - Instruction formats
  - Instruction specifications

ISA
Storage Resources

Storage resources

Risorse disponibili al programmatore:

- Instruction memory: dove le istruzioni sono immagazzinate
- Data memory: dove i dati sono immagazzinati

- Register file: i registri dove i dati vengono salvati internamente al processore

- Program counter: puntatore alla prossima istruzione

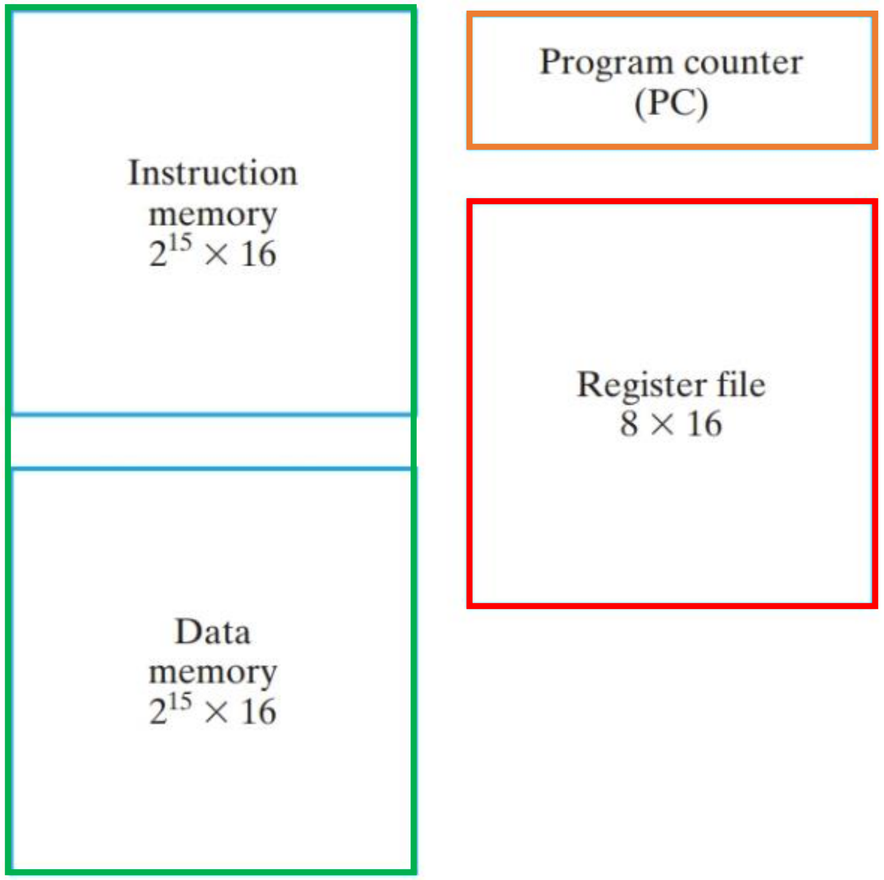


FIGURE 8-13
Storage Resource Diagram for a Simple Computer

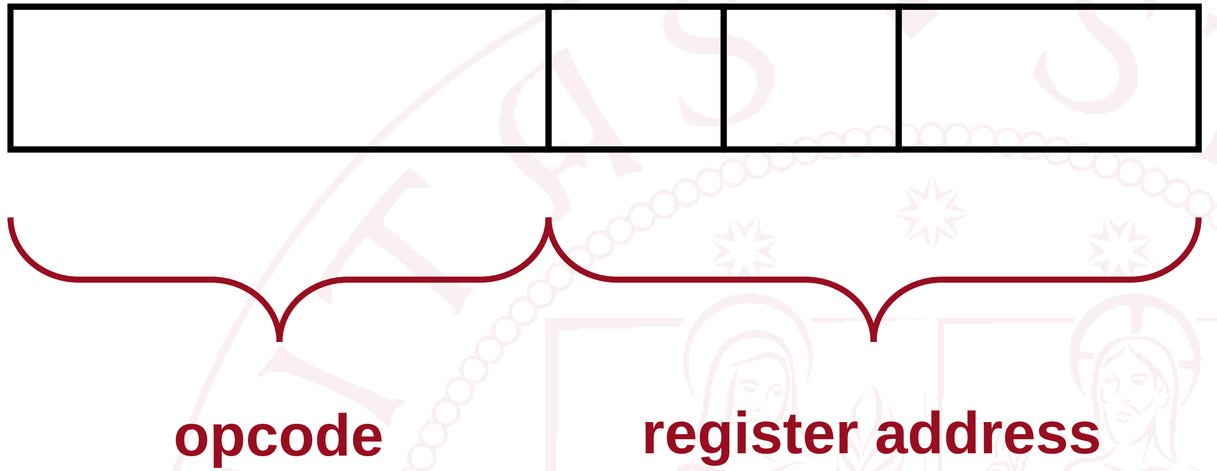
ISA
Instruction Formats

Instruction formats

- Rappresentiamo una istruzione come un box rettangolare che contiene un certo numero di bit

- L'ordine dei bit corrisponde a come è immagazzinata l'istruzione in memoria

- L'istruzione è divisa in campi che specificano:
  - L'operation code (opcode)
  - Indirizzo di registri



Instruction formats: operation code (opcode)

- L'opcode è un gruppo di bits che specifica l'operazione da eseguire (e.g., add, sub, shift..)

- Il numero di bit per l'opcode dipende dal numero totale di possibili operazioni nell'instruction set (m bits corrispondono a 2^m operazioni possibili)

- L'assegnazione di un'operazione per ogni combinazione di bit è fatta nella fase di design

Ad esempio:

- Processore con 128 possibili operazioni
- $2^m = 128 \rightarrow m = 7$ → opcode di 7 bit
- opcode: 0000010
- Quando questo opcode viene riconosciuto, una specifica sequenza di operazioni viene applicata al datapath (in questo caso, quella che permette di fare la somma)

Instruction formats: register addresses, operands

- L’operazione identificata dall’opcode deve usare dati provenienti da registri o dalla memoria
- L’istruzione quindi deve identificare anche l’indirizzo di registro o memoria dove trovare gli **operandi** e dove l'eventuale **risultato** deve essere immagazzinato
- Gli operandi possono essere specificati in due modi:
  - esplicitamente, se nell’istruzione ci sono bit specifici per la sua identificazione
  - implicitamente, se l’identificazione è inclusa nell’operazione stessa

Operandi espliciti

Ad esempio l’operazione di addizione ($R1 \leftarrow A + B$) può contenere:

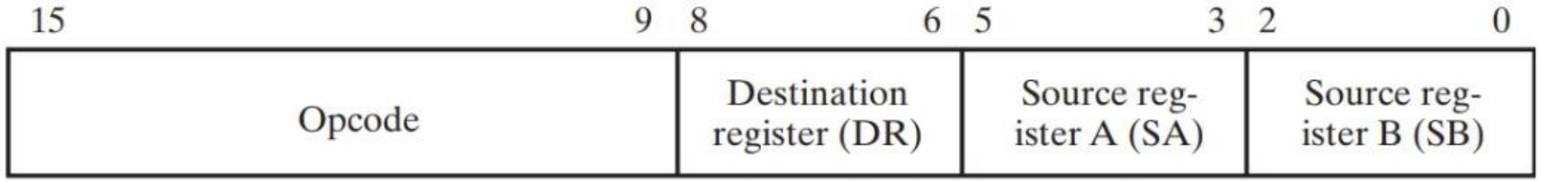
- 2 registri per ognuno degli operandi (A, B)
- 1 registro per il risultato ($R1$)

Operandi impliciti

Ad esempio nell’operazione di incremento ($R7 \leftarrow R7 + 1$) uno dei due operandi è implicitamente $+1$

Instruction formats (1): su registri

- 8 registri: R0-R7
- 16 bits di istruzione



(a) Register

- **opcode**: specifica un'operazione su tre o meno registri
- **Destination register (DR):** destinazione del risultato
- **Source register A (SA):** operando A
- **Source register B (SB):** operando B

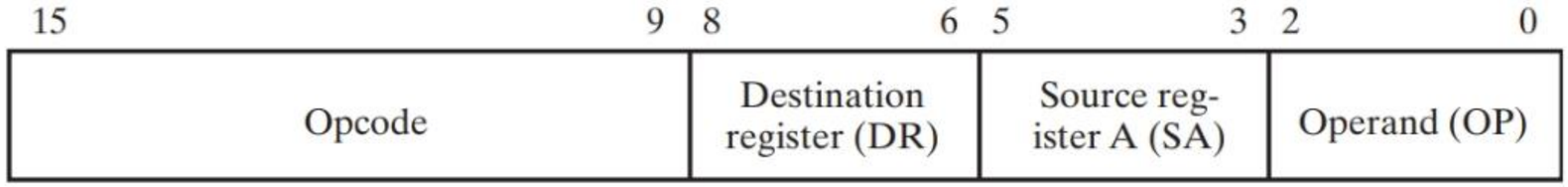
Esempio 1:
- opcode identifica una sottrazione
- SA: 010 (R2); SB: 011 (R3); DR: 001 (R1)
- $R1 \leftarrow R2 - R3$

DR non ha effetto, dato che
Lo store previene la scrittura
su registro (RW=0)

Esempio 2:
- opcode identifica uno store in memoria
- SA: 100 (R4); SB: 101 (R5); DR: XXX
- $M[R4] \leftarrow R5$
- Notare l'uso di M[ ] per identificare un indirizzo in memoria

Instruction formats (2): operando immediato

- 8 registri: R0-R7
- 16 bits di istruzione



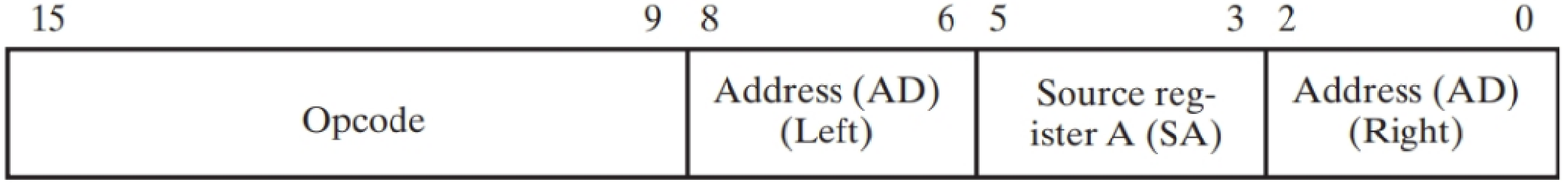
(b) Immediate

- **opcode**: specifica un'operazione su due registri
- **Destination register (DR):** destinazione del risultato
- **Source register A (SA):** operando A
- **Operand (OP):** operando immediato (immediatamente disponibile, non da registri)

Esempio 1:
- opcode identifica una addizione immediata
- SA: 111 (R7); DR: 010 (R2); OP: 011 (value 3)
- $R2 \leftarrow R7 + 3$

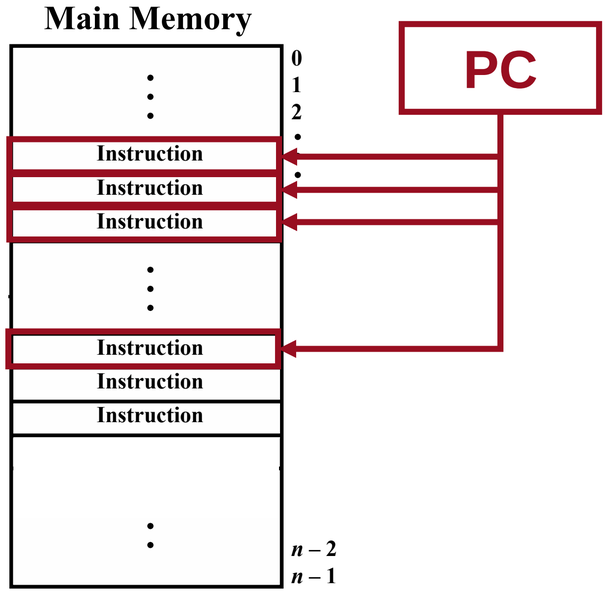
Instruction formats (3): jump e branch

- 8 registri: R0-R7
- 16 bits di istruzione



(c) Jump and branch

Cos'è un'istruzione jump o branch?



- Le istruzioni sono scritte in memoria in modo sequenziale
- Il PC è responsabile del puntamento alla prossima istruzione (viene aggiornato [PC+1] ad ogni ciclo di clock)
- A volte è necessario «saltare» degli indirizzi e eseguire una porzione di istruzioni successiva
- Questi cambi di ordine di esecuzione sono chiamati **jump** o **branch**

Jump e Branch (esempio)

```
17 public class Test {
18
19
20     public static void main( String[] args ) {
21
22         int year = 2023;
23         String pippo = "Have a nice "+year;
24
25
26
27
28
29         System.out.println( pippo );
30
31     }
32
33 }
```

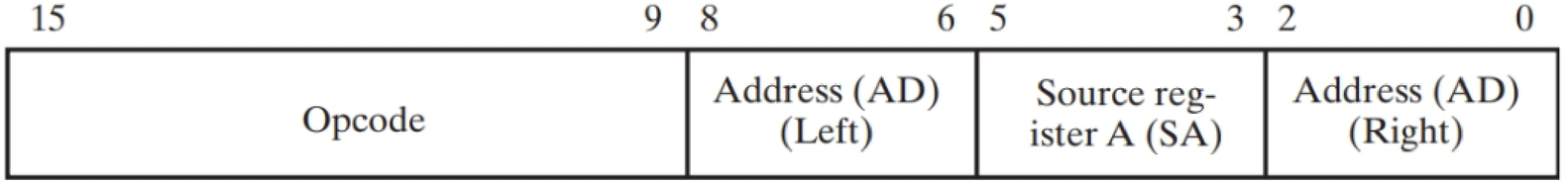
branch

jump

```
1019     /**
1020      * Prints a String and then terminate the line.  This method behaves as
1021      * though it invokes {@link #print(String)} and then
1022      * {@link #println()}.
1023      *
1024      * @param x  The {@code String} to be printed.
1025      */
1026     public void println(String x) {
1027         if (getClass() == PrintStream.class) {
1028             writeln(String.valueOf(x));
1029         } else {
1030             synchronized (this) {
1031                 print(x);
1032                 newLine();
1033             }
1034         }
1035     }
```

Instruction formats (3): branch

- 8 registri: R0-R7
- 16 bits di istruzione



(c) Jump and branch

- **opcode**: specifica un'operazione di jump o branch
- **Source register A (SA):** operando A
- **Address (AD):** split address (left e right) che identifica la destinazione del branch

Il metodo di indirizzamento è basato sull'**address offset** ed è relativo al valore corrente del PC.

- L'address offset è costituito da 6 bits
- L'address offset è trattato come complemento a 2 con segno
- Un'estensione di segno a 16 bit viene eseguita prima dell'addizione al PC

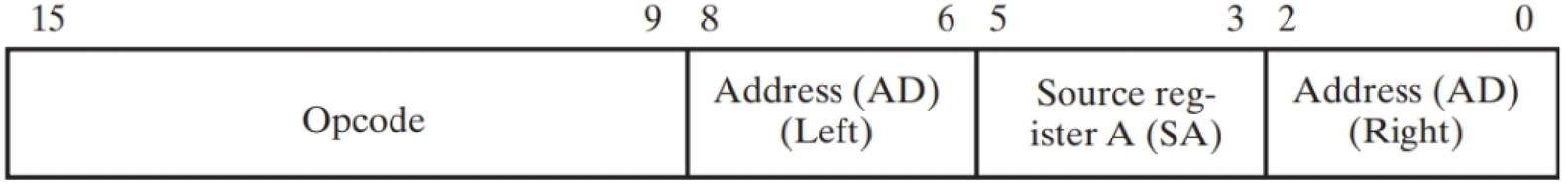
Se il bit più a sinistra di AD è 1:
i 10 bit a sinistra di AD sono portati a 1
offset negativo in complemento a 2

Se il bit più a sinistra di AD è 0:
i 10 bit a sinistra di AD sono portati a 0
offset positivo in complemento a 2

se
sign extension

Instruction formats (3): branch

- 8 registri: R0-R7
- 16 bits di istruzione



(c) Jump and branch

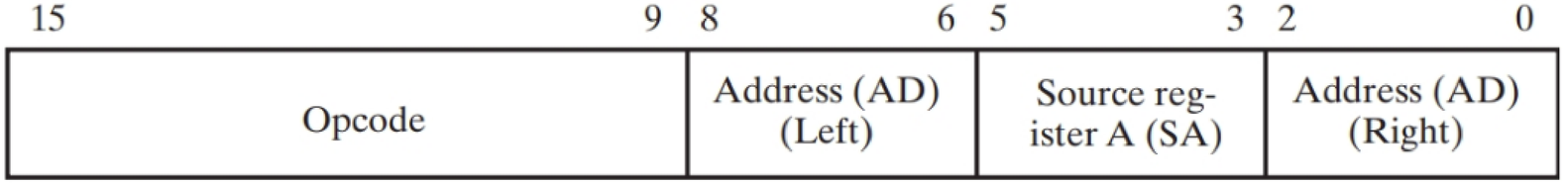
- **opcode**: specifica un'operazione di jump o branch
- **Source register A (SA):** operando A
- **Address (AD):** split address (left e right) che identifica la destinazione del branch

Esempio:

- PC = 55; indirizzo target del branch: 35
- Il branch avviene se e solo se il valore nel registro R6 è uguale a 0
- Opcode specifica un'istruzione branch-on-zero
- SA: 110 (R6); AD (Left): 101; AD (Right): 100 → AD: 101100 (-20)
  - Se R6 = 0 → PC = 55 + (-20) = 35
  - Se R6 ≠ 0 → PC = 55 + 1 = 56

Instruction formats (3): jump

- 8 registri: R0-R7
- 16 bits di istruzione



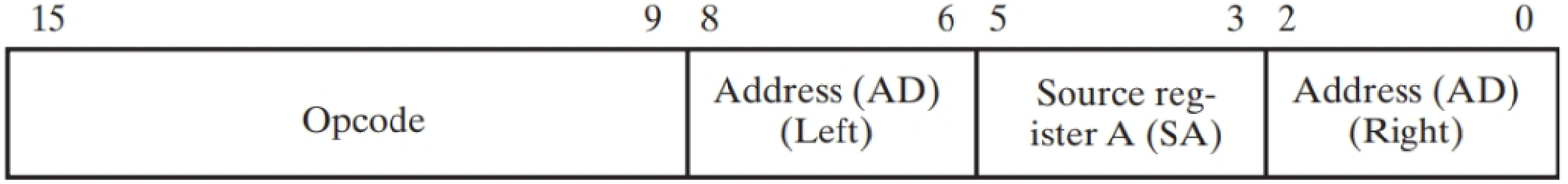
(c) Jump and branch

- **opcode**: specifica un'operazione di jump o branch
- **Source register A (SA):** operando A
- **Address (AD):** xxxxx

- Il valore di SA indica un registro
- Il registro contiene un valore a 16 bit
- Il valore del registro viene usato come indirizzo cui fare il jump

Instruction formats (3): jump e branch

- 8 registri: R0-R7
- 16 bits di istruzione



(c) Jump and branch

- **opcode**: specifica un'operazione di jump o branch
- **Source register A (SA):** operando A
- **Address (AD):** split address (left e right) che identifica la destinazione del branch

- Il branch permette di «saltare» in un intorno del indirizzo corrente del PC
- Il **jump** permette un range maggiore utilizzando il contenuto *unsigned* del registro a 16 bits
- il branch è condizionale, il jump no

ISA
Instruction Specifications

Instruction Specifications

- Esempi di istruzioni per il nostro Simple Computer
- Viene specificato il formato dell'istruzione (ad esempio: ADD RD, RA, RB)
- La descrizione in notazione simbolica: (ad esempio: R[DR] ←R[SA] + R[SB])
- I bit di stato che sono interessati dall'istruzione (ad esempio: N, Z)
- Per tutte le istruzioni con *, il PC viene incrementato di 1 per il prossimo ciclo (PC ←PC + 1)

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

X *CORREGGERE! no PC+1 dopo JUMP !!*

Instruction Specifications

Control word: 16 bit, 15 operazioni

Istruzione: 16 bit, 19 operazioni

TABLE 8-2
Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

TABLE 8-4
***G* Select, *H* Select, and *MF* Select Codes Defined in Terms of *FS* Codes**

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

Configurazioni di FS che non esponiamo al programmatore

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{\text{R[SA]}}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Instruction Specifications

- Descrivono ognuna delle istruzioni che possono essere eseguite dal sistema
- Ad ogni istruzione è associato un opcode e un nome mnemonic
- La notazione dell'istruzione consiste nel mnemonic e nei campi addizionali richiesti

Istruzione	Mnemonic	Formato	Descrizione	Bit di stato
Addition	ADD	RD, RA, RB	R[DR] ←R[SA] + R[SB]	N,Z

- Questa notazione simbolica è convertita nella rappresentazione binaria da un programma chiamato **assembler**

- La rappresentazione binaria è pronta per essere eseguita

Instruction Specifications: esecuzione

- Memoria a 16 bits
- Istruzioni e dati sono in memoria

- Quattro istruzioni:
  1. **Sottrazione**
     (opcode: 5 [0000101])
  2. **Storage**
     (opcode: 32 [0100000])
  3. **Addizione immediata**
     (opcode: 66 [1000010])
  4. **Branch on zero**
     (opcode: 96 [1100000])

TABLE 8-9
Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	00000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

Indirizzi di memoria
dove si trovano le istruzioni
e i dati

Instruction Specifications: esecuzione

1. Address 25: **R1 ← R2 – R3**

0000101 001 010 011

SUB R1, R2, R3

DR:1 SA:2 SB:3

2. Address 35: **M[R4] ← R5**

0100000 000 100 101

ST R4, R5

SA:4 SB:5

3. Address 45: **R2 ← R7 + 3**

1000010 010 111 011

ADI R2, R7, 3

DR:2 SA:7 OP:3

Store in memoria

TABLE 8-9
Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000			Data = 80 (dopo lo store)

Vediamo i valori dei registri:

…	
R4	70
R5	80
…	

Instruction Specifications: esecuzione

4. Address 55: **PC ← PC – 20 (if R6=0)**

- Nell'esempio le word sono di 16 bits
- Nei moderni computer di solito la lunghezza può essere di 32 o 64 bits (range maggiore per operandi immediati e gli indirizzi)
- Anche i registri sono di solito più grandi

TABLE 8-9
Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	00000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

Operazioni e microoperazioni

- Una **computer operation** è un'istruzione binaria immagazzinata nella memoria del computer
  - L'unità di controllo utilizza l'indirizzo fornito dal PC per recuperare l'istruzione dalla memoria (**fetch**)
  - L'unità di controllo decodifica l'istruzione (e.g., opcode, register addresses) per compiere le microoperazioni richieste per l'esecuzione dell'istruzione

- Una **microoperazione** è definita dai bits nella control word ed è decodificata dall'hardware per la sua esecuzione

- L'esecuzione di una computer operation spesso richiede una sequenza di microoperazioni

Computer a ciclo singolo

- Il recupero e l'esecuzione di un'istruzione viene eseguita in un singolo ciclo di clock
- La **memoria M** è connessa all'Address Out, Data Out e Data In
- La **memoria M** ha un segnale di controllo per abilitare/disabilitare la scrittura
- L'**unità di controllo** recupera l'istruzione (PC) e la decodifica
- La **control word** risultante è collegata al datapath e viene eseguita in un ciclo di clock
- A parte il PC (sequenziale), l'instruction decoder, branch control, function unit sono circuiti combinatori (singolo ciclo)

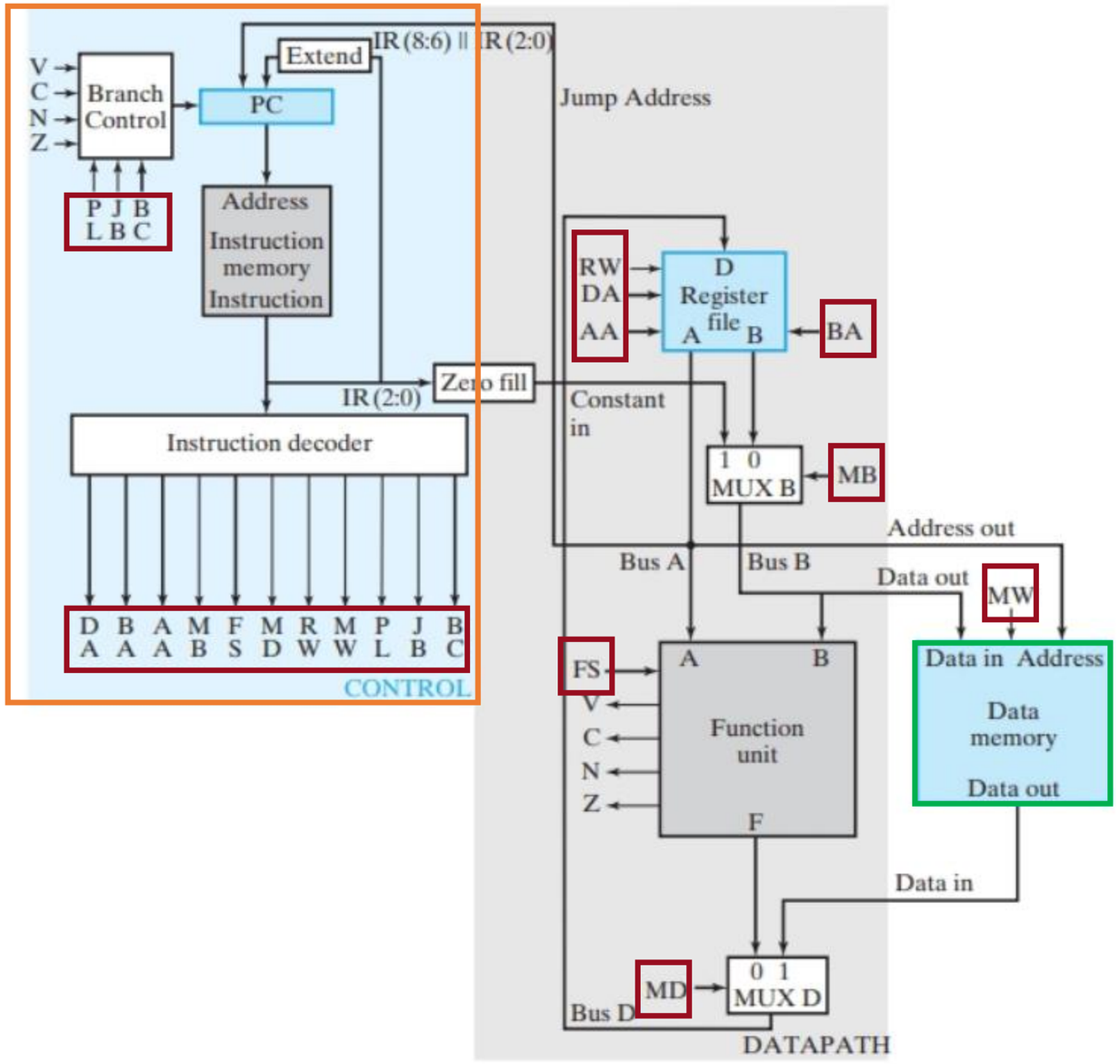


FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Computer a ciclo singolo: esempio (1)

SUB R1, R2, R3 (R1←R2-R3)

- PC = 25
- Fetch dell'istruzione dall'indirizzo di memoria 25
- Instruction decoder:

DA	BA	AA	MB	FS	MD	RW	MW	PL	JB	BC
001	011	010	0	0101	0	1	0	0	X	X
R1	R3	R2	-	SUB	-	-	-	-	-	-

- Microperazione
- PC = PC + 1 = 26

IN UN SINGOLO CICLO DI CLOCK

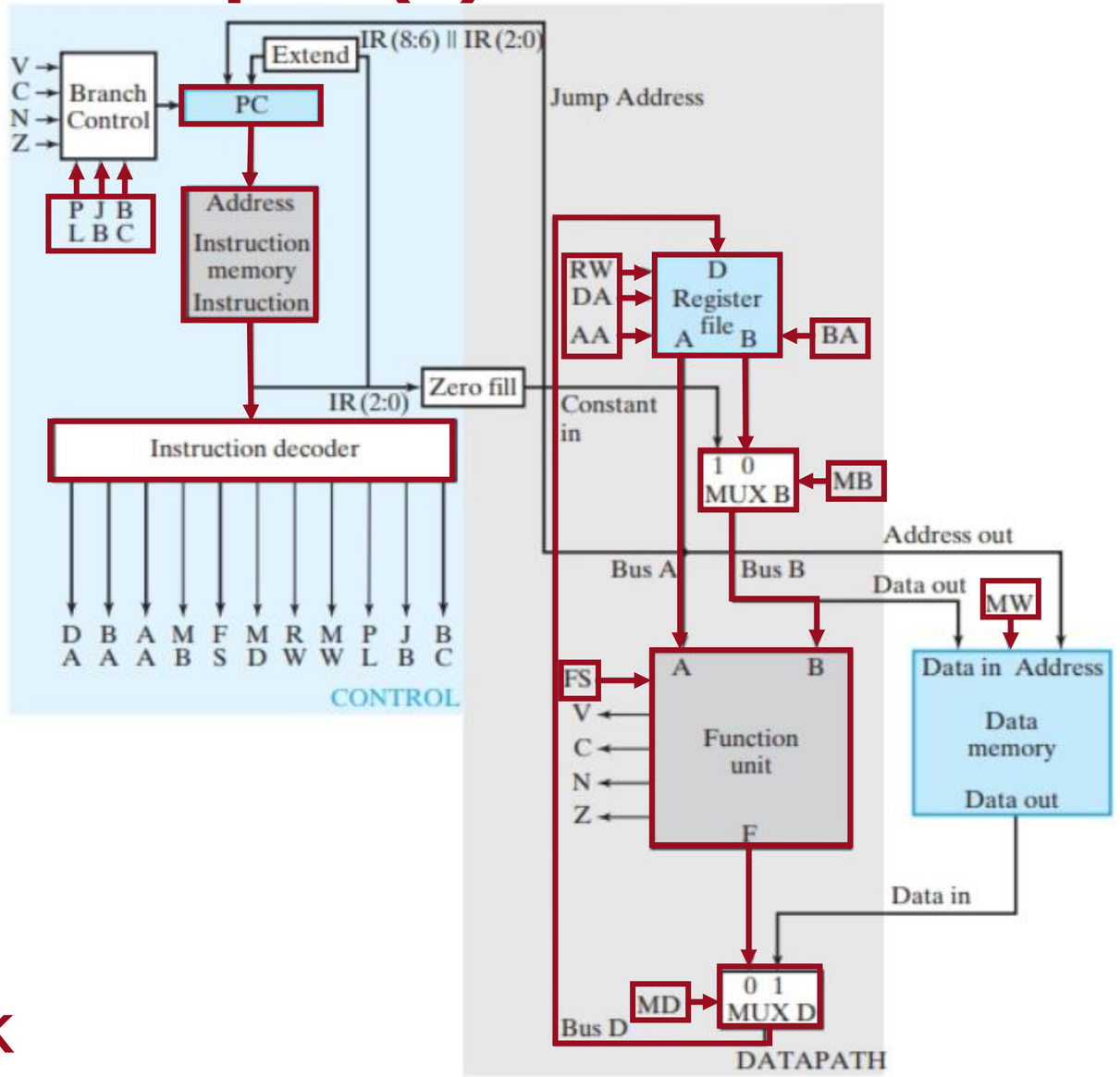


FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Computer a ciclo singolo: esempio (2)

ADI R2←R7 + 3

- PC = 55
- Fetch dell'istruzione dall'indirizzo di memoria 55
- Instruction decoder:

DA	BA	AA	MB	FS	MD	RW	MW	PL	JB	BC
010	011	111	1	1..10	0	1	0	0	X	X
R2	3	R7	-	ADI	-	-	-	-	-	-

- Microperazione
- PC = PC + 1 = 56

IN UN SINGOLO CICLO DI CLOCK

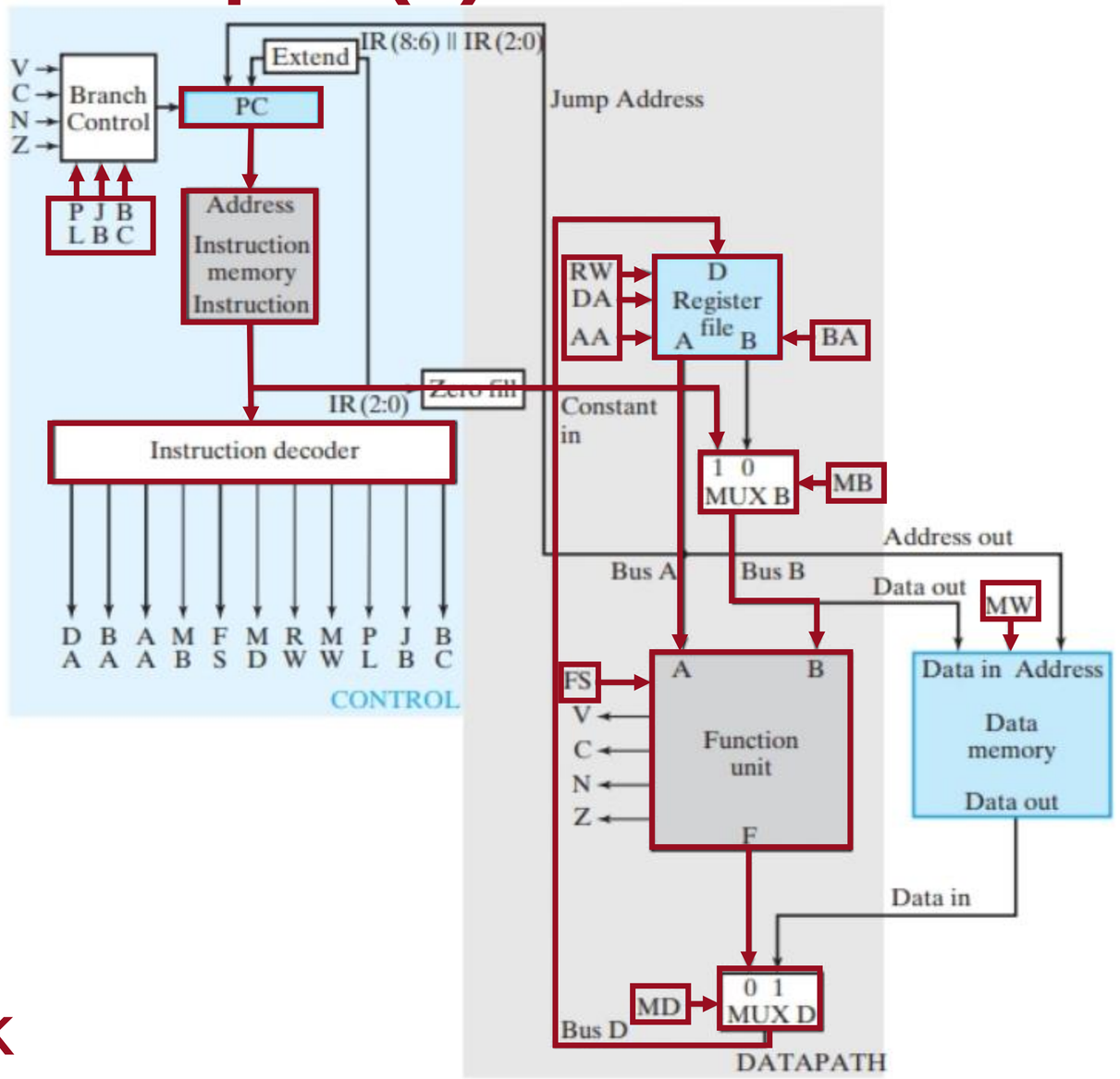


FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Instruction Decoder

- È un circuito combinatorio che può fornire tutte le control words al datapath
- Alcuni campi possono essere ottenuti direttamente dall'istruzione
  (DA, AA, BA, BC ← DR, SA, SB, OpCode[9])
- Anche alcuni bit di controllo sono ottenuti direttamente dall'istruzione
  (MB, MD, JB ← OpCode[15], OpCode[13])
- Altri bit invece richiedono logica combinatoria
- Ci sono tre bit aggiuntivi per il controllo del PC: PL, JB, BC (PC Load, Jump or Branch, Branch Condition)
  - PL = 0 → PC viene incrementato di 1
  - PL = 1 → se c'è un branch o jump
  - PL = 1, JB = 1 → Jump
  - PL = 1, JB = 0 → Branch
  - PL = 1, JB = 0, BC = 0 → Branch on zero
  - PL = 1, JB = 0, BC = 1 → Branch on negative

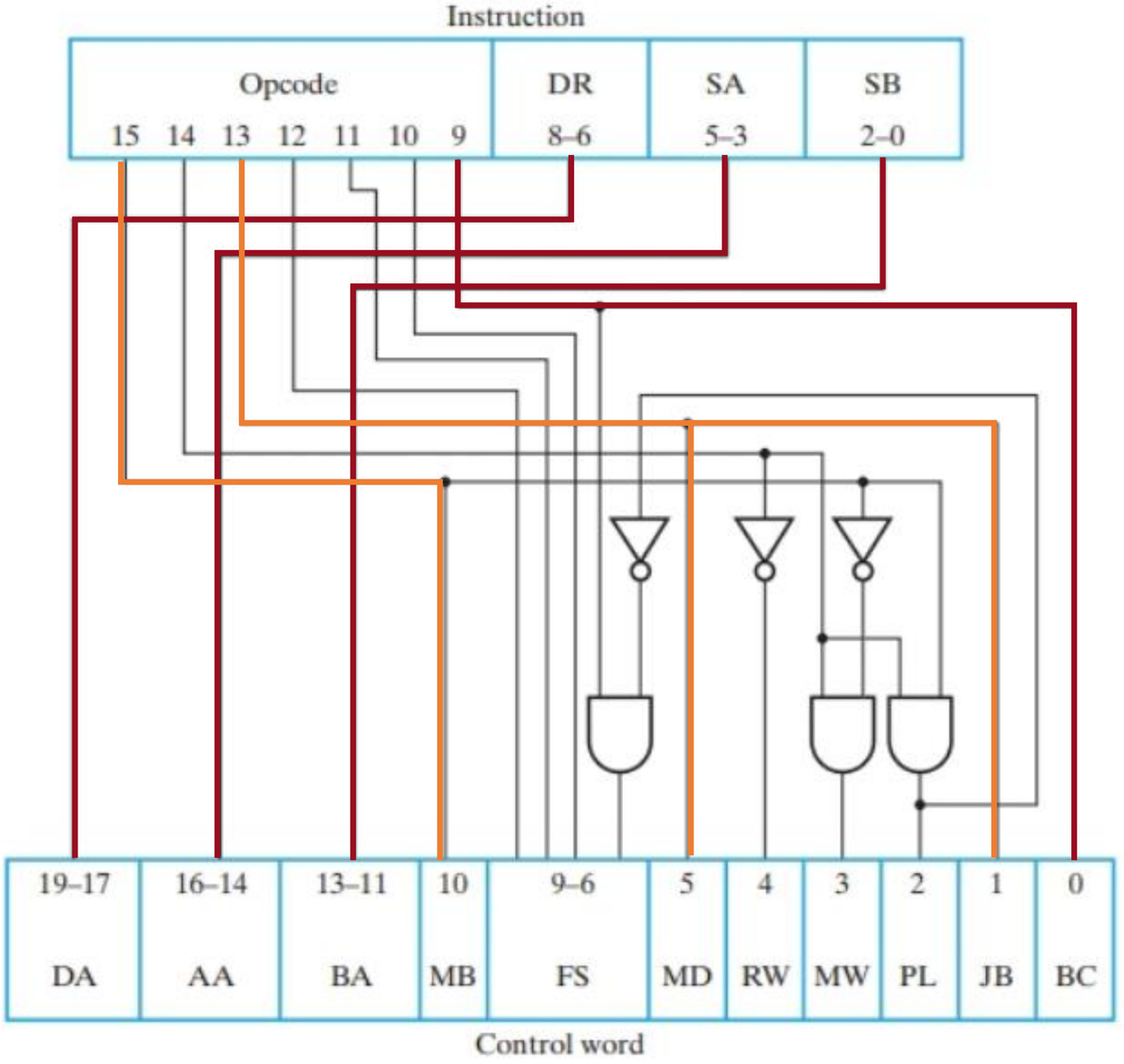


FIGURE 8-16
Diagram of Instruction Decoder

Come costruiamo un instruction decoder?

- Le instruzioni sono raggruppate in gruppi (function types)
- I gruppi sono definiti a seconda dell'utilizzo di specifiche risorse hardware
- Ad esempio il primo tipo utilizza:
  - ALU
  - MUX B per usare registri come sorgente (MB)
  - MUX D per usare output della function unit (MD)
  - Scrive su registri (RW)

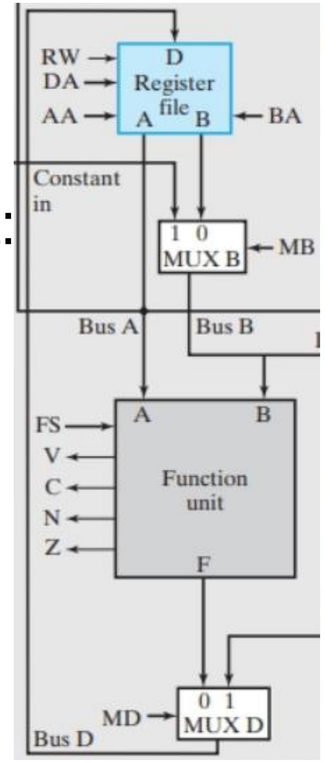


TABLE 8-10
Truth Table for Instruction Decoder Logic

	Instruction Bits				Control-Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
Function-unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
Memory read	0	0	1	X	0	1	1	0	0	X	X
Memory write	0	1	0	X	0	X	0	1	0	X	X
Function-unit operations using register and constant	1	0	0	X	1	0	1	0	0	X	X
Conditional branch on zero (Z)	1	1	0	0	X	X	0	0	1	0	0
Conditional branch on negative (N)	1	1	0	1	X	X	0	0	1	0	1
Unconditional jump	1	1	1	X	X	X	0	0	1	1	X

Design dell'instruction decoder

L’ottimizzazione di questa tabella di verità porta all’implementazione del circuito visto in precedenza

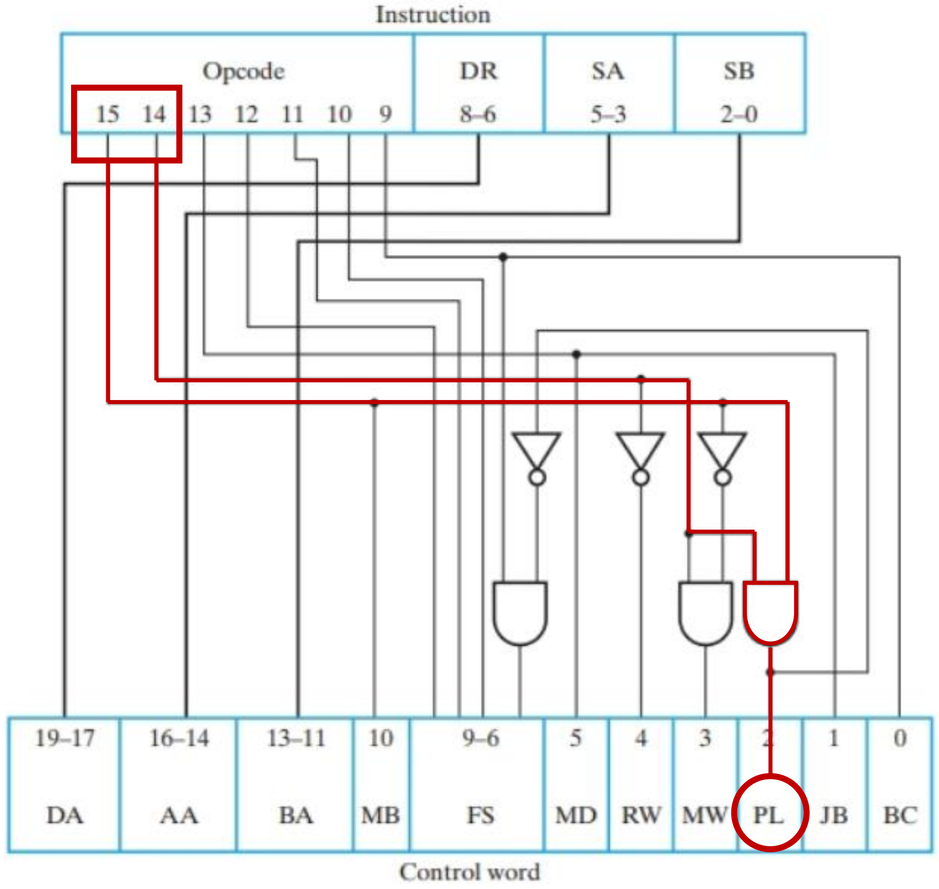


FIGURE 8-16
Diagram of Instruction Decoder

TABLE 8-10
Truth Table for Instruction Decoder Logic

	Instruction Bits				Control-Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
Function-unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
Memory read	0	0	1	X	0	1	1	0	0	X	X
Memory write	0	1	0	X	0	X	0	1	0	X	X
Function-unit operations using register and constant	1	0	0	X	1	0	1	0	0	X	X
Conditional branch on zero (Z)	1	1	0	0	X	X	0	0	1	0	0
Conditional branch on negative (N)	1	1	0	1	X	X	0	0	1	0	1
Unconditional jump	1	1	1	X	X	X	0	0	1	1	X

Esempio: possiamo derivare PL = 1 per le operazioni di Jump e Branch dai bit 14 e 15

Design dell'instruction decoder

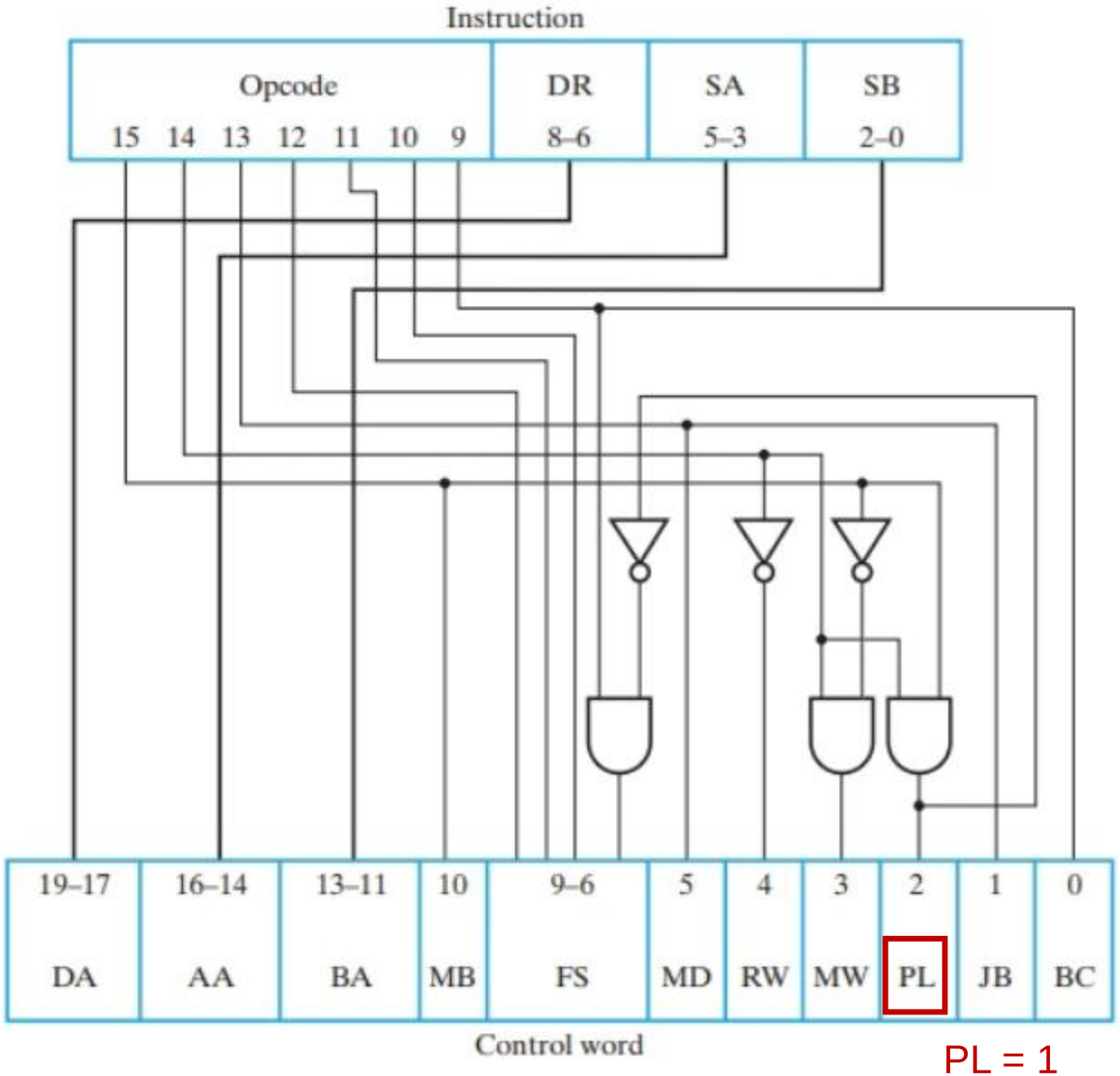


FIGURE 8-16
Diagram of Instruction Decoder

PL = 1
BIT 15 e 14 = 1

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Esempi di istruzioni

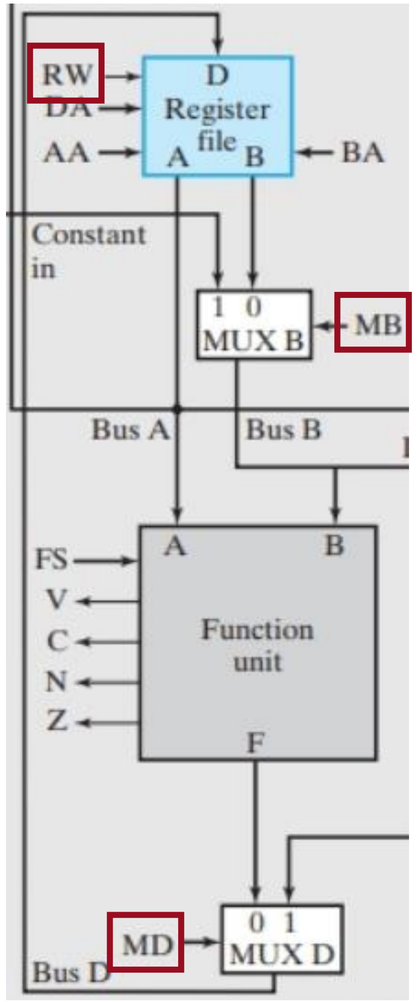
TABLE 8-11
Six Instructions for the Single-Cycle Computer

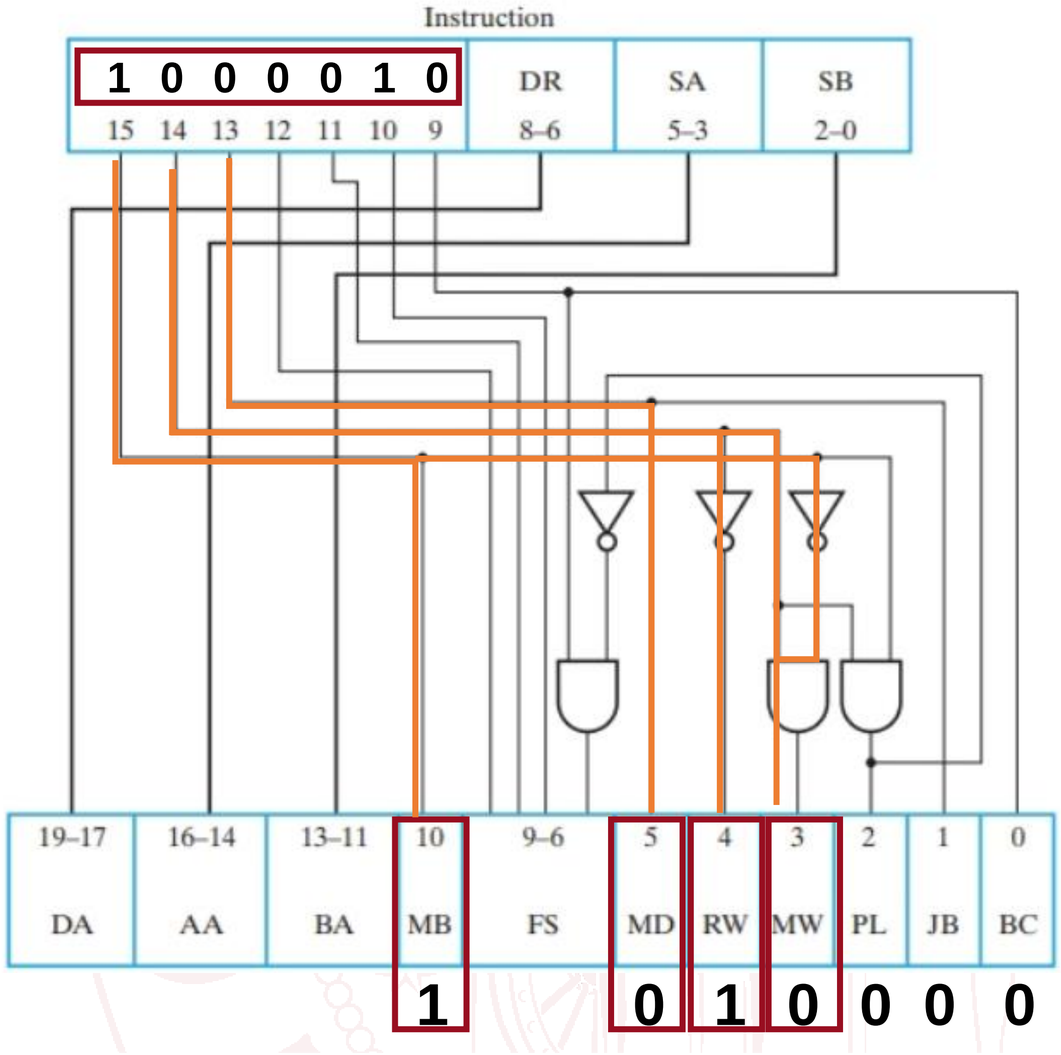
Operation Code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000010	ADI	Immediate	Add immediate operand	$R[\text{DR}] \leftarrow R[\text{SA}] + \text{zf } I(2{:}0)$	1	0	1	0	0	0	0
0010000	LD	Register	Load memory content into register	$R[\text{DR}] \leftarrow \text{M}[R[\text{SA}]]$	0	1	1	0	0	1	0
0100000	ST	Register	Store register content in memory	$\text{M}[R[\text{SA}]] \leftarrow R[\text{SB}]$	0	1	0	1	0	0	0
0001110	SL	Register	Shift left	$R[\text{DR}] \leftarrow \text{sl } R[\text{SB}]$	0	0	1	0	0	1	0
0001011	NOT	Register	Complement register	$R[\text{DR}] \leftarrow \overline{R[\text{SA}]}$	0	0	1	0	0	0	1
1100000	BRZ	Jump/Branch	If $R[\text{SA}] = 0$, branch to $PC + \text{se } AD$	If $R[\text{SA}] = 0$, $PC \leftarrow PC \text{ se AD}$ If $R[\text{SA}] \neq 0$, $PC \leftarrow PC + 1$	1	0	0	0	1	0	0

Esempi di istruzioni

Operation Code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000010	ADI	Immediate	Add immediate operand	$R[DR] \leftarrow R[SA] + zf\ I(2{:}0)$	1	0	1	0	0	0	0

- MB = 1, usa costante nella somma
- MD = 0, usa output function unit
- RW = 1, scrivi su registro
- MW = 0, non scrivere su memoria
- PL, JB, BC = 0, no jump/branch





Esempio di programma

- Implementazione dell'espressione aritmetica: $83 - (2 + 3)$
- Dati del problema:
  - R3 = 248
  - M[248] = 2
  - M[249] = 83
  - Risultato salvato in M[250]

1	LD	R1, R3	Carica R1 con il contenuto della memoria all'indirizzo 248 (R1 = 2)
2	ADI	R1, R1, 3	Addizione immediata di R1 + 3 (R1 = 5)
3	NOT	R1, R1	Complemento di R1
4	INC	R1, R1	Incremento di R1 (complemento a due, R1 = -5)
5	INC	R3, R3	Incremento di R3 (R3 = 249)
6	LD	R2, R3	Carica su R2 il contentuo della memoria all'indirizzo 249 (R2 = 83)
7	ADD	R2, R2, R1	Somma il contenuto di R2 e R1 e scrivilo su R2 (R2 = 83 + (-5) = 78)
8	INC	R3, R3	Incremento di R3 (R3 = 250)
9	ST	R3, R2	Scrittura di R2 su M[R3] (M[250] = 78)

Limiti del computer a ciclo singolo

- Operazioni complesse che non possono essere eseguite in un singolo ciclo di clock
  - ad esempio: moltiplicazioni

- Accesso alla memoria
  - Un ciclo di clock per ottenere l'istruzione di lettura/scrittura
  - Un ciclo per leggere/scrivere il dato

- Massima frequenza di clock dipendente dal ritardo di propagazione del segnale nel circuito (caso peggiore)
  - In caso di un ritardo di 9.8 ns → Max Freq Clock= 102 MHz

Computer a ciclo multiplo

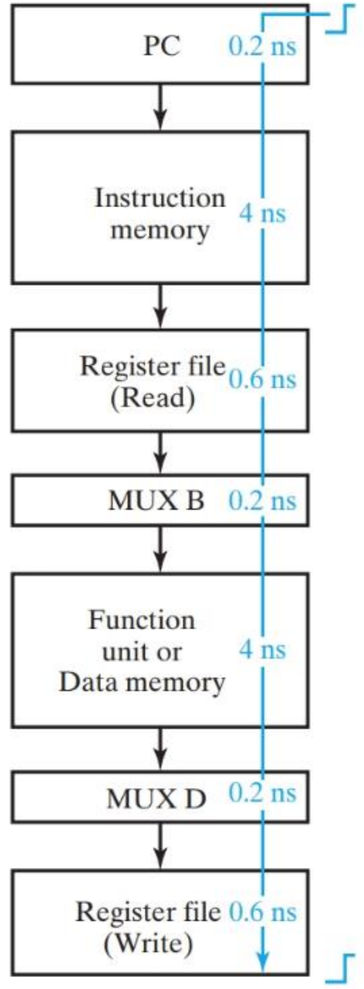


FIGURE 8-17
Worst-Case Delay Path in Single-Cycle Computer

Svolgimento

Vogliamo fare la somma dei numeri di un vettore.

- Il vettore ha n elementi
- Il primo elemento si trova alla riga 27 della memoria
- Nella riga 27-1 è scritta la lunghezza del vettore (che ha 4 elementi)
- Il risultato della somma va scritto nella riga 32
- Nel registro R0 è scritto il numero 26
- Nel registro R1 è scritto il numero 32

Registro	Valore
0	26
1	32
2	…
3	…

Posizione in memoria	Valore
26	4
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Soluzione

```
// implementazione assembly senza ciclo for
// CODICE CREATO IN CLASSE

     // faccio la prima somma:
     // leggo primo e secondo elemento
     // e li sommo
     // uso R0 come indirizzo da usare per gli accessi in memoria
INC   R0    R0          // r0 = r0+1     r0 = 27
LD    R2    R0          //               r2 = 11
INC   R0    R0          // r0 = r0+1     r0 = 28
LD    R3    R0          //               r3 = 22
ADD   R2    R2    R3    // r2 = r2+r3    r2 = 33

     // leggo terzo elemento e lo sommo
INC   R0    R0          // r0++          r0 = 29
LD    R4    R0          //               r4 = 33
ADD   R2    R2    R4    //               r2 = 66

     // leggo quarto elemento e lo sommo
INC   R0    R0          // r0++          r0 = 30
LD    R3    R0          //               r3 = 44
ADD   R5    R2    R3

     // salvo il risultato in memoria
ST    R1    R5
```

Registro	Valore
0	26
1	32
2	
3	
4	
5	
6	
7	

Posizione in memoria	Valore
26	4
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Soluzione

```
// implementazione assembly senza ciclo for
// SOLUZIONE UGUALE ALLA PRECEDENTE MA FATTA DAL PROF E COMMENTATA

INC R4, R0              // ho creato il numero 27
                        // in R4 abbiamo 27

LD R5, R4               // carico in R5 il valore alla
                        // riga 27 della memoria
                        // in R5 abbiamo 11

INC R4, R4              // incrementiamo l'indice
                        // in R4 abbiamo 28

LD R6, R4               // carico in R6 il valore della
                        // riga 28 della memoria
                        // in R6 abbiamo 22

ADD R5, R5, R6          // sommiamo in R5 il valore 22
                        // in R5 abbiamo 11+22 = 33

INC R4, R4              // in R4 abbiamo 29
LD R6, R4               // in R6 abbiamo 33
ADD R5, R5, R6          // in R5 abbiamo 33 + 33 = 66

INC R4, R4              // in R4 abbiamo 30
LD R6, R4               // in R6 abbiamo 44
ADD R5, R5, R6          // in R5 abbiamo 66 + 44 = 110

ST R1, R5               // salviamo il risultato
                        // nella riga 32 della memoria
```

Registro	Valore
0	26
1	32
2	
3	
4	
5	
6	
7	

Posizione in memoria	Valore
26	4
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Esercizio

Vogliamo fare la somma dei numeri di un vettore.

- Il vettore ha n elementi
- Il primo elemento si trova alla riga 27 della memoria
- Nella riga 27-1 è scritta la lunghezza del vettore (che ha 4 elementi)
- Il risultato della somma va scritto nella riga 32
- Nel registro R0 è scritto il numero 26
- Nel registro R1 è scritto il numero 32

VOGLIAMO USARE UN CICLO

Registro	Valore
0	26
1	32
2	…
3	…

Posizione in memoria	Valore
26	200
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Soluzione

```
// SOLUZIONE FATTA IN CLASSE
// pseudo codice java e traduzione dei comandi
int num = 200;                      LD
int[] pippo = int[ num ];
int somma = 0;                      LDI
for( int i=0; i< num; i++ ) {       LDI   i=0
                                    SUB   i - num
                                    BRN   se i >=num salta a "qui"
                                    INC   i++
     int elem = pippo[ i ];         LD
     somma = somma + elem;          ADD
}                                   JMP   riga 5
qui

--------------------
// implementazione assembly con ciclo

LD R2, R0        // num = M[r0] = 200
LDI R3, 0        // somma = 0

LDI R4, 0        // i=0
SUB R5, R2, R4   // r5 = num-i
BRZ R5, 7        // se i==num salta il ciclo
INC R0, R0       // in r0 abbiamo 27
LD R6, R0        // elem = pippo[r0]
ADD R3, R3, R6   // somma += elem (r6)
INC R4, R4       // i++
LDI R7, 18
JMP R7
ST R1, R3        // salvo il risultato in memoria
```

Registro	Valore
0	26
1	32
2	
3	
4	
5	
6	
7	

Posizione in memoria	Valore
26	4
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Soluzione

```
// SOLUZIONE FATTA DAL PROF E COMMENTATA
// pseudo codice java
int[] pippo = new int[4];

int i=0;                          // indice
int contatore = 4;                // contatore
int somma = pippo[i];             // accumulatore
contatore--;
while( contatore > 0 ) {
      i++;
      int val = pippo[i];
      somma = somma + val;
      contatore--;
}

--------------------
// implementazione assembly con ciclo

INC R4, R0              // in R4 ho indice: lo inizializzo a 27
LD R5, R4               // salvo nell'accumulatore il valore 11
LD R7, R0               // R7 contiene il contatore, parte da 4
DEC R7, R7              // R7 contiene 3

ciclo:
INC R4, R4              // in R4 abbiamo l'indice i: lo incremento
LD R6, R4               // in R6 carichiamo il valore
ADD R5, R5, R6          // in R5 aggiorniamo l'accumulatore
DEC R7, R7              // R7 contiene 2, 1, 0

BRZ R7, 2               // se R7 e' zero, esco dal ciclo (non eseguo jump)
JUMP ciclo              // re-inizio ciclo

ST R1, R5               // salvo il risultato in memoria
```

Registro	Valore
0	26
1	32
2	
3	
4	
5	
6	
7	

Posizione in memoria	Valore
26	4
27	11
28	22
29	33
30	44
31	…
32	…

TABLE 8-8
Instruction Specifications for the Simple Computer

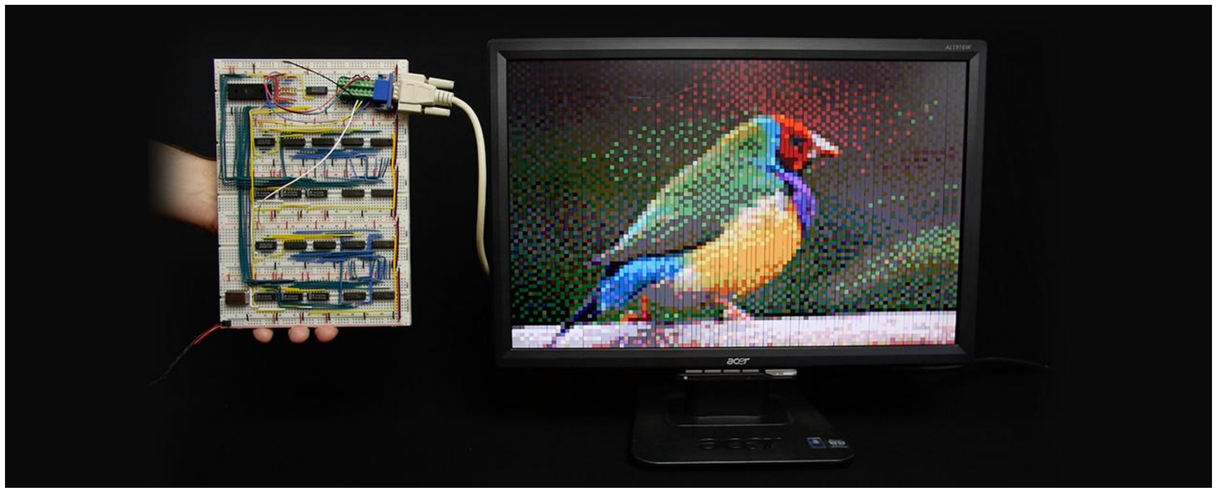
Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

…dopo tanto software

- Se non sapete cosa vedere durante il weekend, vi propongo un video di programmazione hardware

- La costruzione di una scheda video tramite logiche discrete, tanti fili, qualche schema, della documentazione …e molta abilità a piegare i fili!



Video uno (la base)
(bello ad esempio vedere implementare i contatori)
https://www.youtube.com/watch?v=I7rce6IQDWs

Video due (la conclusione)
(con tanto di programmazione di eeprom)
https://www.youtube.com/watch?v=uqY3FMuMuRo

Ben Eater https://eater.net/