

Corso di
Architettura degli Elaboratori
a.a. 2018/2019

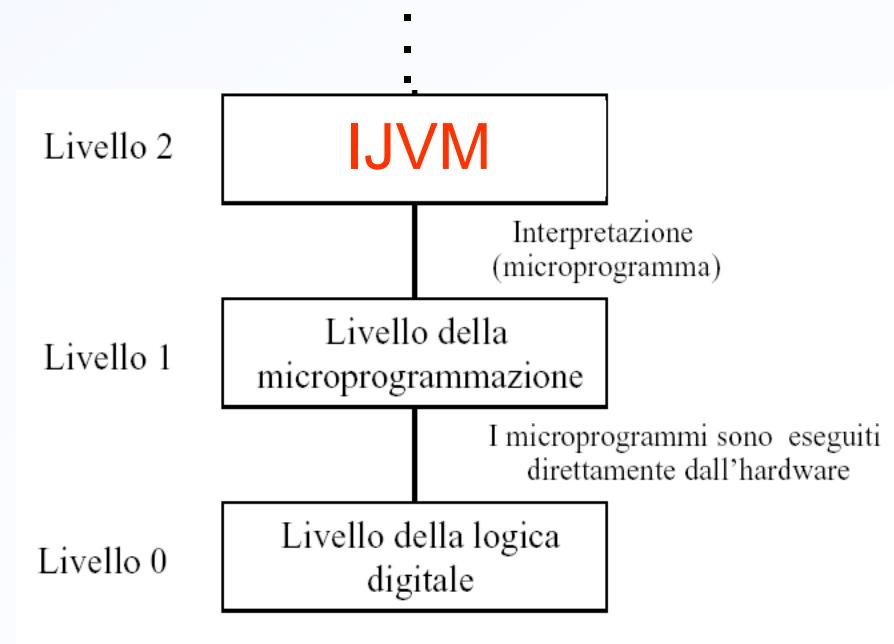
Il linguaggio della microarchitettura:
data path e formato delle microistruzioni

“...capii che la soluzione era trasformare l’unità di controllo in un calcolatore in miniatura aggiungendo una seconda matrice per determinare il flusso di controllo al microlivello e fornendo microistruzioni condizionali”

Wilkes, 1985

Esempio: una microarchitettura per IJVM

- Consideriamo come livello ISA un sottoinsieme della Java Virtual Machine che contiene solo istruzioni su numeri interi:
 - IJVM** (Integer Java Virtual Machine)
- Il microprogramma avrà il compito di leggere (fetch), decodificare ed eseguire le istruzioni IJVM mediante “cicli di data-path”
- Assumeremo che l'interprete risieda in una **ROM** dedicata



ISA IJVM

- Insieme di 20 istruzioni
- Ciascuna costituita da un codice operazione (*opcode*) ed eventualmente un operando (*memory offset* oppure *costante*)
- Troppo semplice per scrivere programmi complicati
- Troppo complicata per essere eseguita direttamente dai circuiti

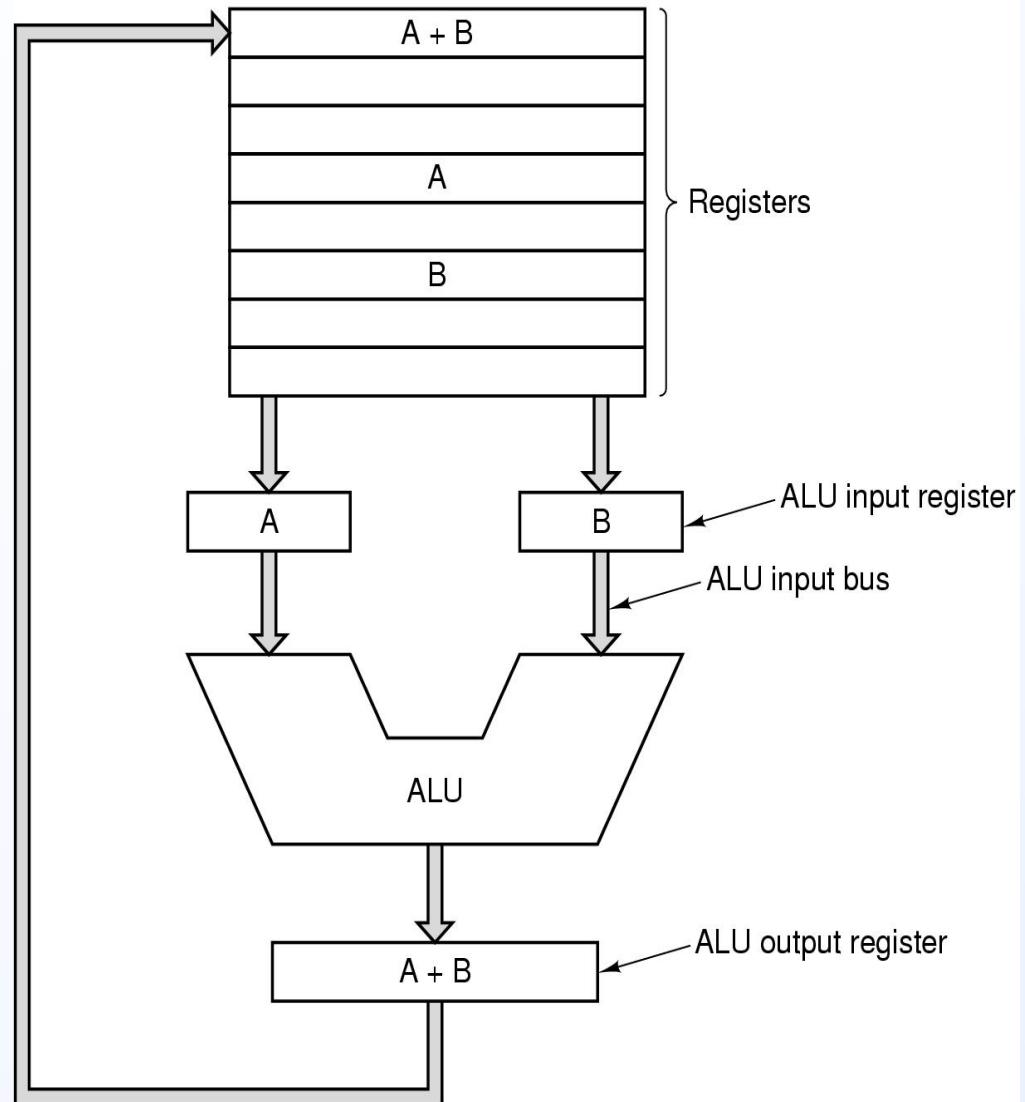
L'insieme di istruzioni IJVM

Hex	Mnemonic	Meaning
0x10	BIPUSH <i>byte</i>	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO <i>offset</i>	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ <i>offset</i>	Pop word from stack and branch if it is zero
0x9B	IFLT <i>offset</i>	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ <i>offset</i>	Pop two words from stack; branch if equal
0x84	IINC <i>varnum const</i>	Add a constant to a local variable
0x15	ILOAD <i>varnum</i>	Push local variable onto stack
0xB6	INVOKEVIRTUAL <i>disp</i>	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE <i>varnum</i>	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W <i>index</i>	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

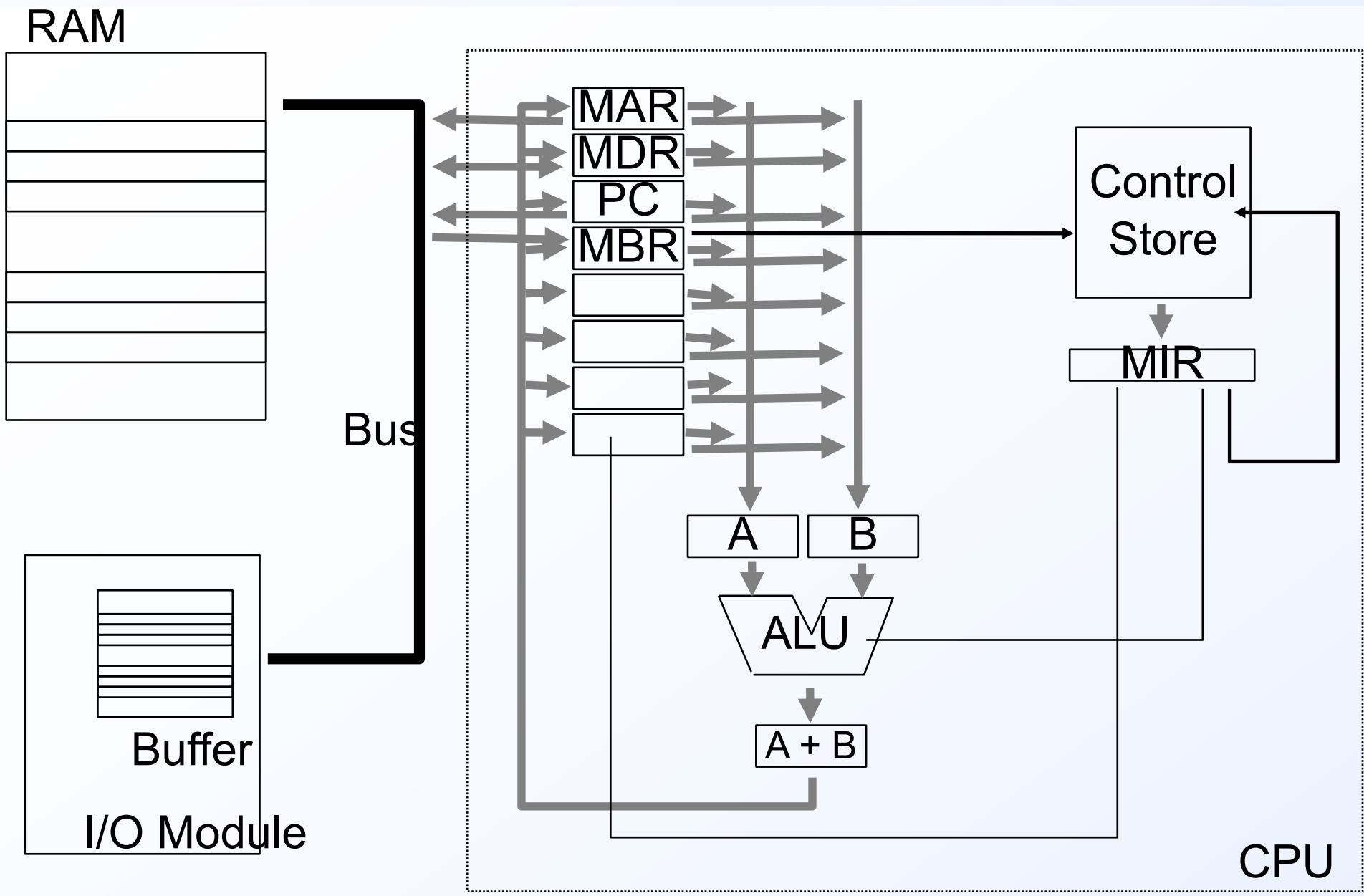
Gli operandi *byte*, *const*, e *varnum* hanno dimensione di 1 byte, *disp*, *index*, e *offset* sono di 2 byte.

Microistruzioni

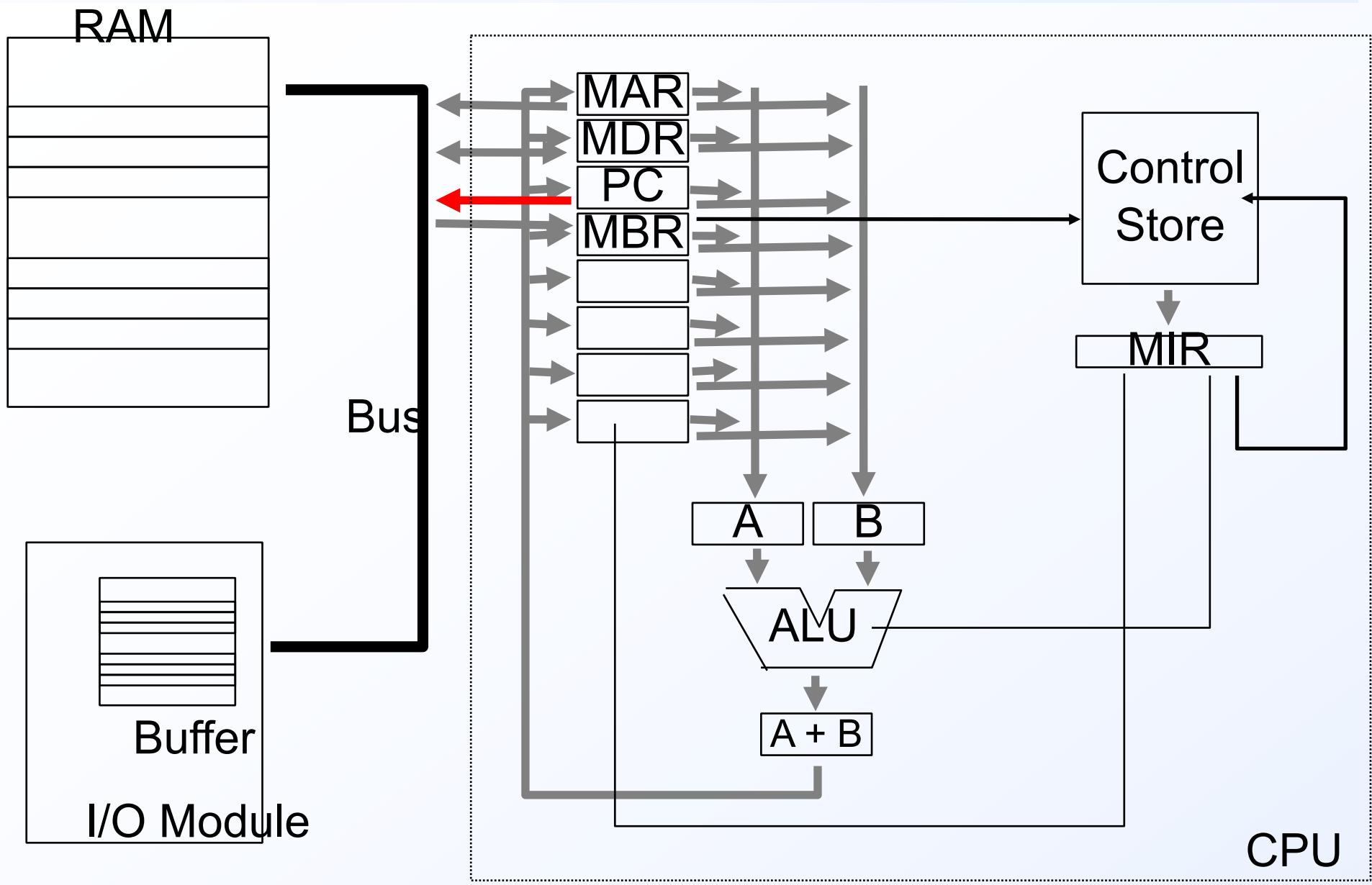
- La IJVM è interpretato dal microinterprete che lo traduce in una sequenza di microistruzioni
- le microistruzioni controllano un **ciclo di data-path**
- Ciclo del data path:** processo di far passare due operandi attraverso l'ALU e memorizzarne il risultato



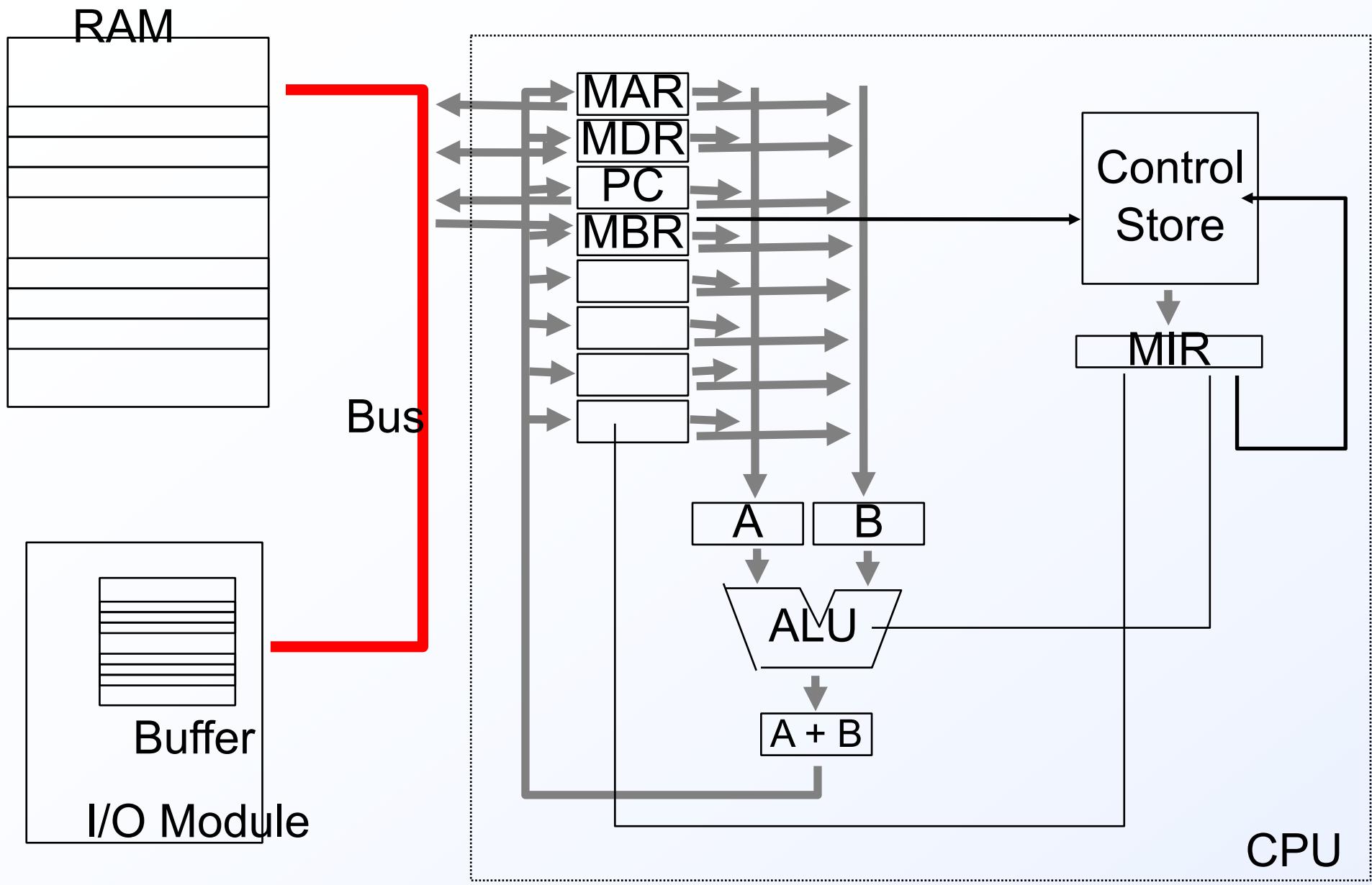
Il ciclo di data path



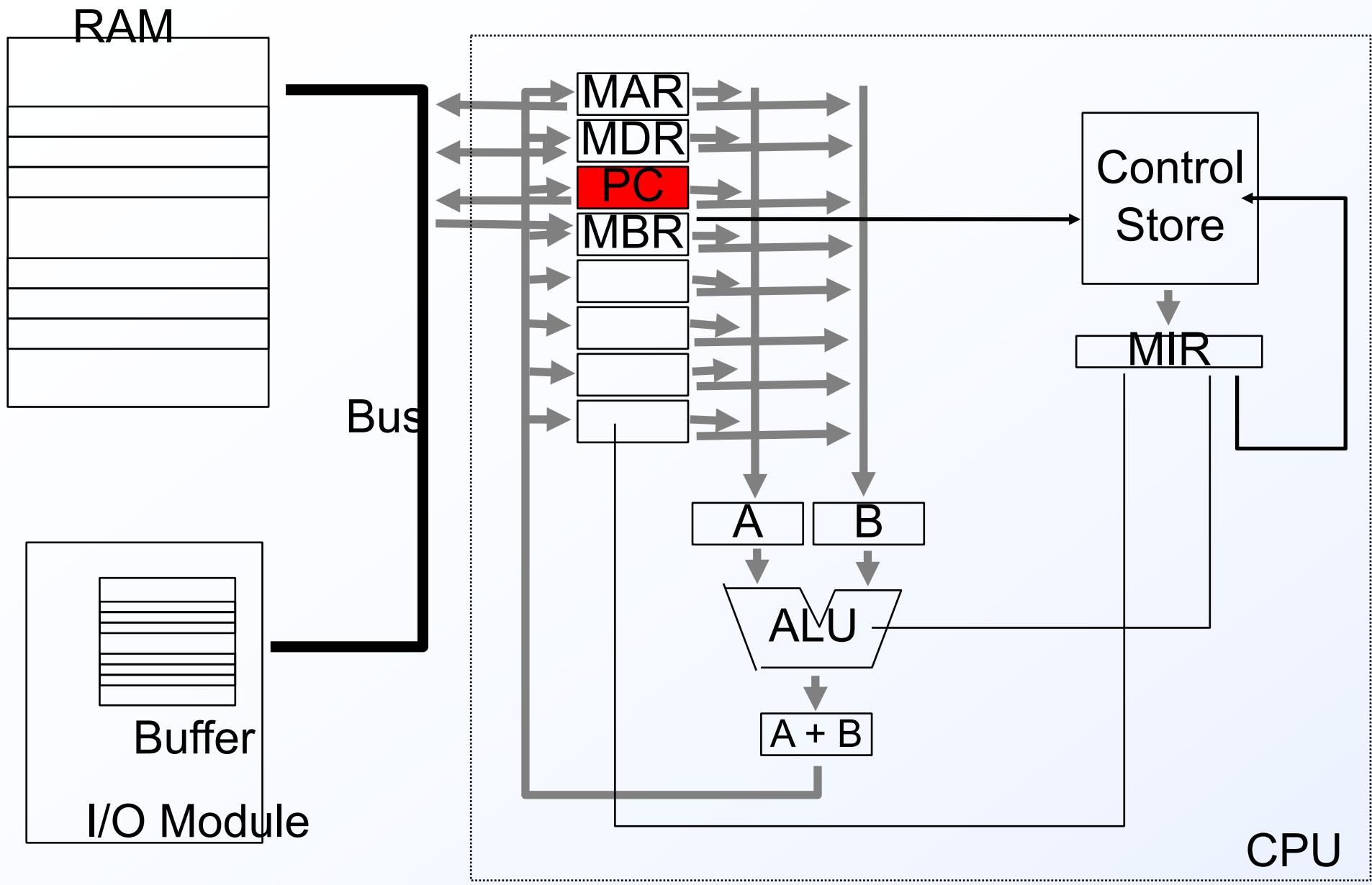
Il ciclo di data path



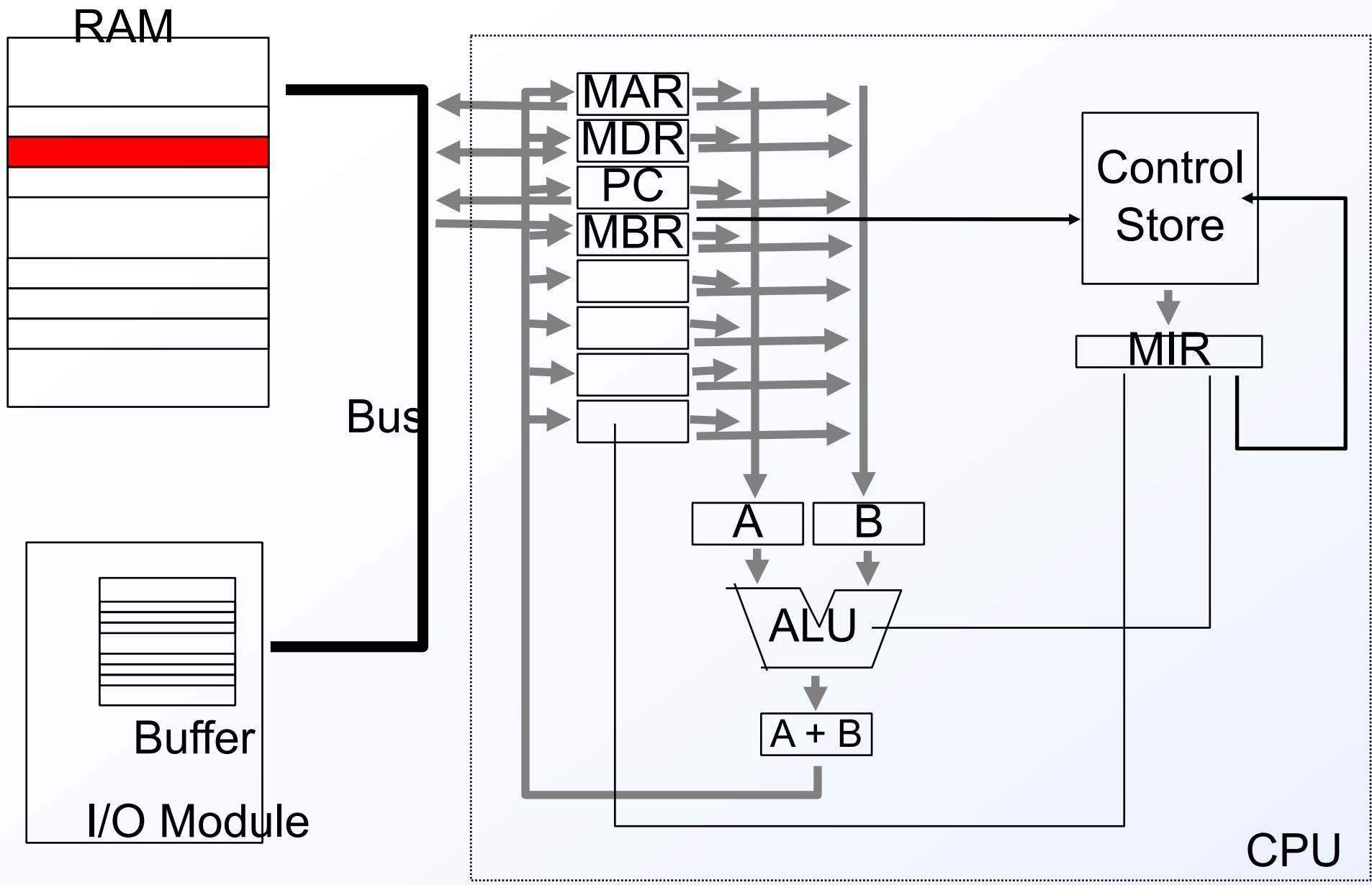
Il ciclo di data path



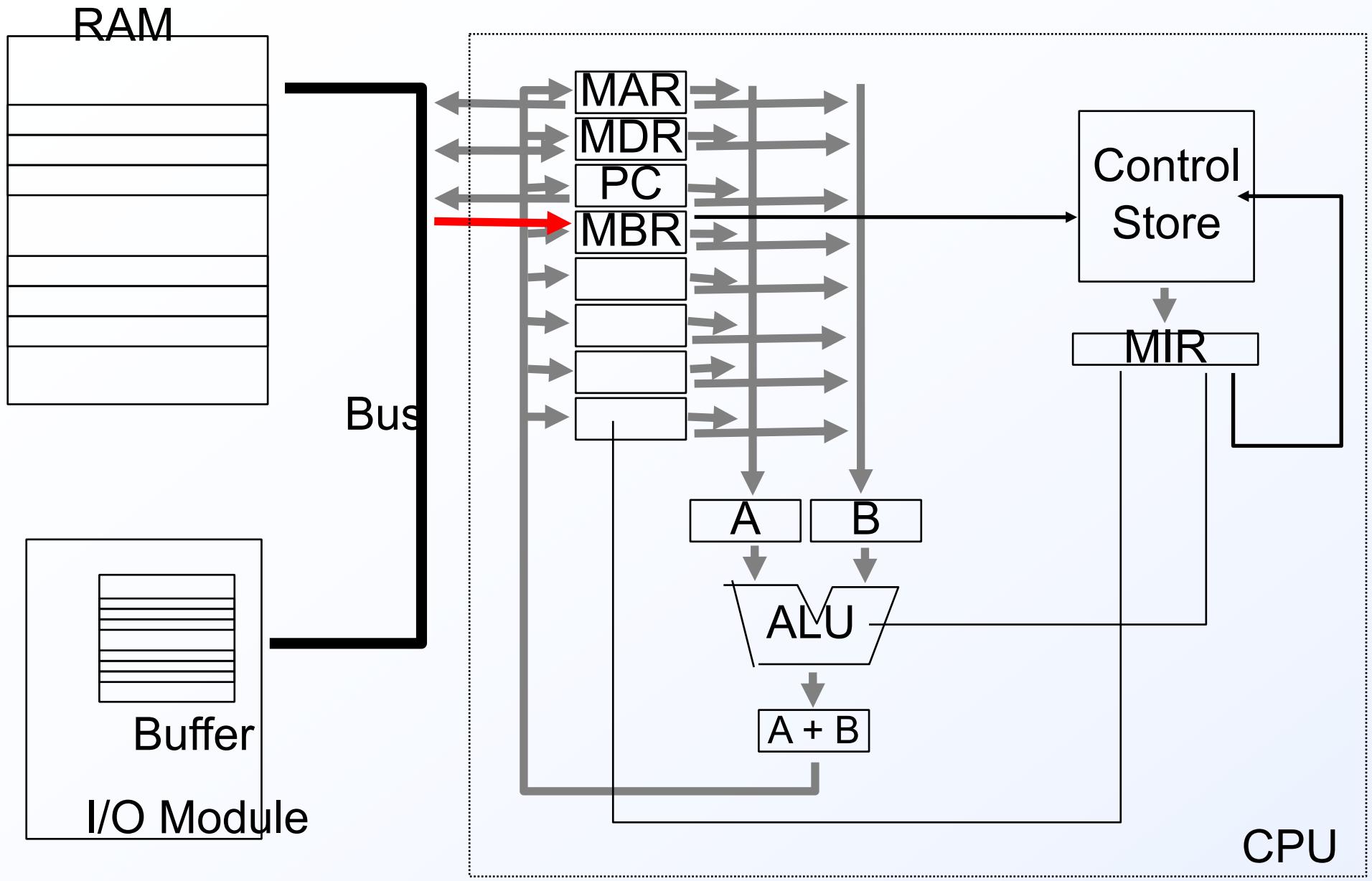
Il ciclo di data path



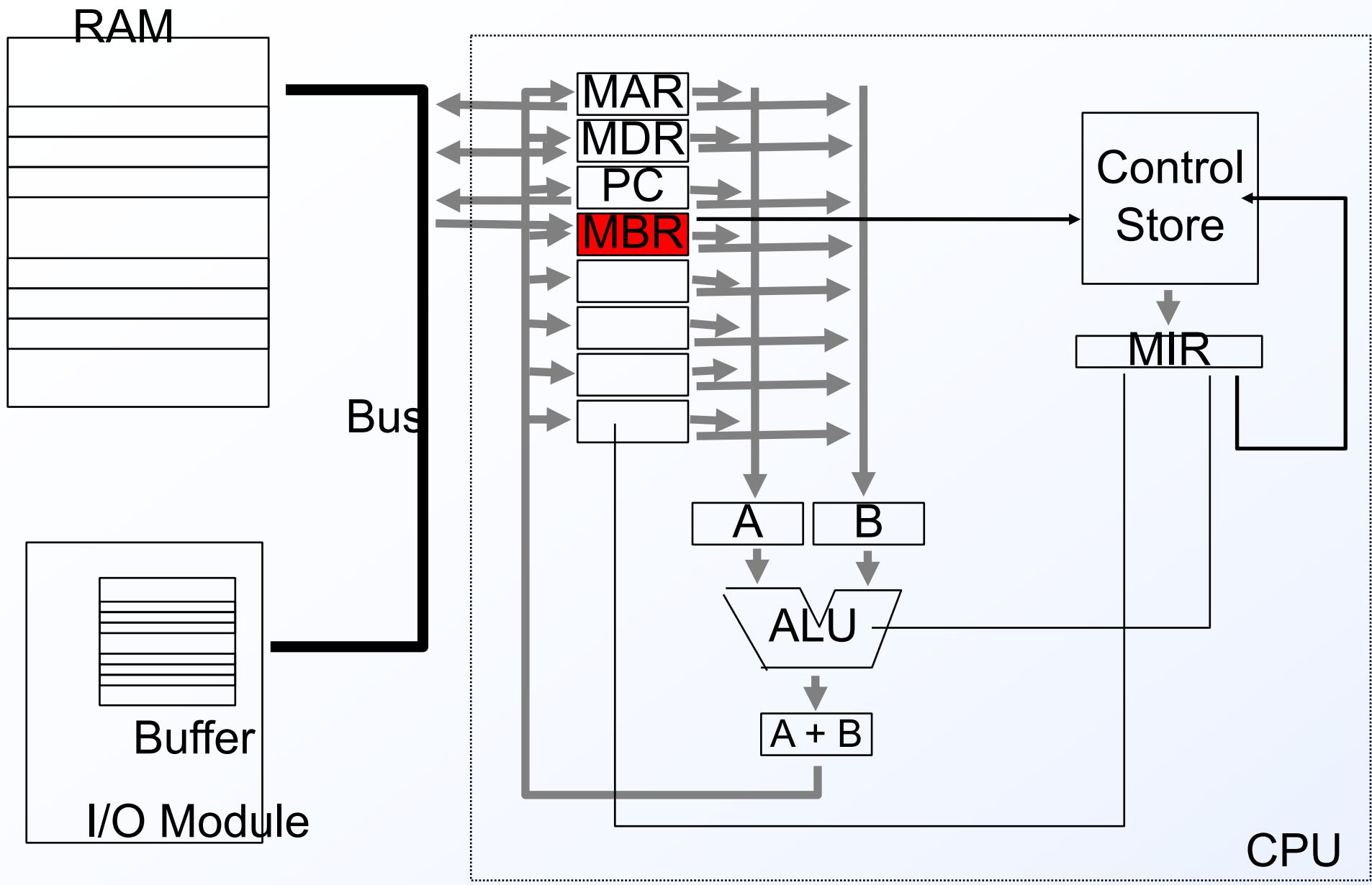
Il ciclo di data path



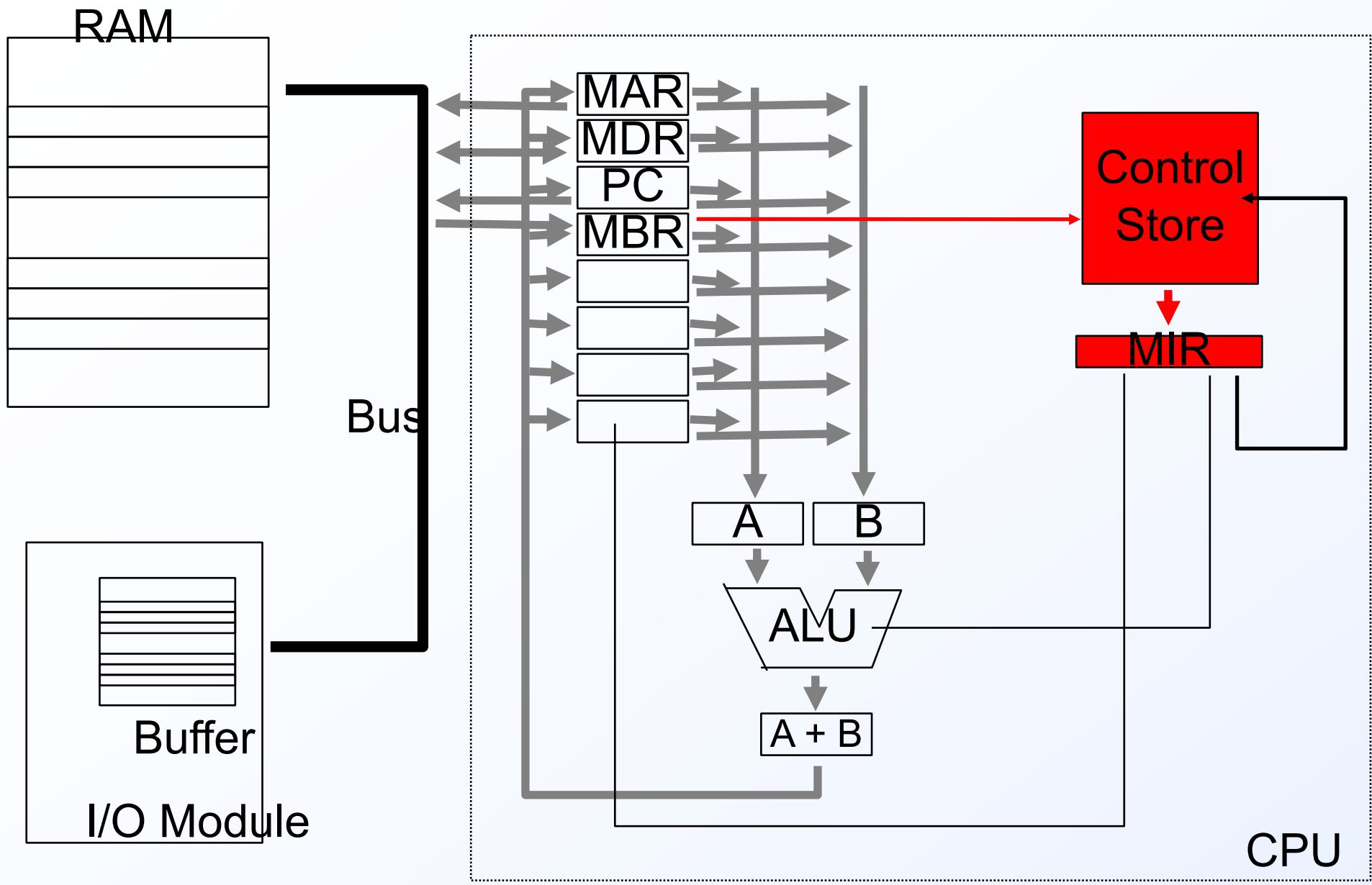
Il ciclo di data path



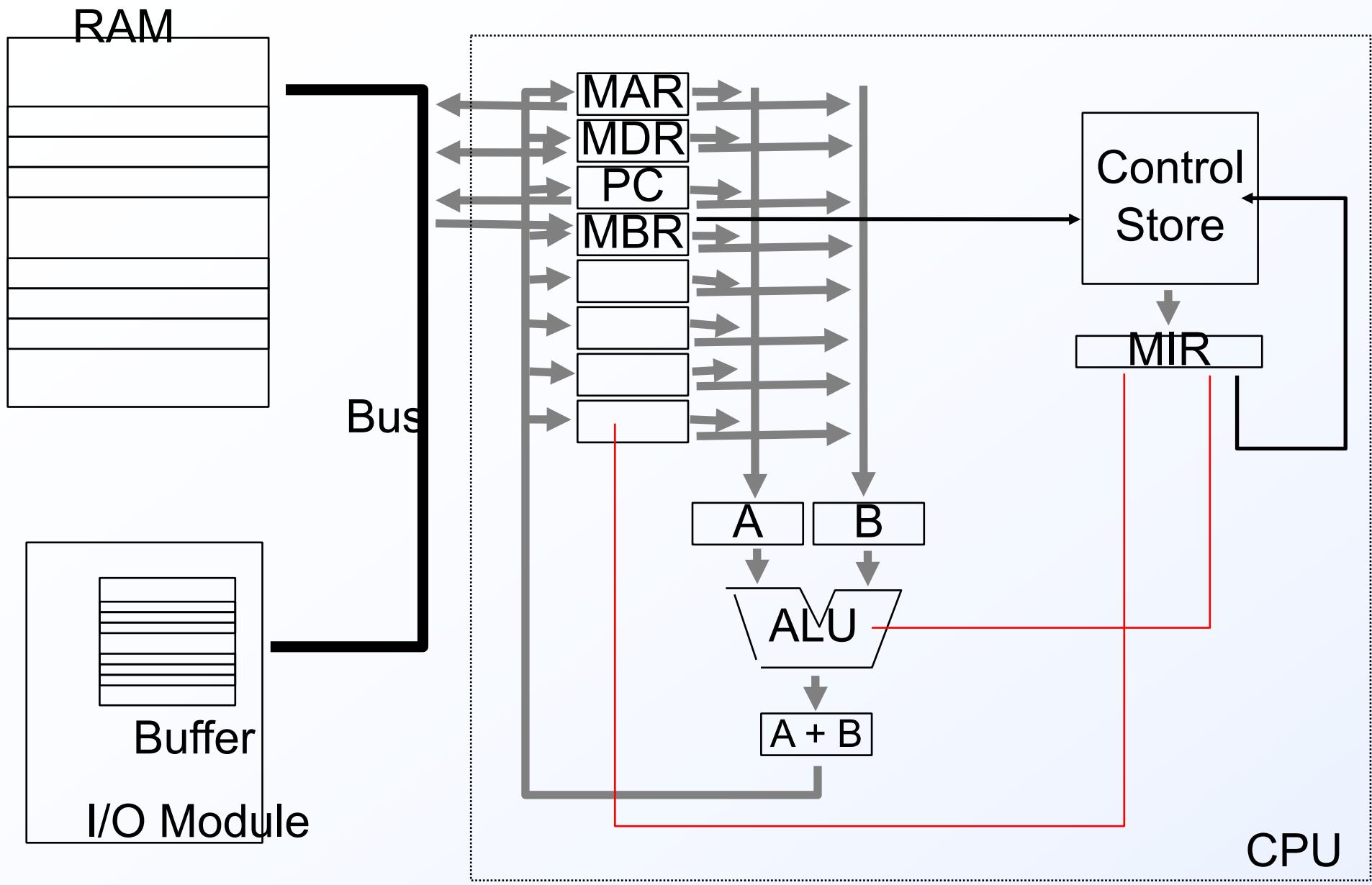
Il ciclo di data path



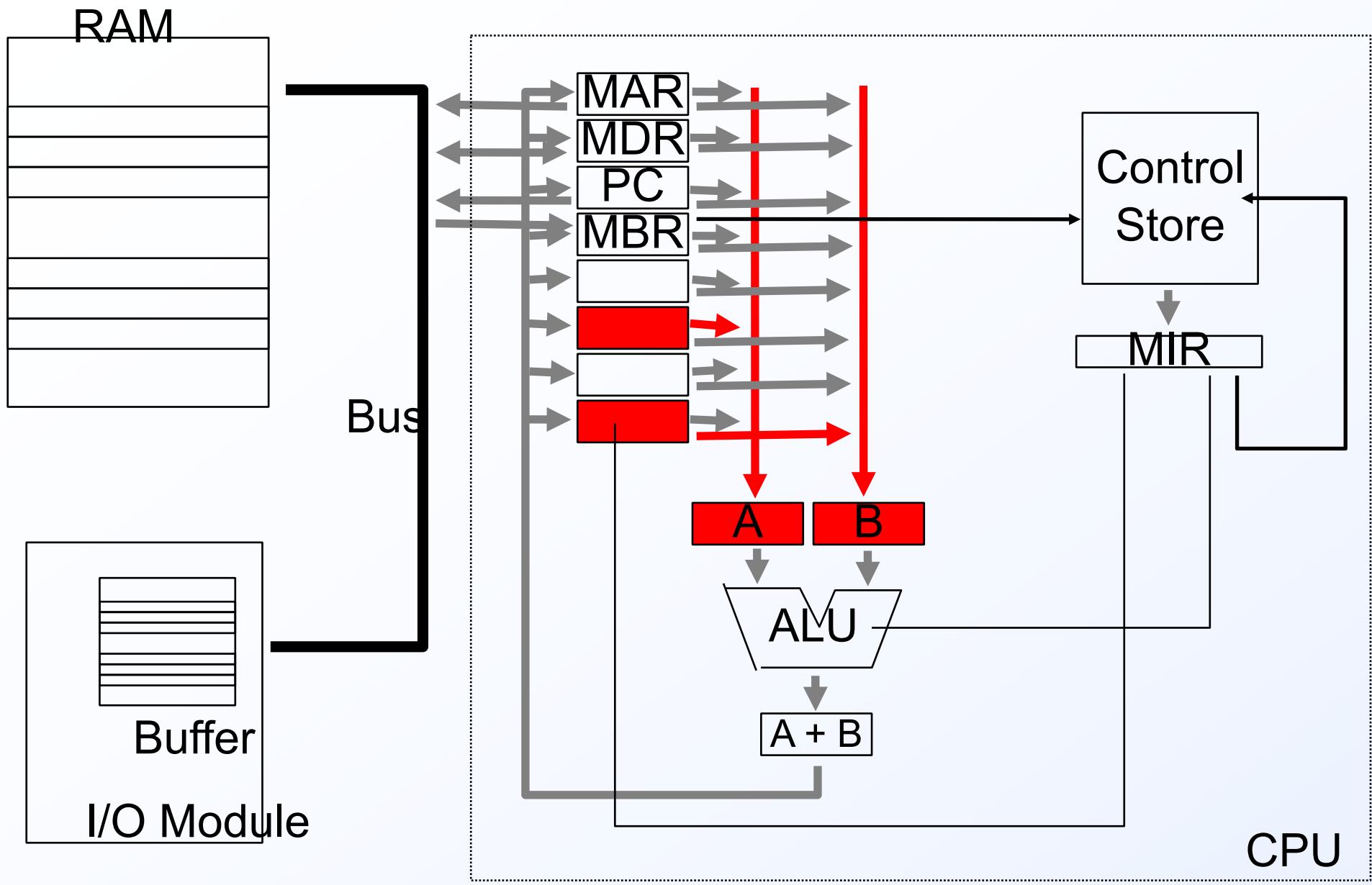
Il ciclo di data path



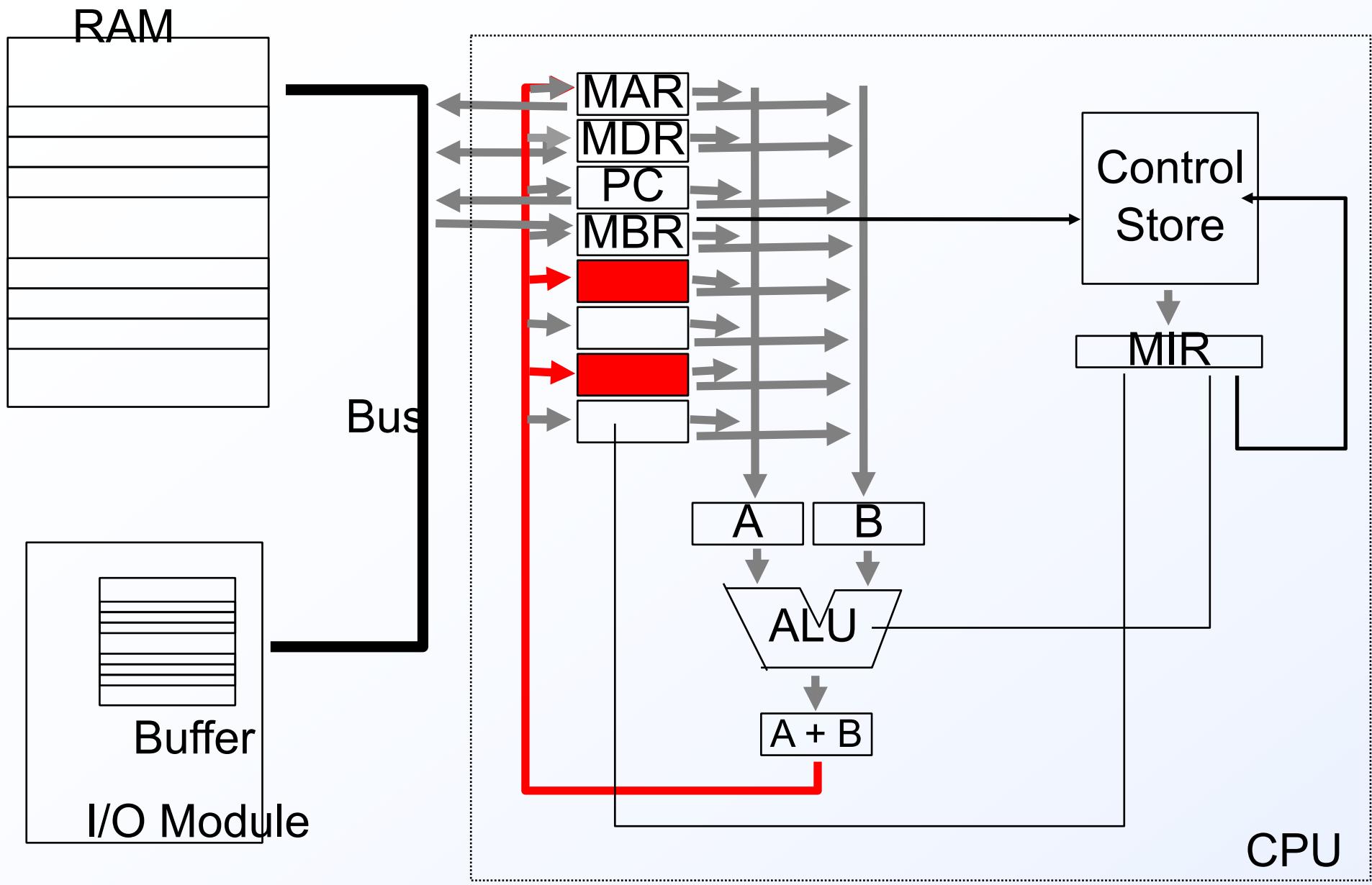
Il ciclo di data path



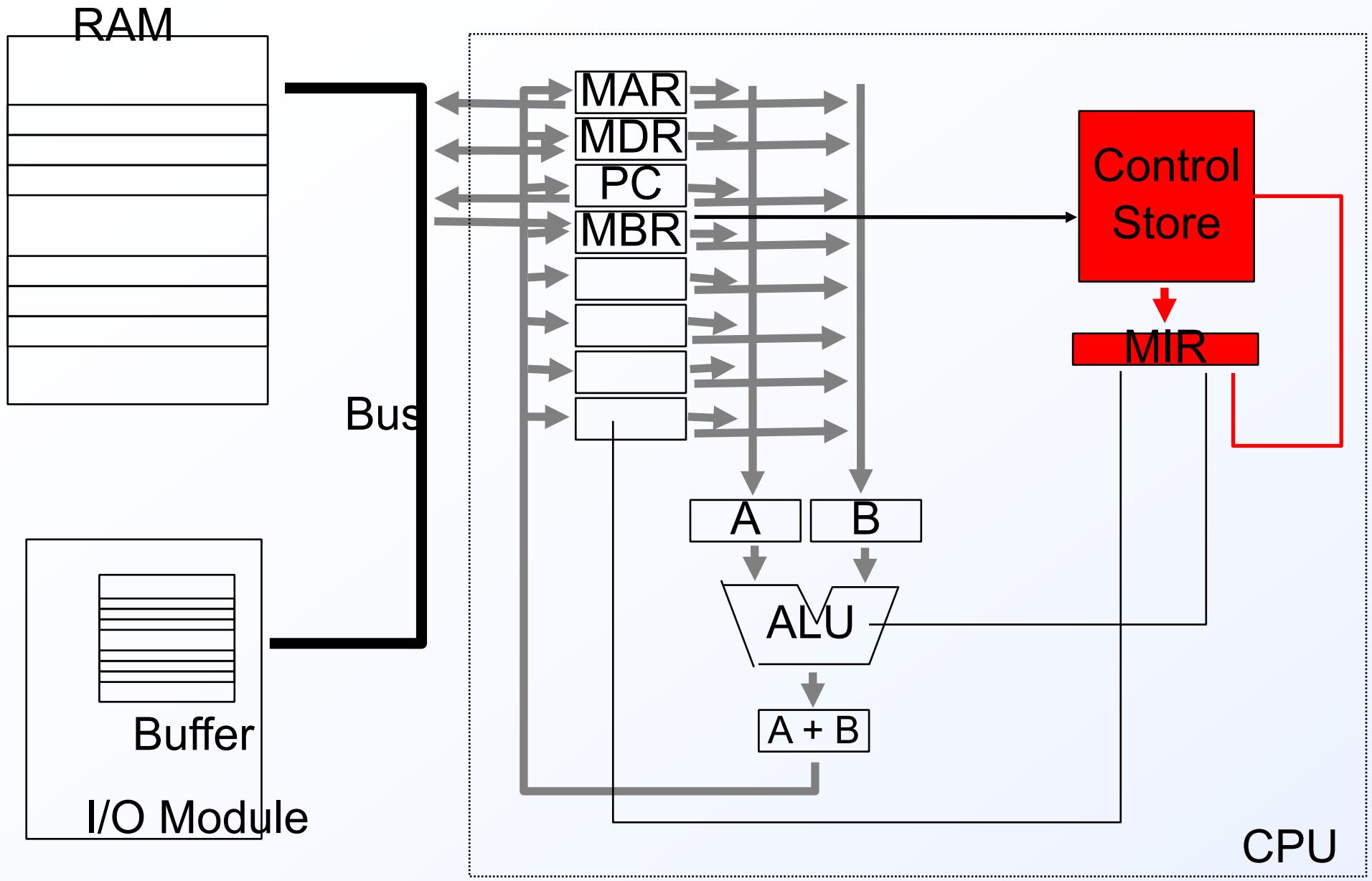
Il ciclo di data path



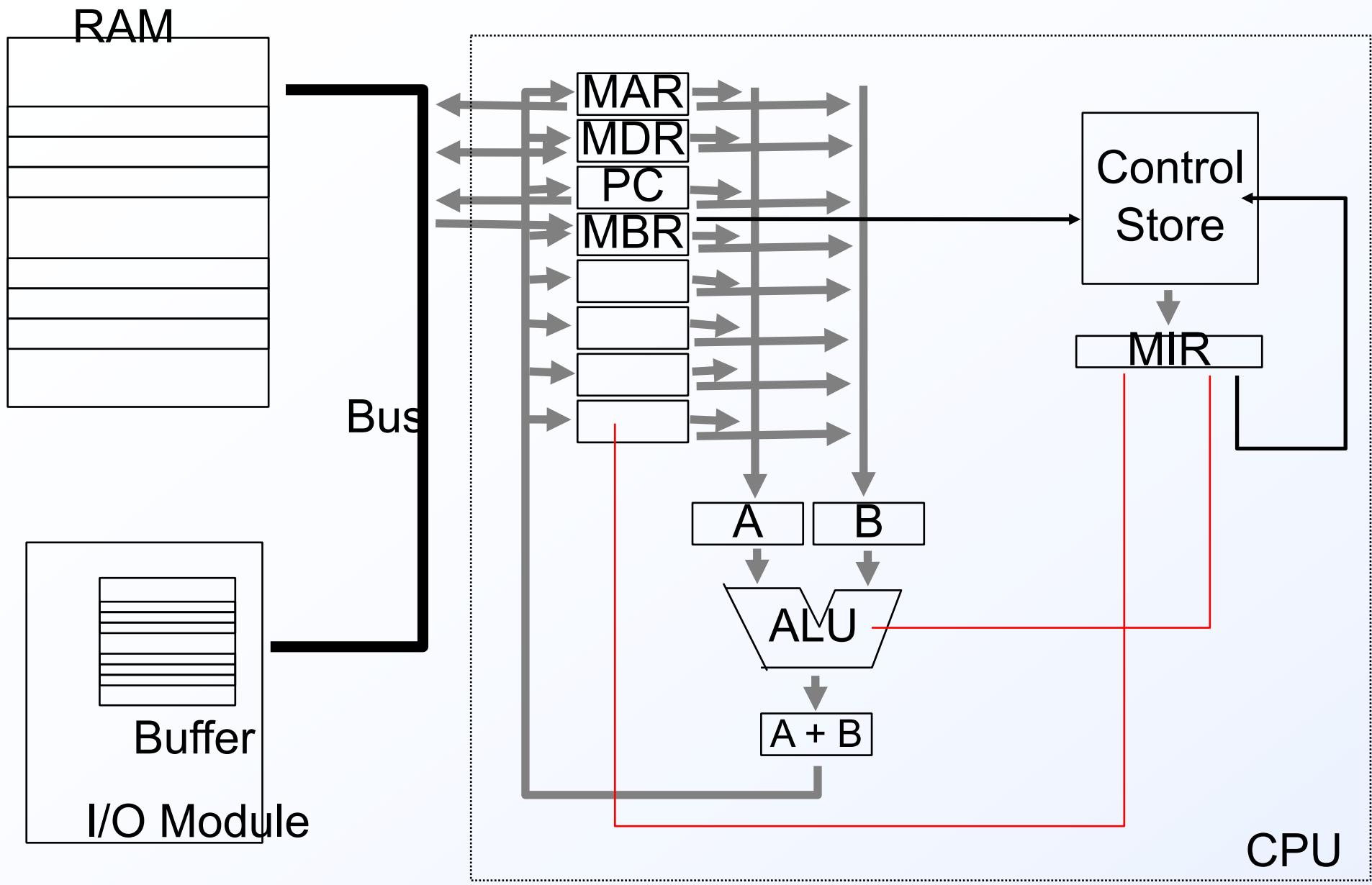
Il ciclo di data path



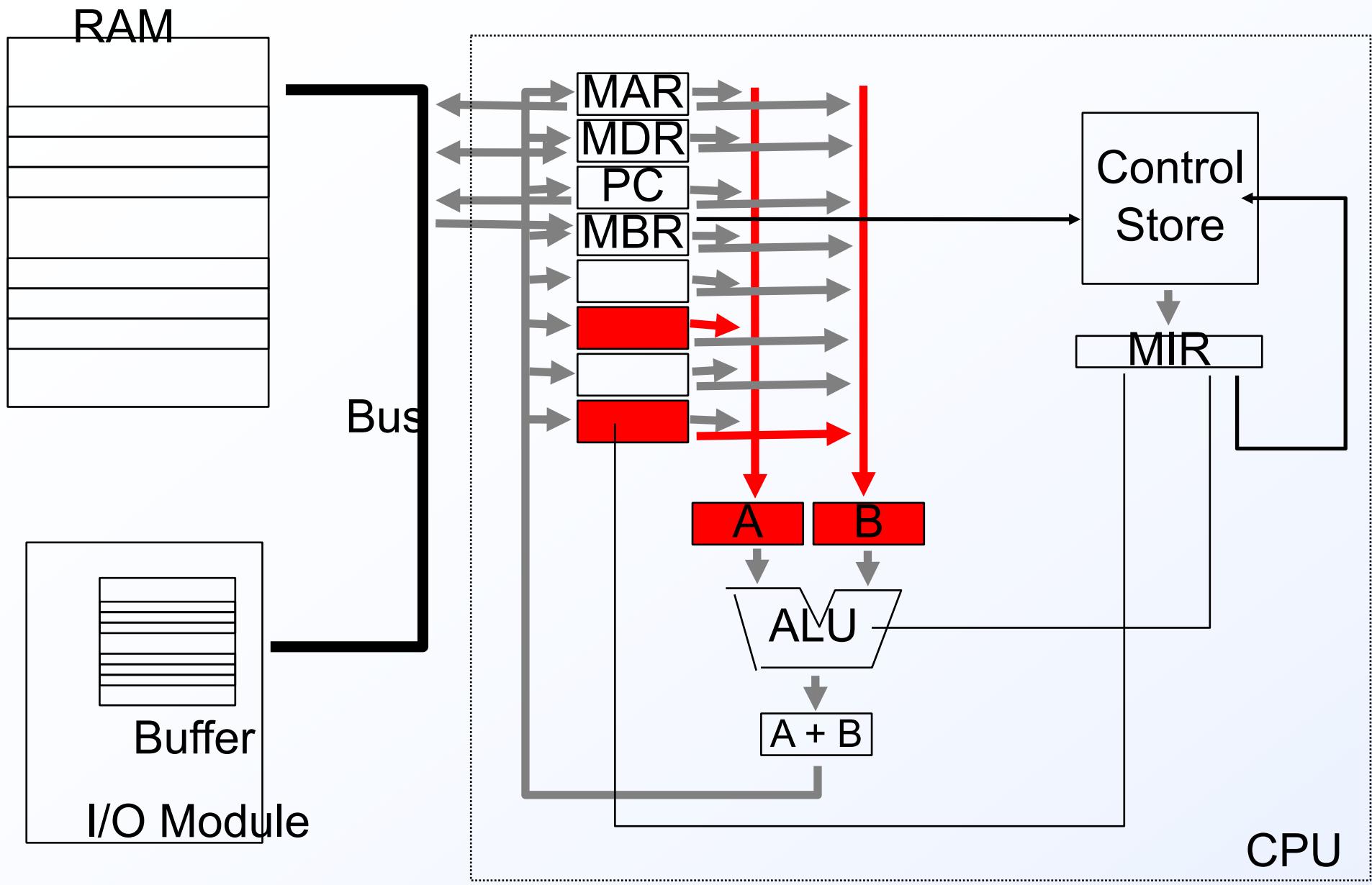
Il ciclo di data path



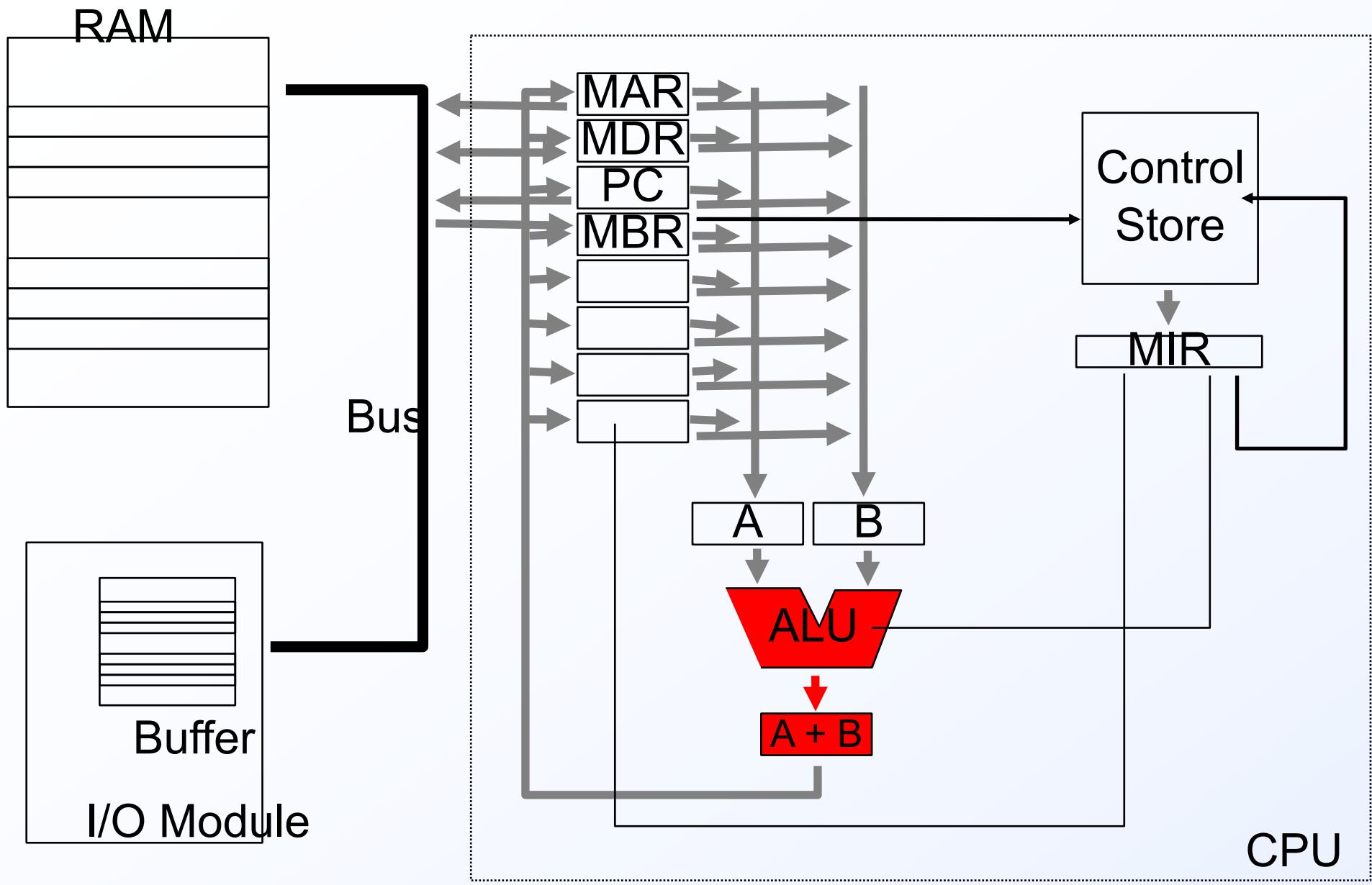
Il ciclo di data path



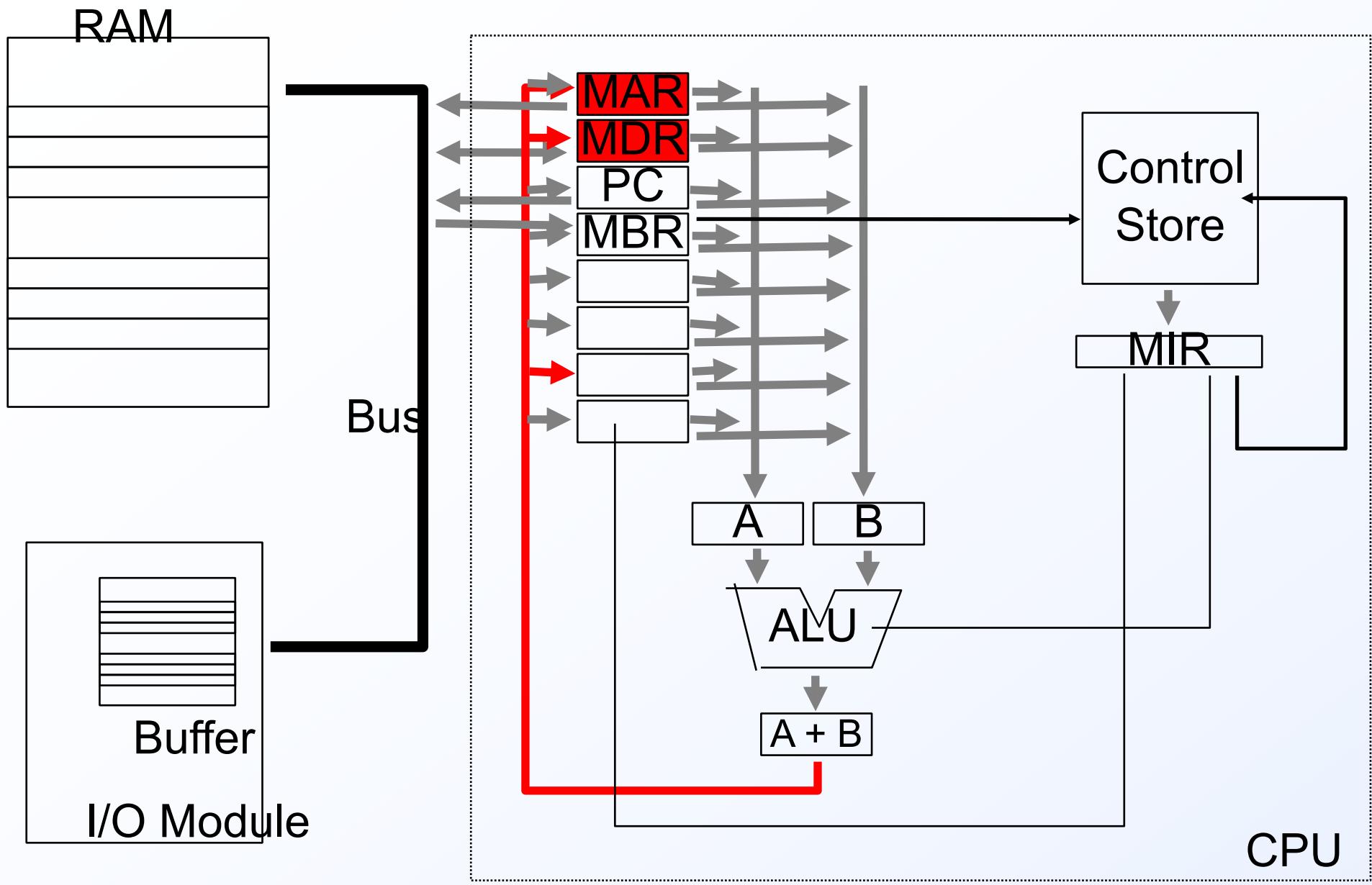
Il ciclo di data path



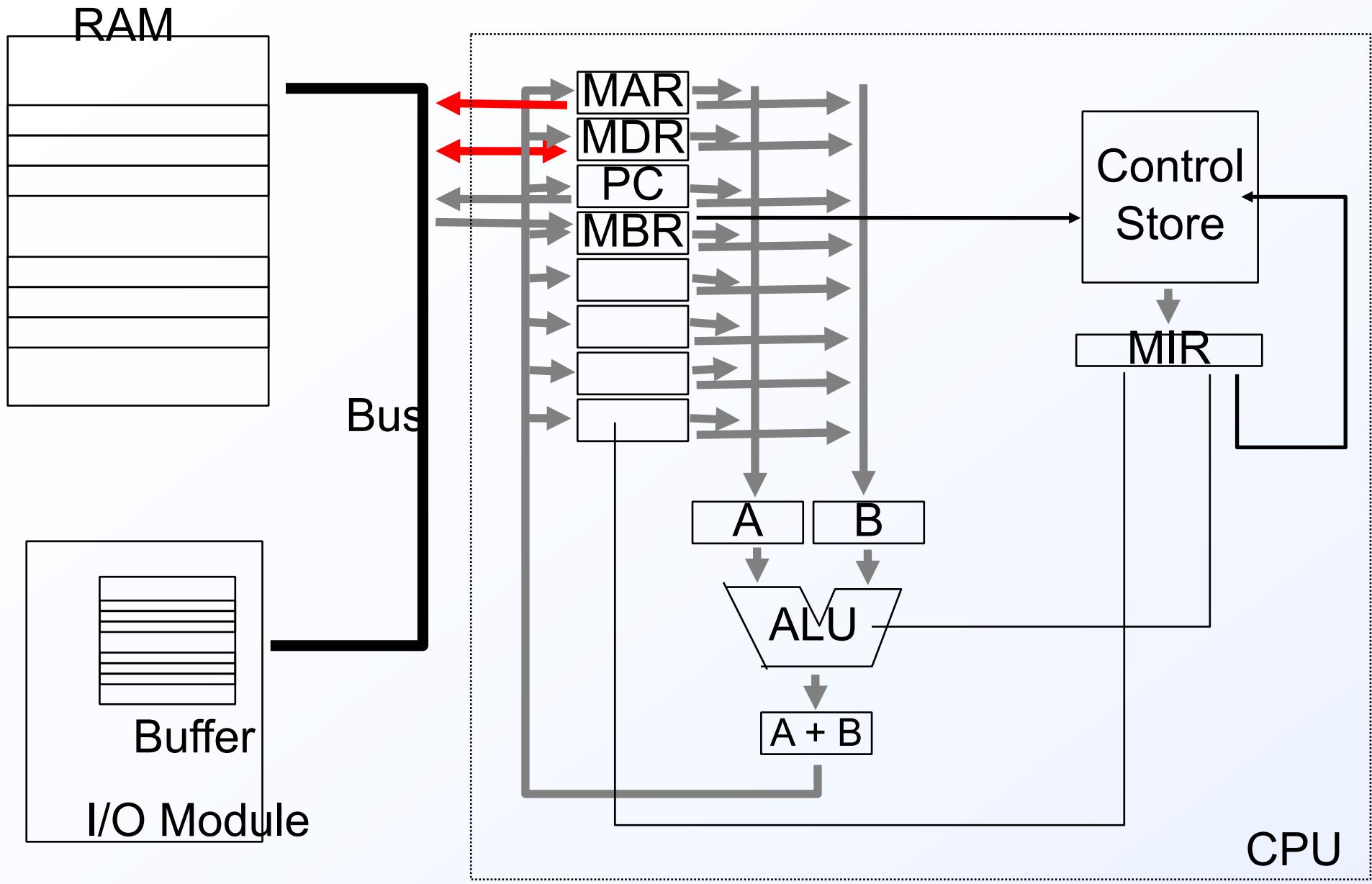
Il ciclo di data path



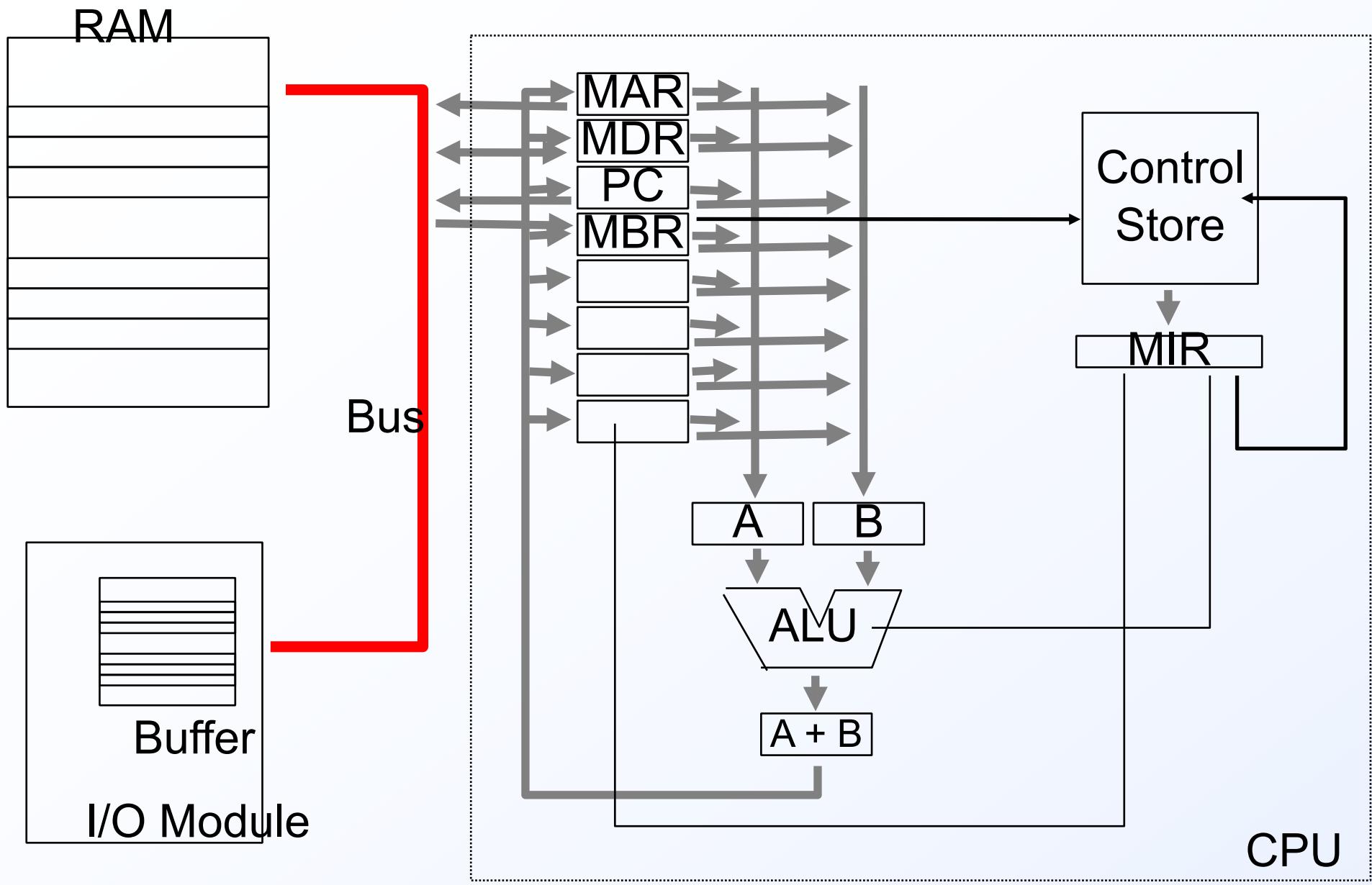
Il ciclo di data path



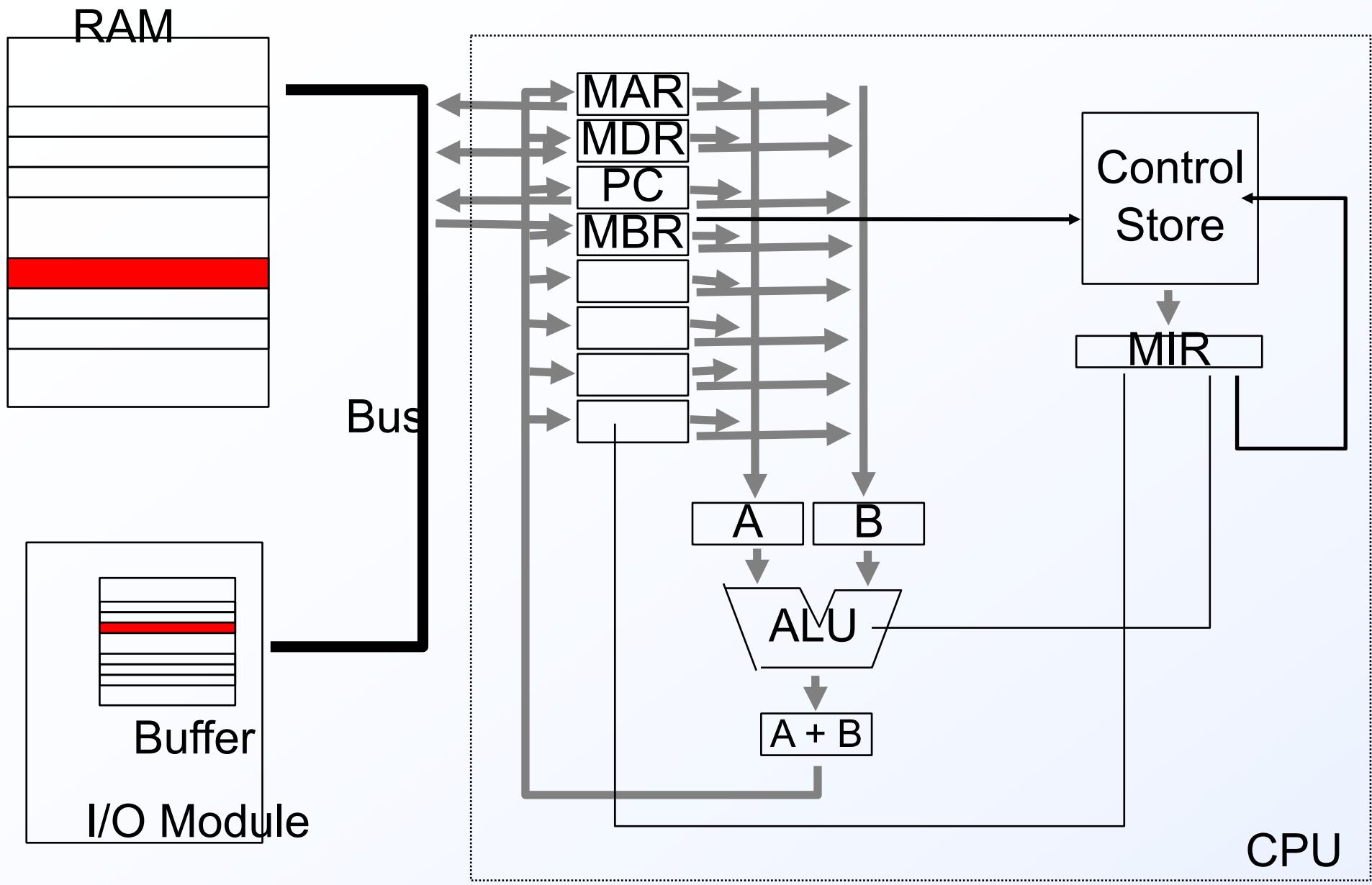
Il ciclo di data path



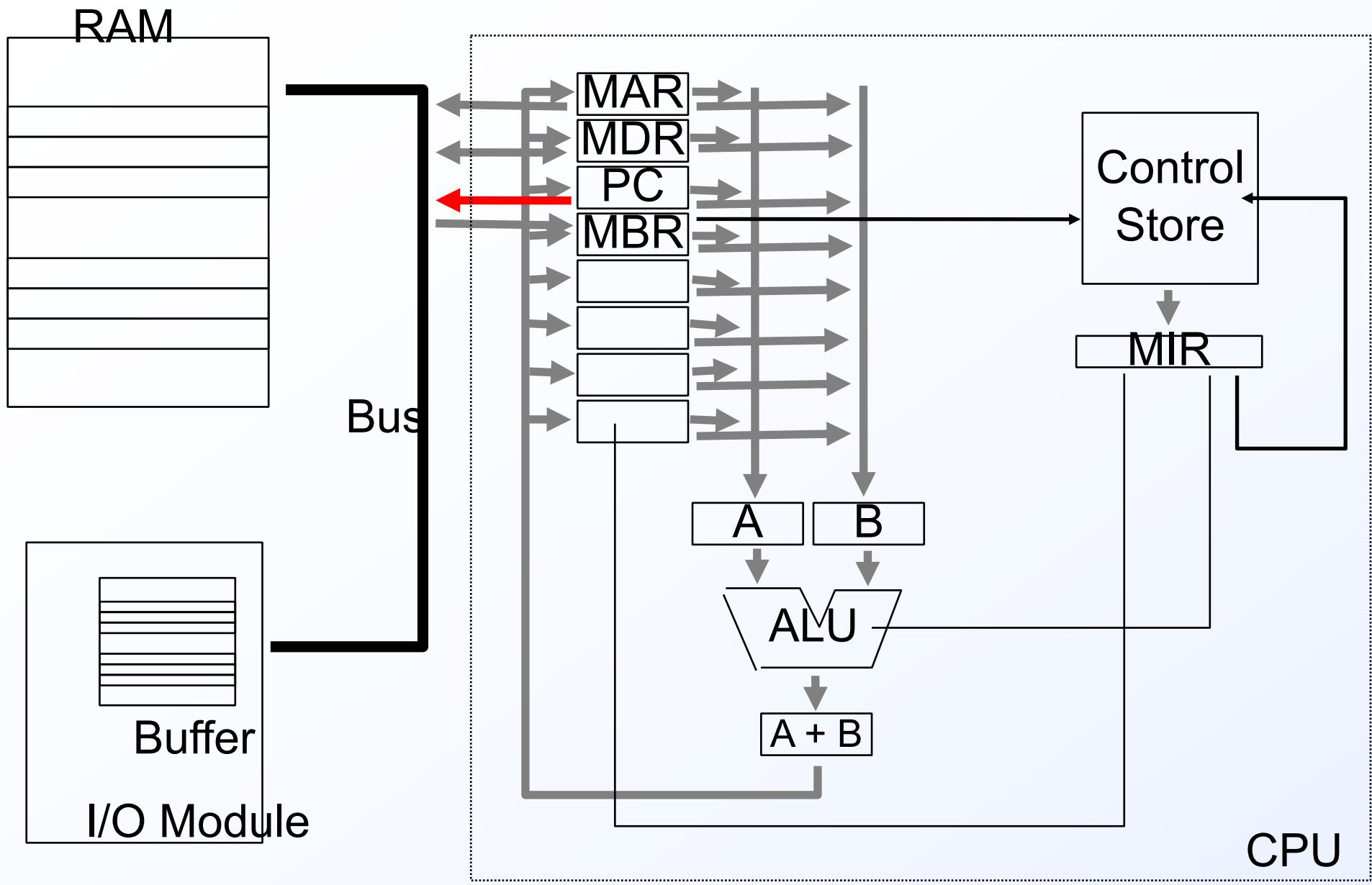
Il ciclo di data path



Il ciclo di data path

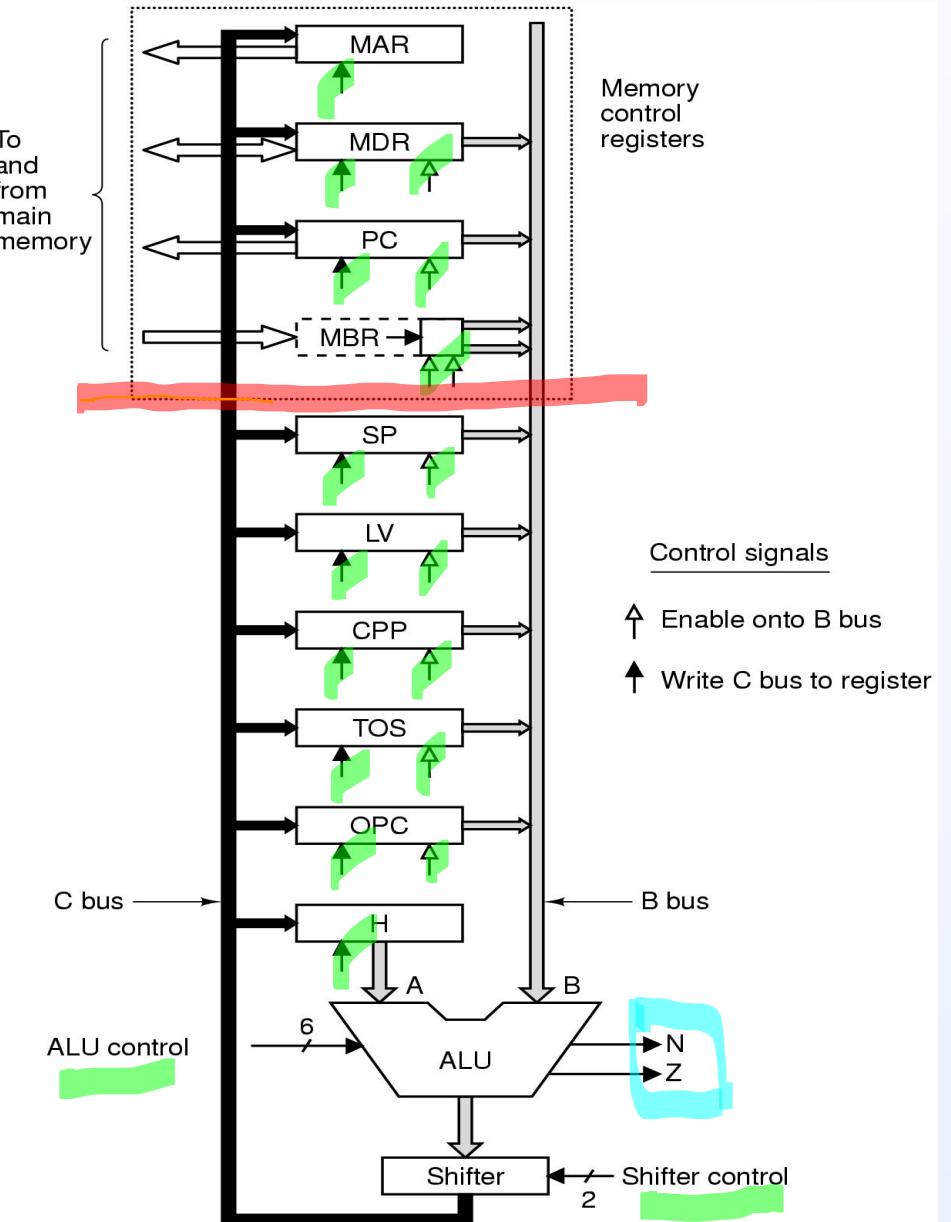


Il ciclo di data path

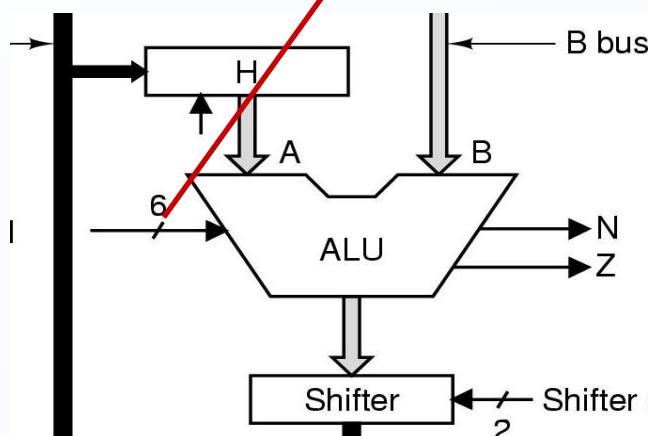


Il data path del nostro esempio

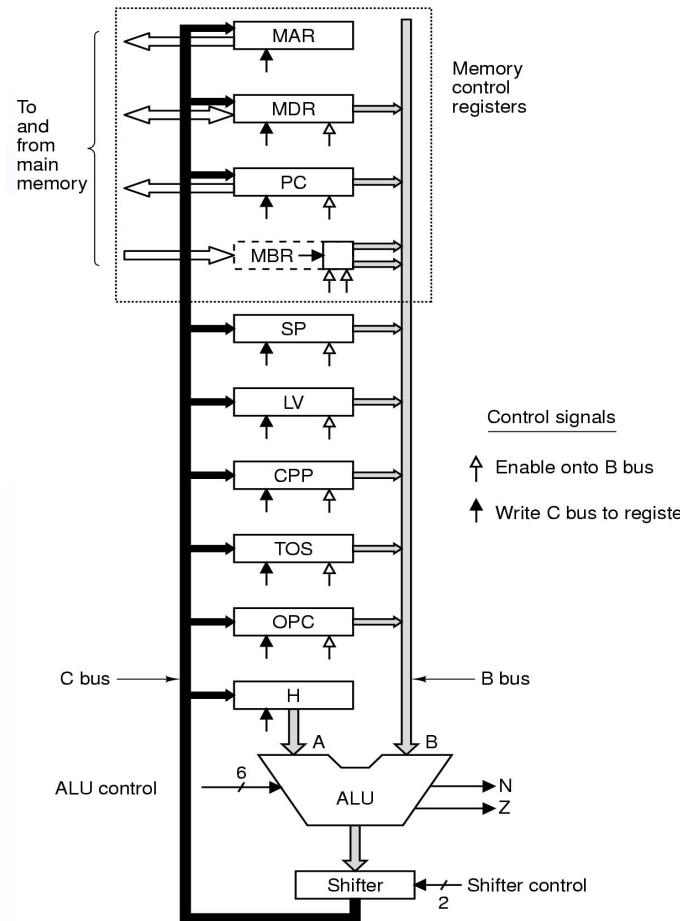
- Registri a **32 bit** con nome simbolico
- ALU con **6** bit di controllo
- Bus C per i dati in output verso i registri dallo shifter
- Bus B per i dati verso l'ALU
- Registro speciale H per l'input di sinistra dell'ALU
- Stesso registro sia per l'input che per l'output dell'ALU
- Shifter con **2** bit di controllo



Bit di controllo dell'ALU

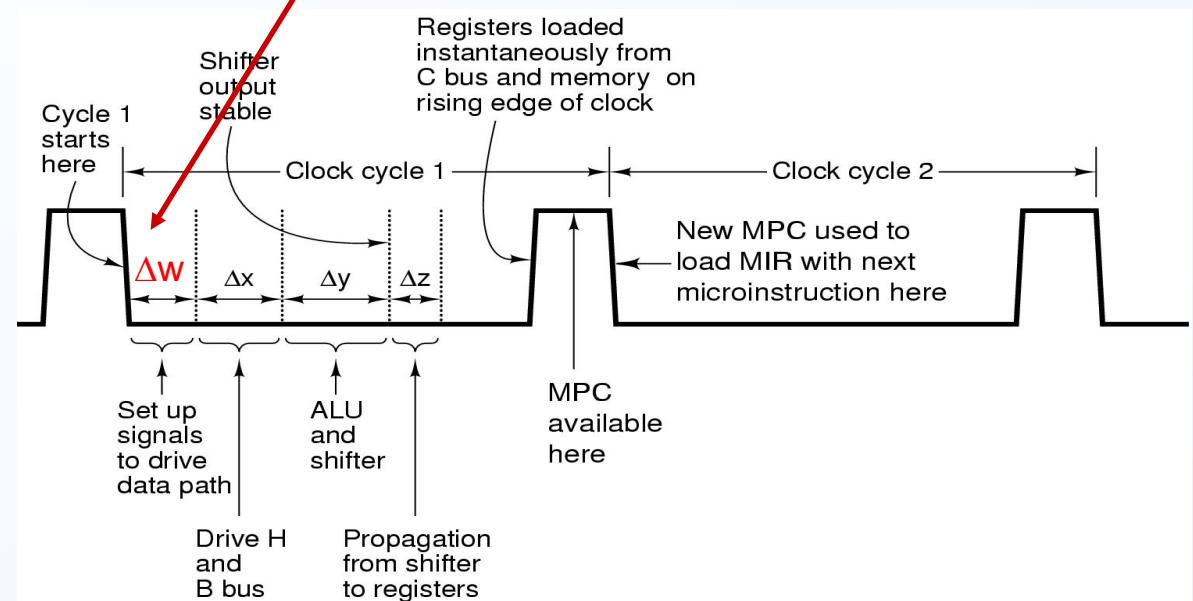


F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	\bar{B}
0	1	1	0	1	0	\bar{A}
1	0	1	1	0	0	\bar{B}
1	1	1	1	0	0	$A + B$
1	1	1	1	0	1	$A + B + 1$
1	1	1	0	0	1	$A + 1$
1	1	0	1	0	1	$B + 1$
1	1	1	1	1	1	$B - A$
1	1	0	1	1	0	$B - 1$
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	-1

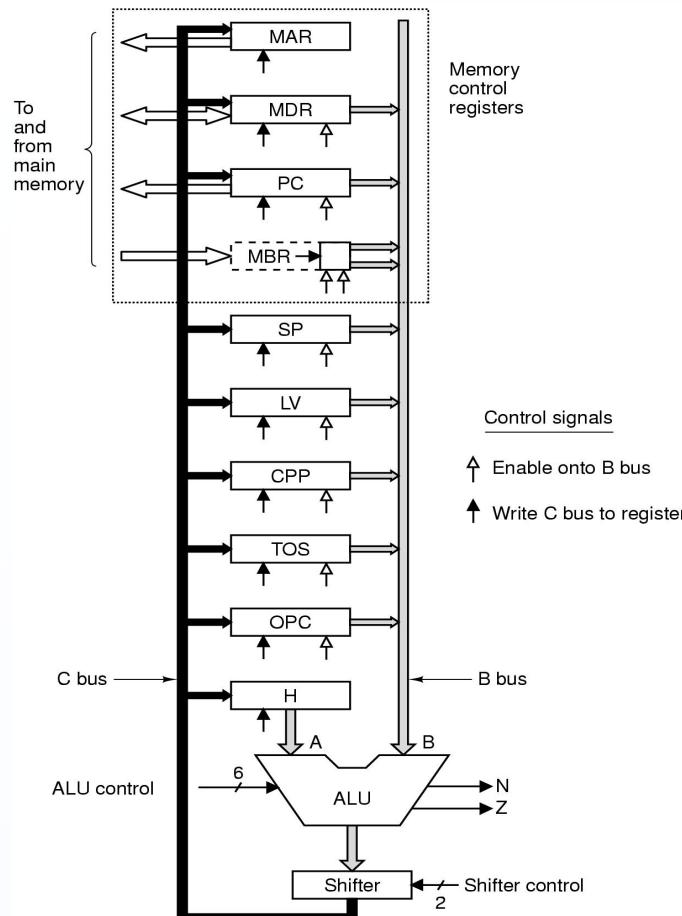


Visualizzazione del data path

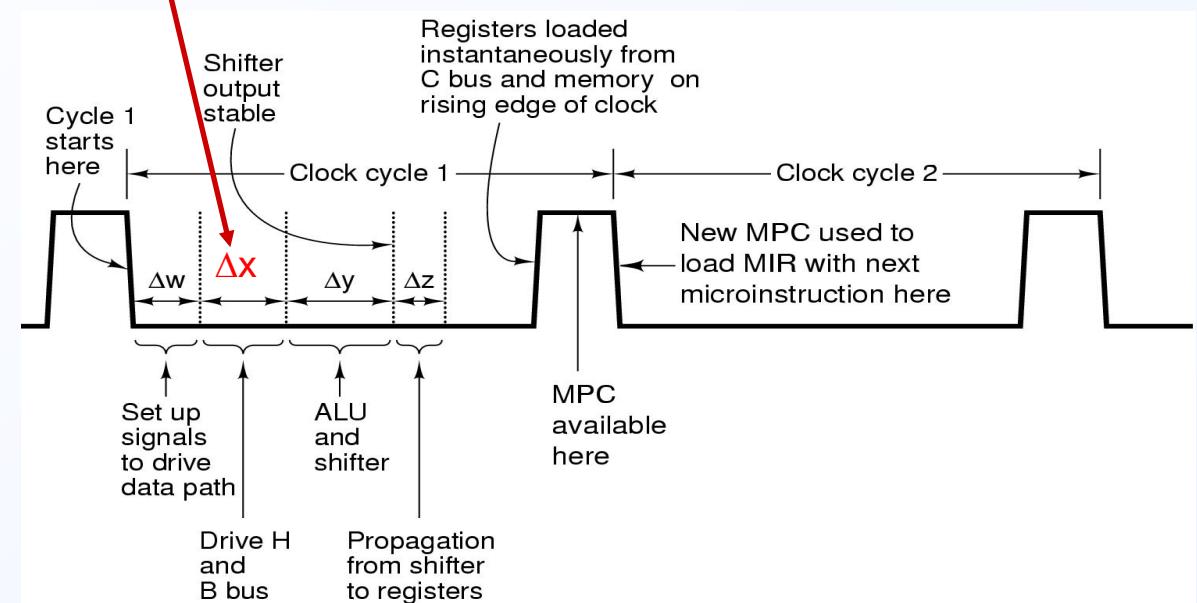
1) I segnali di controllo si stabilizzano



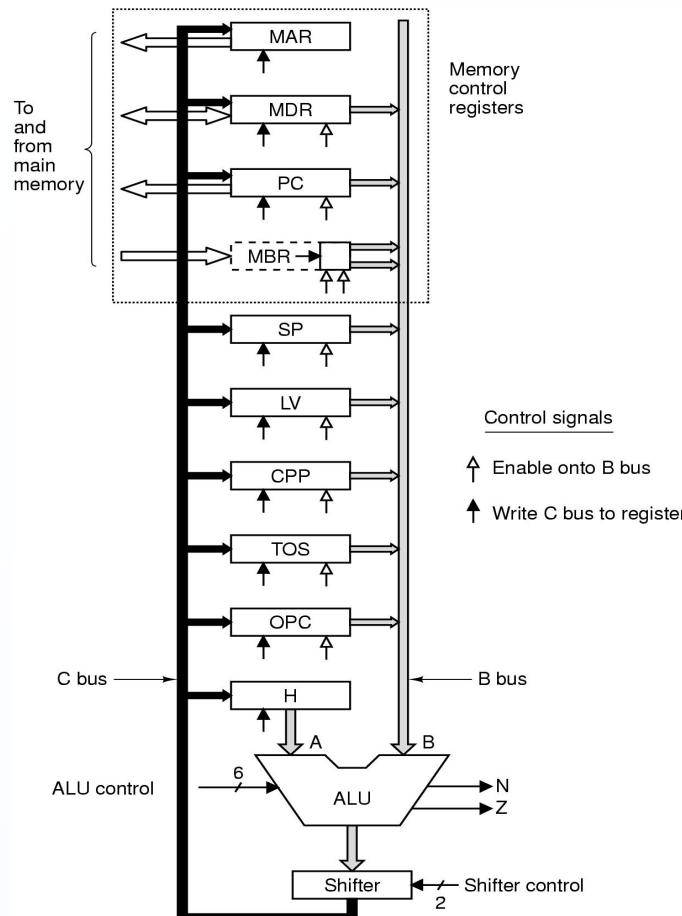
Sincronizzazione del data path



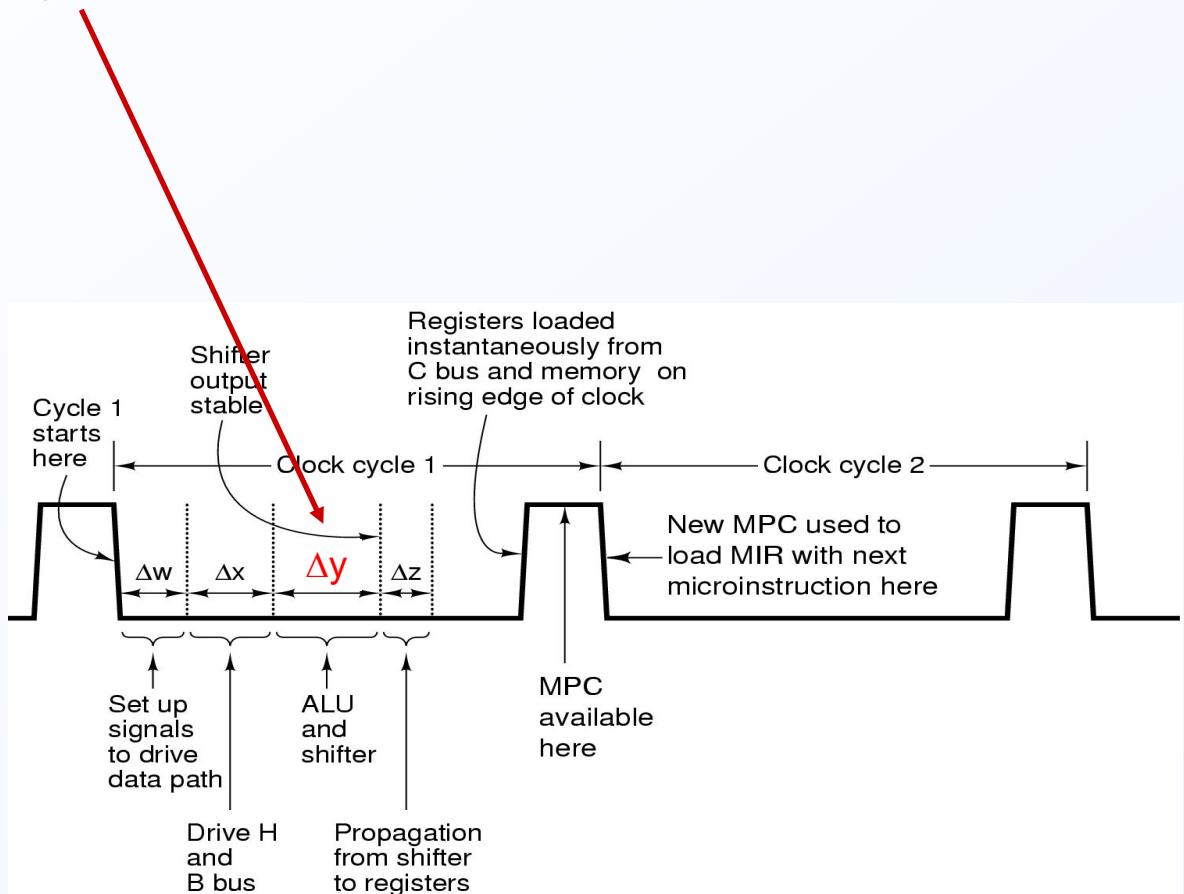
- 1) I segnali di controllo si stabilizzano
- 2) Uno dei registri viene scritto sul bus B



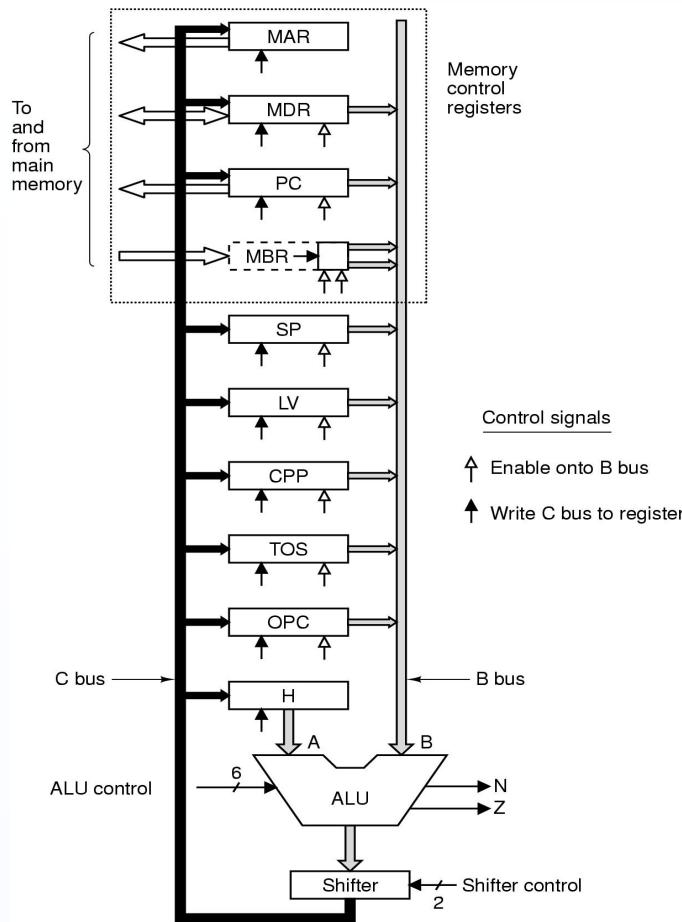
Sincronizzazione del data path



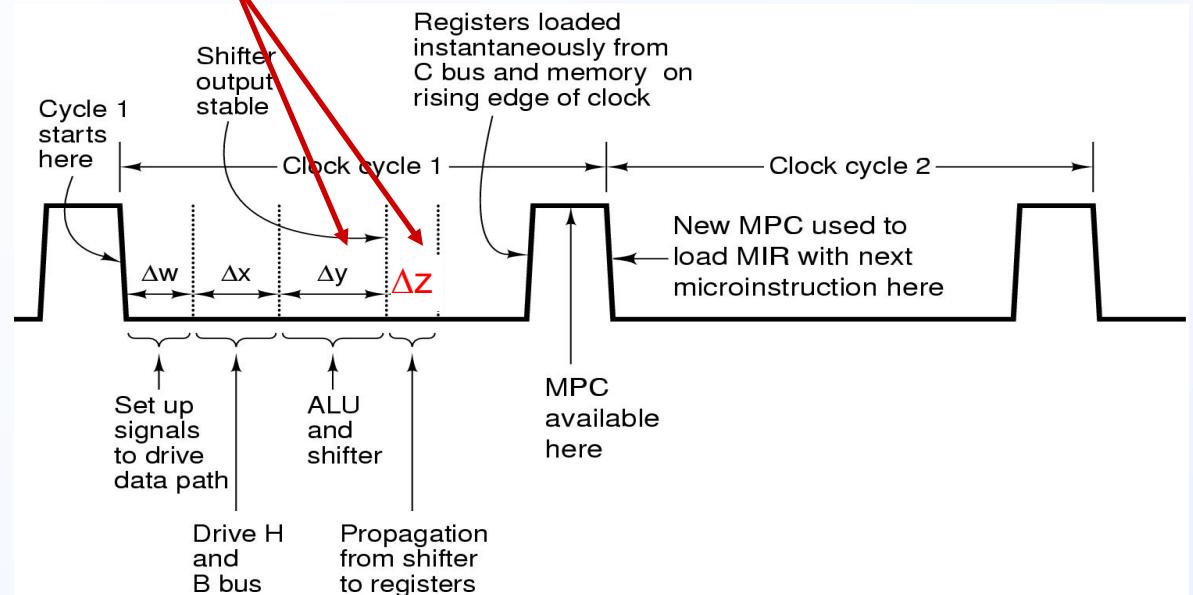
- 1) I segnali di controllo si stabilizzano
- 2) I registri vengono scritti sul bus B
- 3) L'ALU e lo shifter calcolano



Sincronizzazione del data path

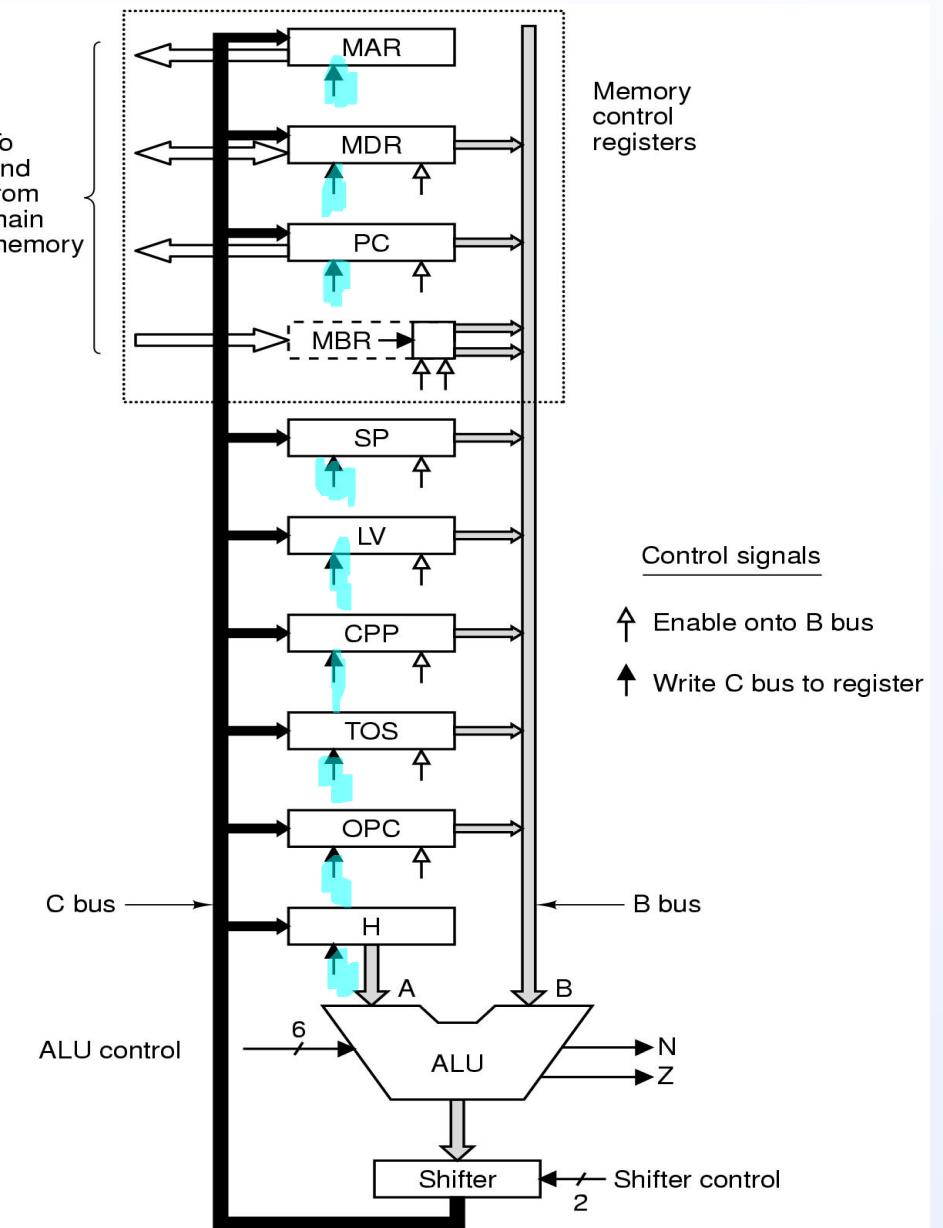


- 1) I segnali di controllo si stabilizzano
- 2) I registri vengono scritti sul bus B
- 3) L'ALU e lo shifter calcolano
- 4) I risultati si propagano lungo il bus C e ritornano ai registri



Data Path del nostro esempio

- È possibile attivare un solo registro per volta come output verso il bus B (sono sufficienti **4** bit di controllo, un decoder 4-16)
- È possibile (e a volte utile) memorizzare l'output dello shifter anche su più di un registro contemporaneamente (sono necessari **9** bit di abilitazione)



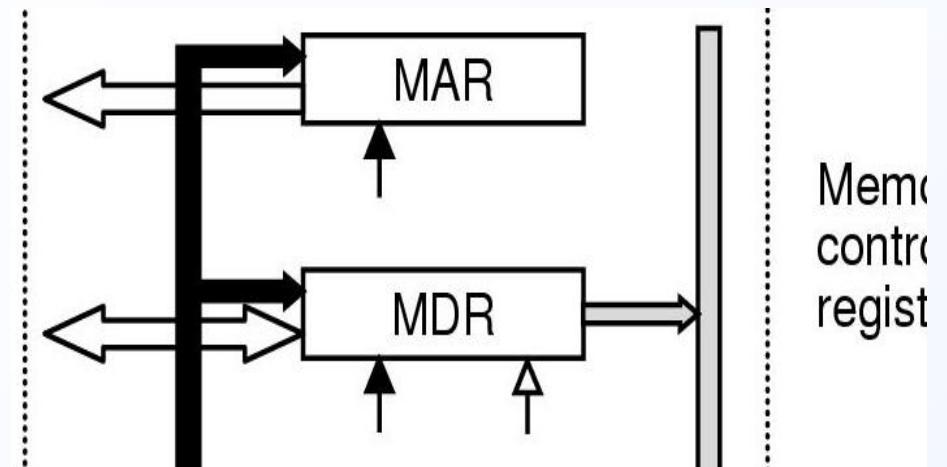
Funzionamento della memoria

La CPU ha due modi per comunicare con la memoria:

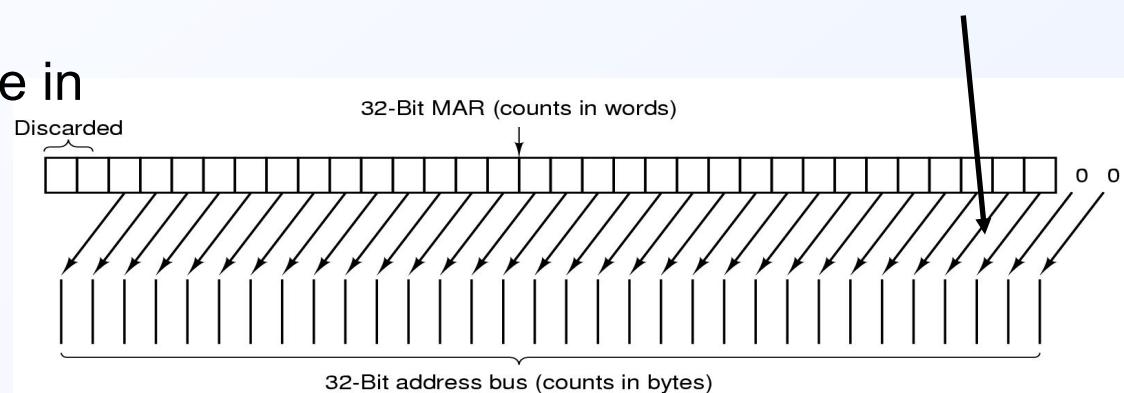
- Una porta di memoria da **32 bit** indirizzabile a parola controllata dai registri **MAR** (Memory address Register) e **MDR** (Memory Data Register)
- Una porta di memoria da **8 bit** indirizzabile a byte controllata dal registro **PC** che legge i byte e li memorizza negli 8 bit meno significativi di MBR. Questa porta può solo leggere in memoria e non può scrivere.

I registri MAR e MDR

- Porta di memoria a 32 bit
- MAR = Memory Address Register
- MDR = Memory Data Register
- Scrittura e lettura a livello ISA
- MAR ha un solo segnale di controllo (solo input dal bus C ma non output verso il bus B)
- Particolare mappatura tra valore in MAR e indirizzo in memoria:
 - MAR indirizza le parole
 - La memoria fisica conta in byte

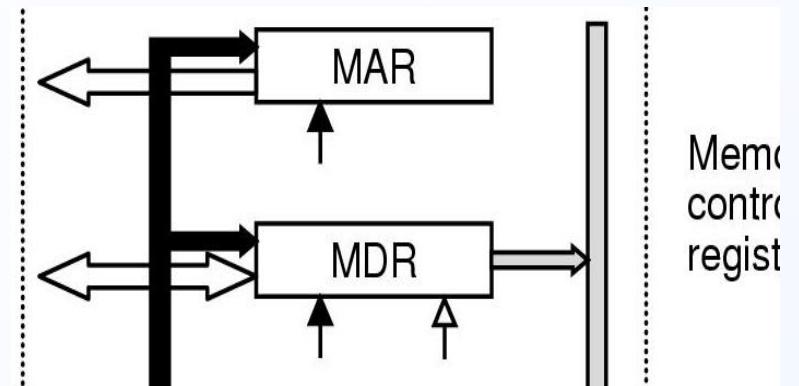


Moltiplica per 4!

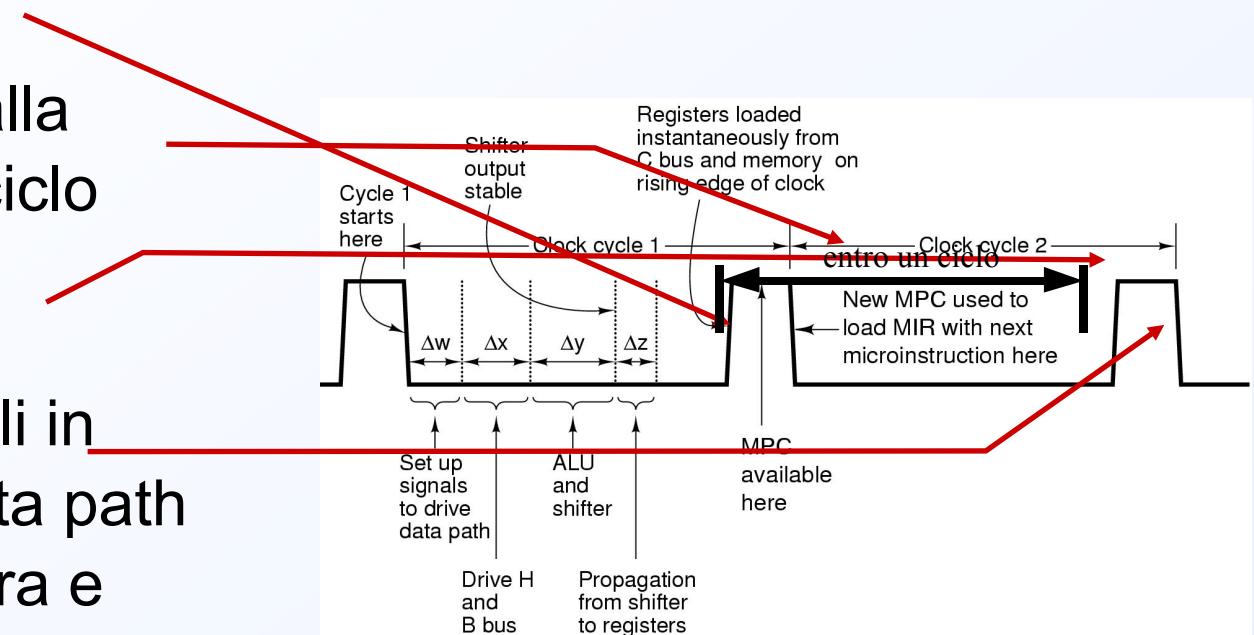


Accesso alla memoria

- Se viene inviata una richiesta di lettura di memoria nel ciclo k i dati saranno disponibili in MDR solo nel ciclo $k + 2$

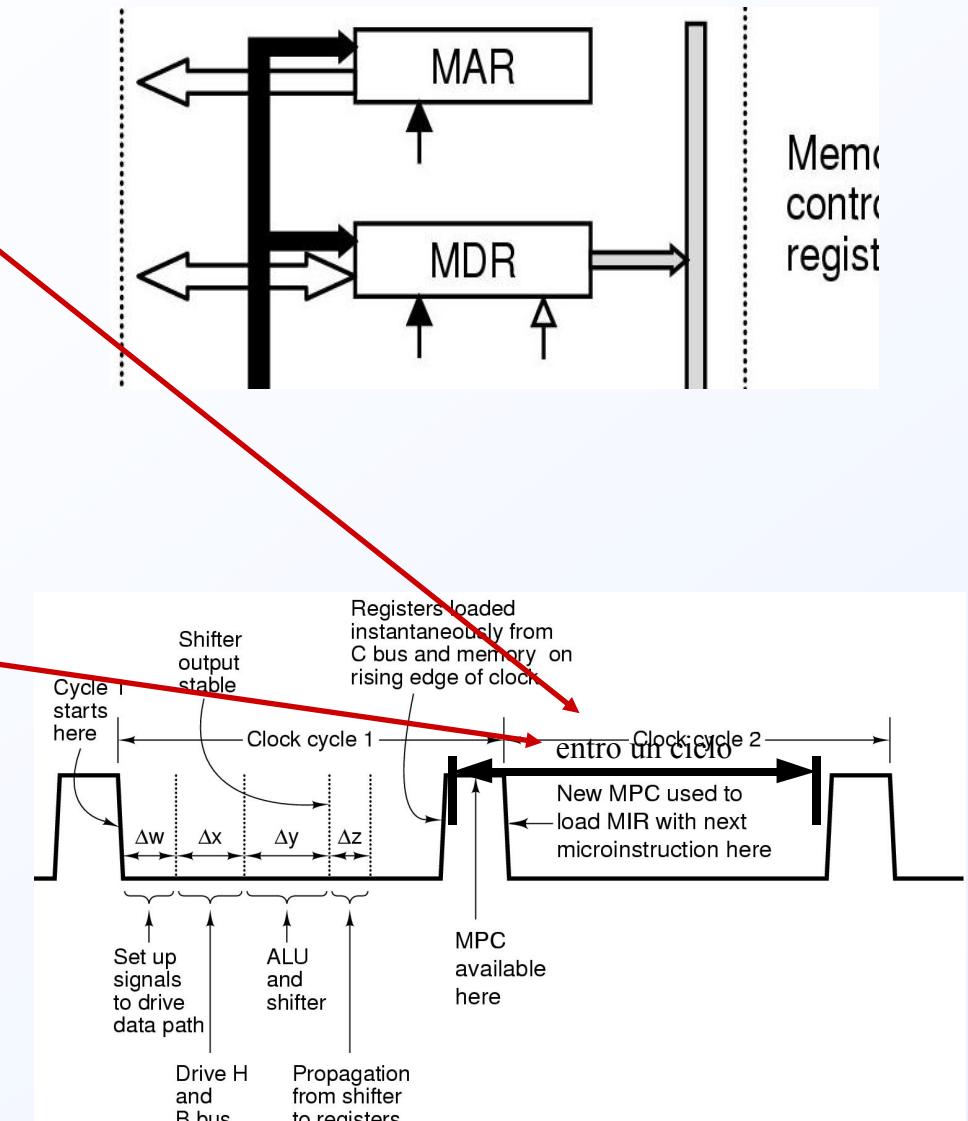


- MAR è caricato qui
- Tempo di accesso alla memoria: entro un ciclo
- MDR è caricato qui
- I dati sono disponibili in MDR (un ciclo di data path tra l'avvio della lettura e l'utilizzo dei dati)



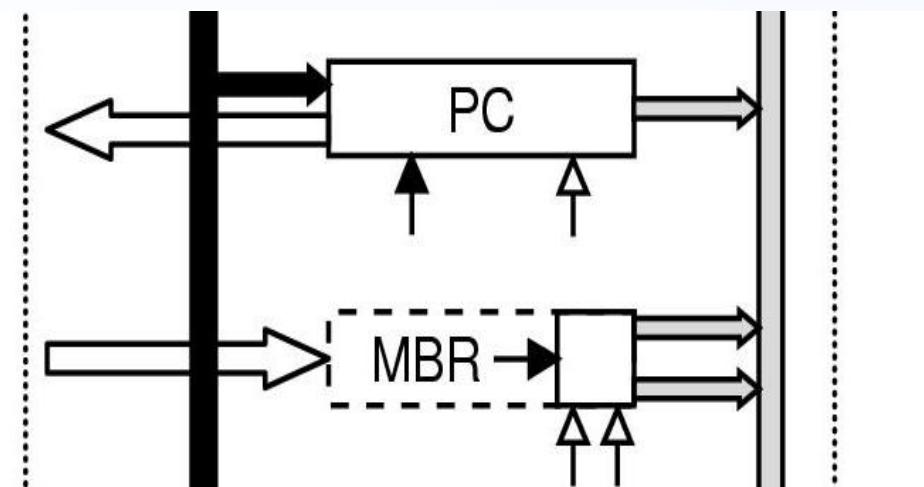
Accesso alla memoria

- Durante il tempo di accesso alla memoria:
 - si può fare altro che non utilizzi i dati che aspettiamo
 - i dati letti da MDR sono quelli vecchi (ciò non è un male se si desidera proprio questo)
- Si possono anche effettuare due richieste di lettura consecutiva
- 2 segnali** per la lettura e scrittura *da* e *per* la memoria



I registri MBR e PC

- Porta di memoria a **8 bit**
- MBR = Memory Buffer Register
- PC = Program Counter
- Lettura del *programma del livello ISA* da eseguire
- MBR: due bit di controllo per l'output su B: ***signed*** e ***unsigned***
- Valgono anche per MBR le considerazioni sui tempi di accesso alla memoria fatti per MDR
- Serve **1** segnale per avviare il fetch da memoria su MBR



Esempio signed and unsigned

Unsigned

- Se MBR contiene i bit **01001000** allora nel bus B verrà inviato

00000000 00000000 00000000 **01001000**

- Se MBR contiene i bit **10011010** allora nel bus B verrà inviato

00000000 00000000 00000000 **10011010**

- In comune hanno l'introduzione di 24 zeri nei supplementari 24 bit

Signed

- Se MBR contiene i bit **01001000** (8 bit in complemento a 2, 72 in base 10) allora nel bus B verrà inviato

00000000 00000000 ~~00000000~~ **01001000**

(32 bit in complemento a 2, sempre 72 in base 10)

- Se MBR contiene i bit **10011010** (8 bit in complemento a 2, -102 in base 10) allora nel bus B verrà inviato

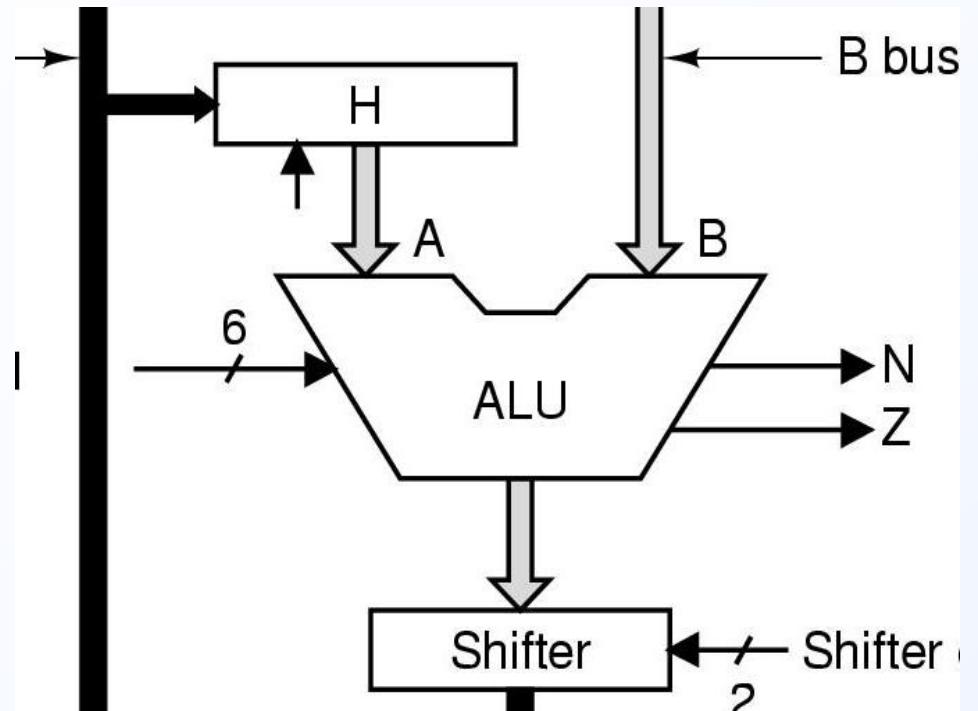
11111111 11111111 11111111 **10011010**

(32 bit in complemento a 2, sempre -102 in base 10)

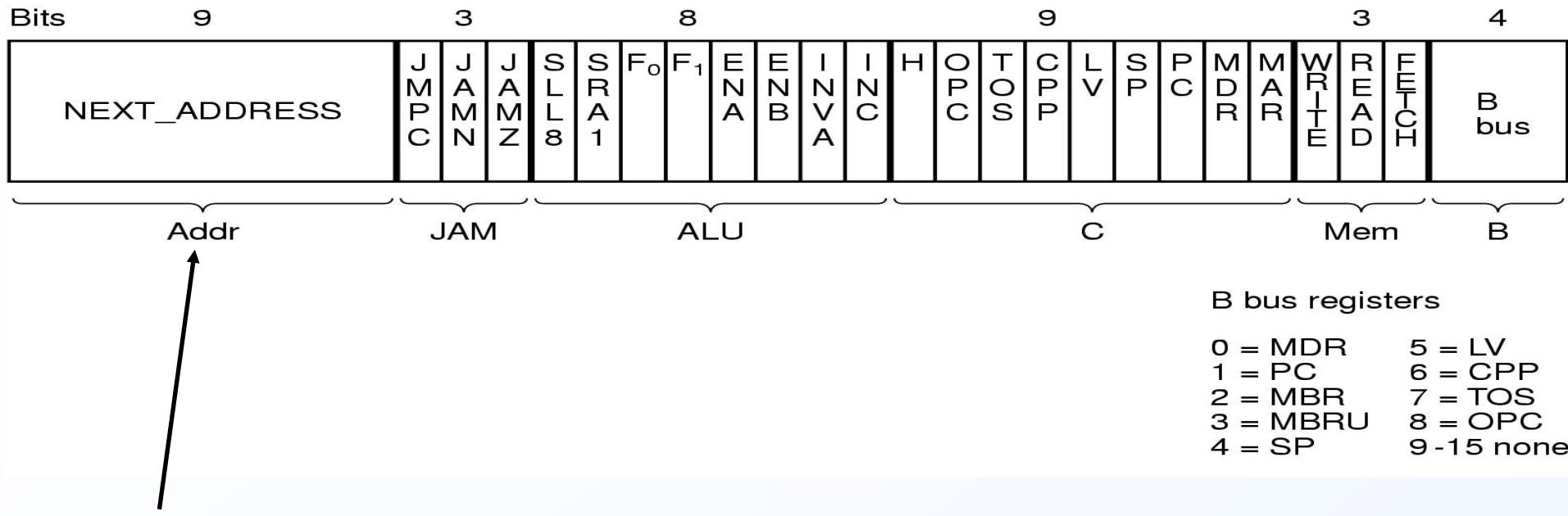
- In comune hanno la replica del bit più significativo nei supplementari 24 bit

Registro H

- Rappresenta l'input di sinistra dell'ALU
- Un solo bit di controllo per l'abilitazione dell'input dal bus C, l'output verso l'ALU è sempre attivo

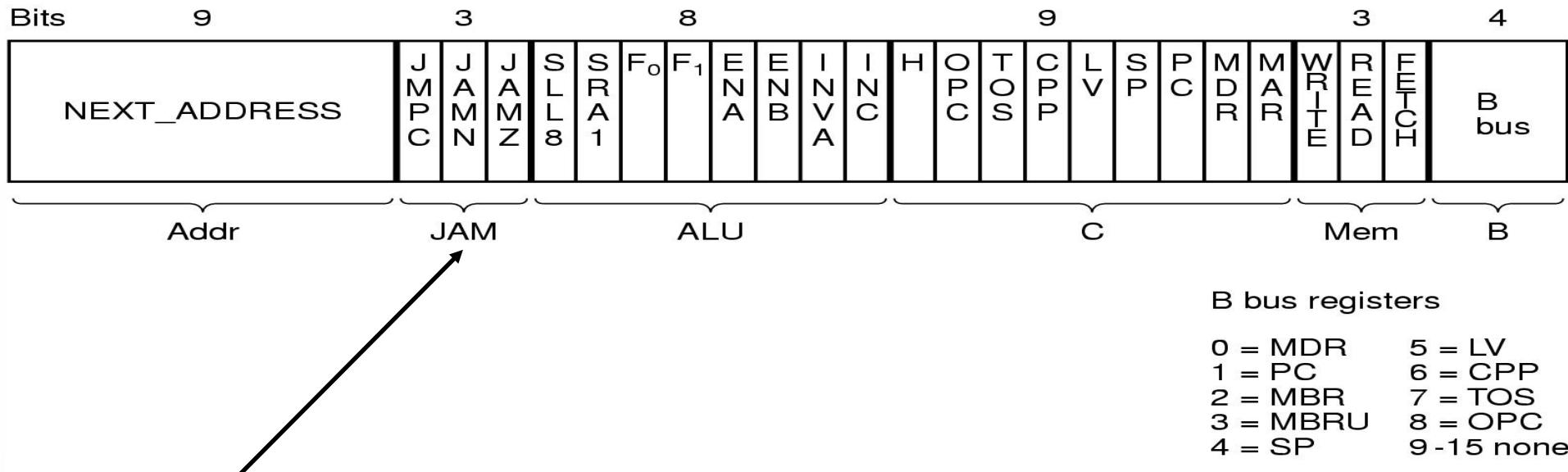


Formato di una microistruzione



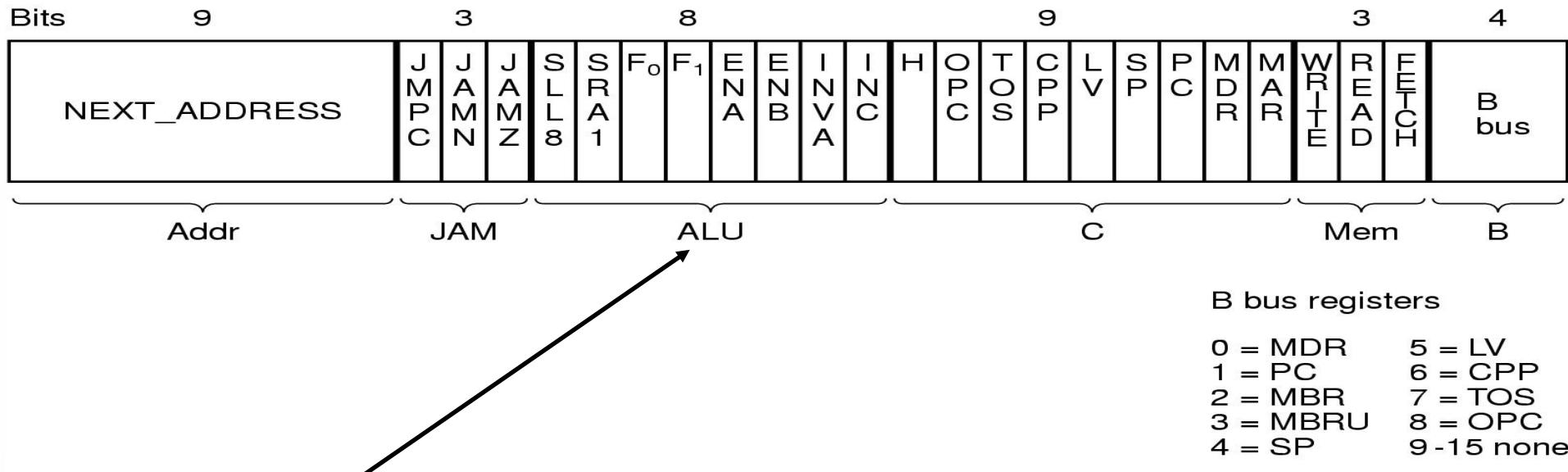
- * **Addr**: indirizzo della microistruzione seguente

Formato di una microistruzione



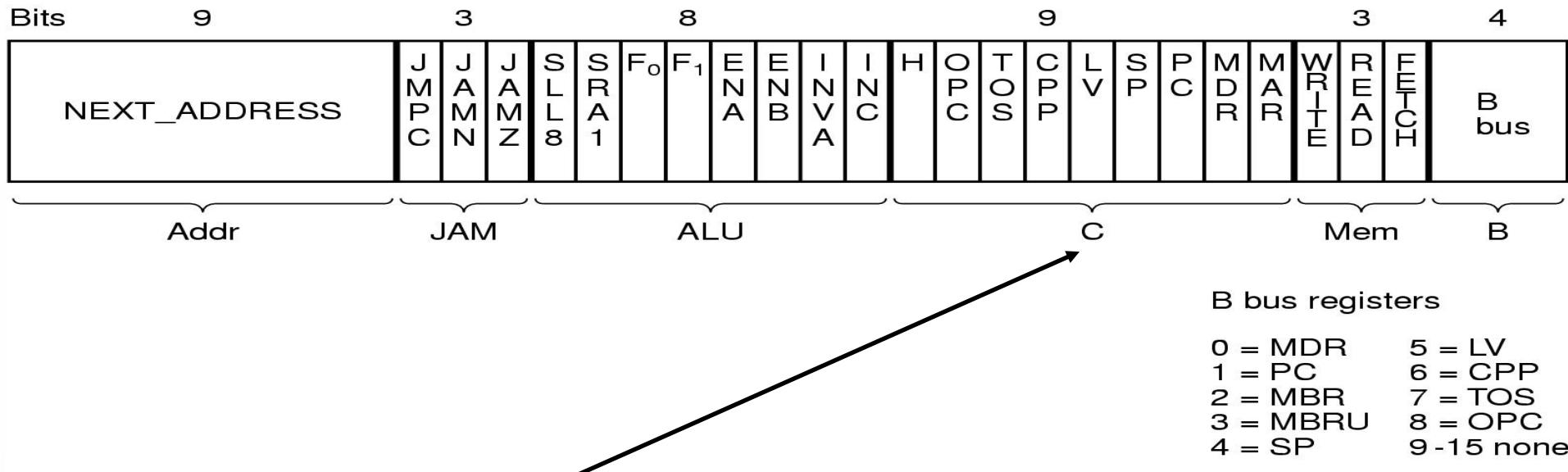
- × **Addr**: indirizzo della microistruzione seguente
- × **JAM**: determina come selezionare l'istruzione successiva

Formato di una microistruzione



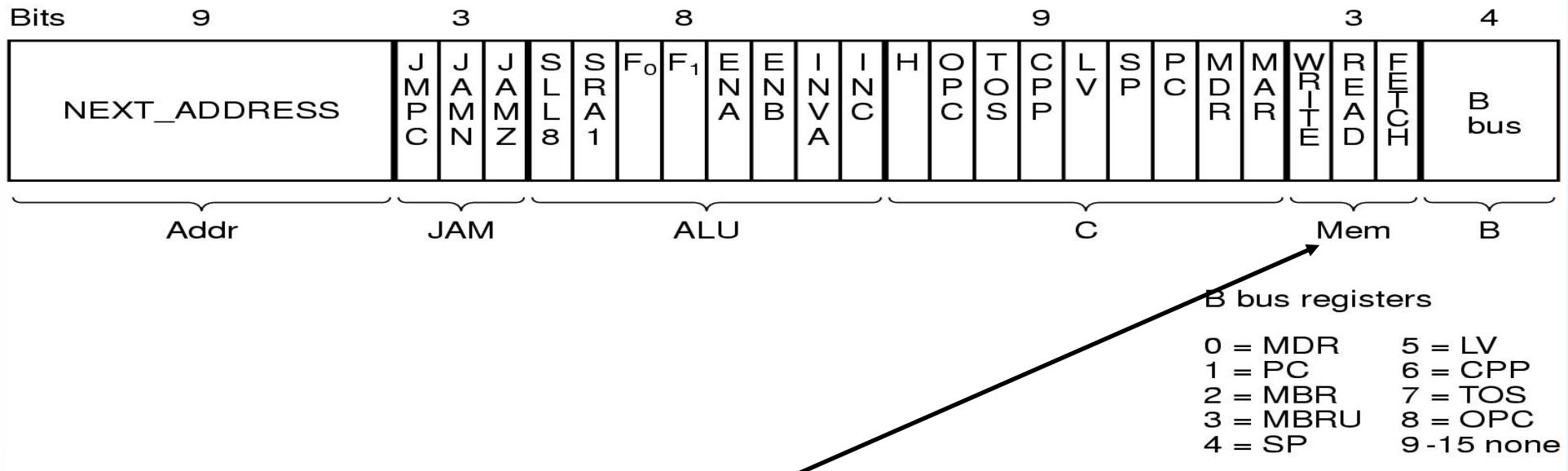
- × **Addr**: indirizzo della microistruzione seguente
- × **JAM**: determina come selezionare l'istruzione successiva
- × **ALU**: bit di controllo dell'ALU e dello shifter

Formato di una microistruzione



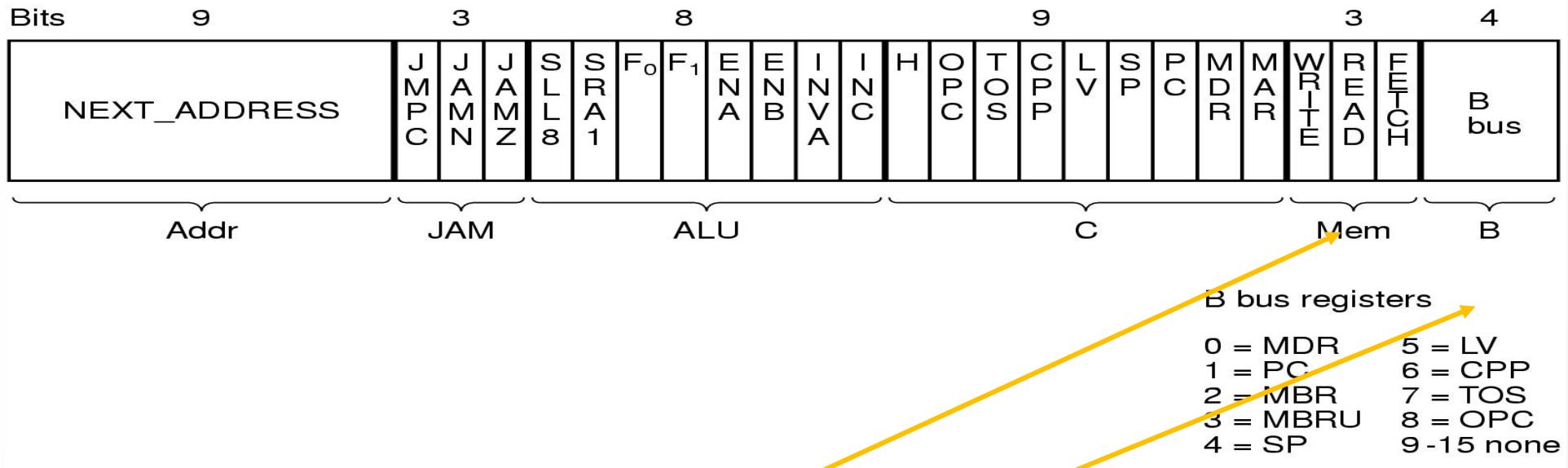
- × **Addr**: indirizzo della microistruzione seguente
- × **JAM**: determina come selezionare l'istruzione successiva
- × **ALU**: bit di controllo dell'ALU e dello shifter
- × **C**: seleziona i registri su cui memorizzare il valore del bus C

Formato di una microistruzione



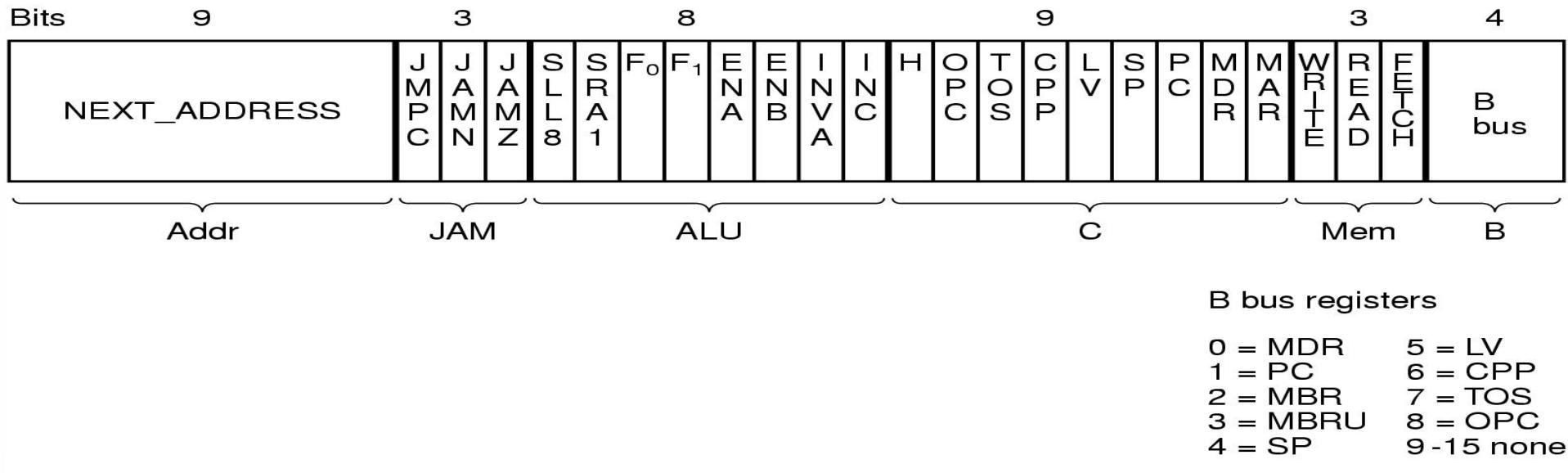
- × **Addr**: indirizzo della microistruzione seguente
- × **JAM**: determina come selezionare l'istruzione successiva
- × **ALU**: bit di controllo dell'ALU e dello shifter
- × **C**: seleziona i registri su cui memorizzare il valore del bus C
- × **Mem**: controlla le operazioni sulla memoria

Formato di una microistruzione



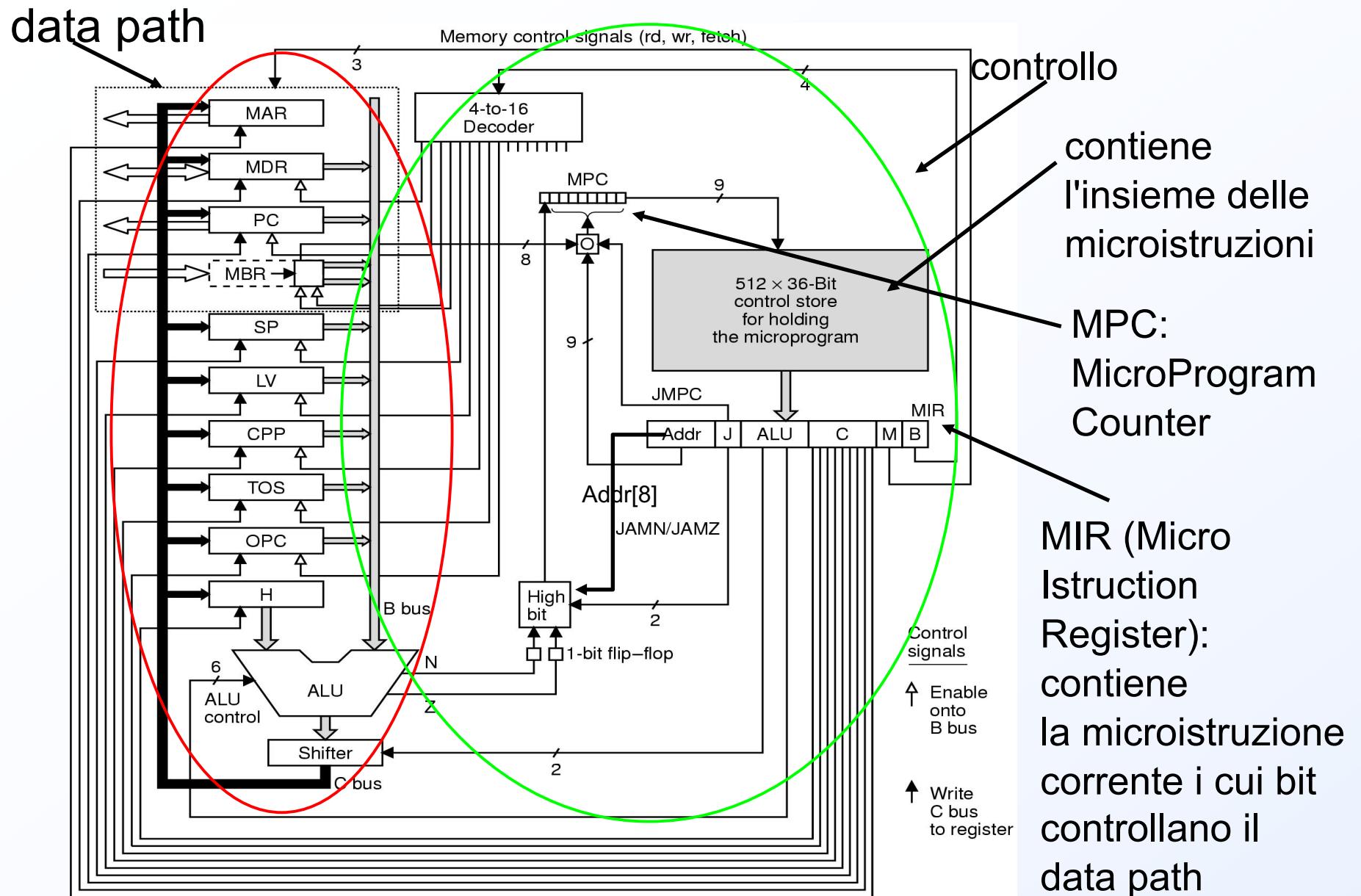
- × **Addr**: indirizzo della microistruzione seguente
- × **JAM**: determina come selezionare l'istruzione successiva
- × **ALU**: bit di controllo dell'ALU e dello shifter
- × **C**: seleziona i registri su cui memorizzare il valore del bus C
- × **Mem**: controlla le operazioni sulla memoria
- × **B**: seleziona l'input per il bus B

Formato di una microistruzione

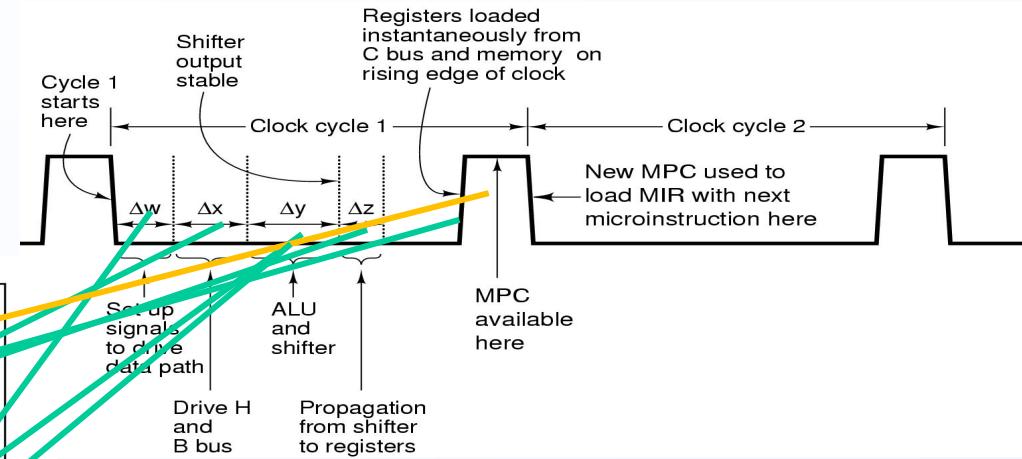
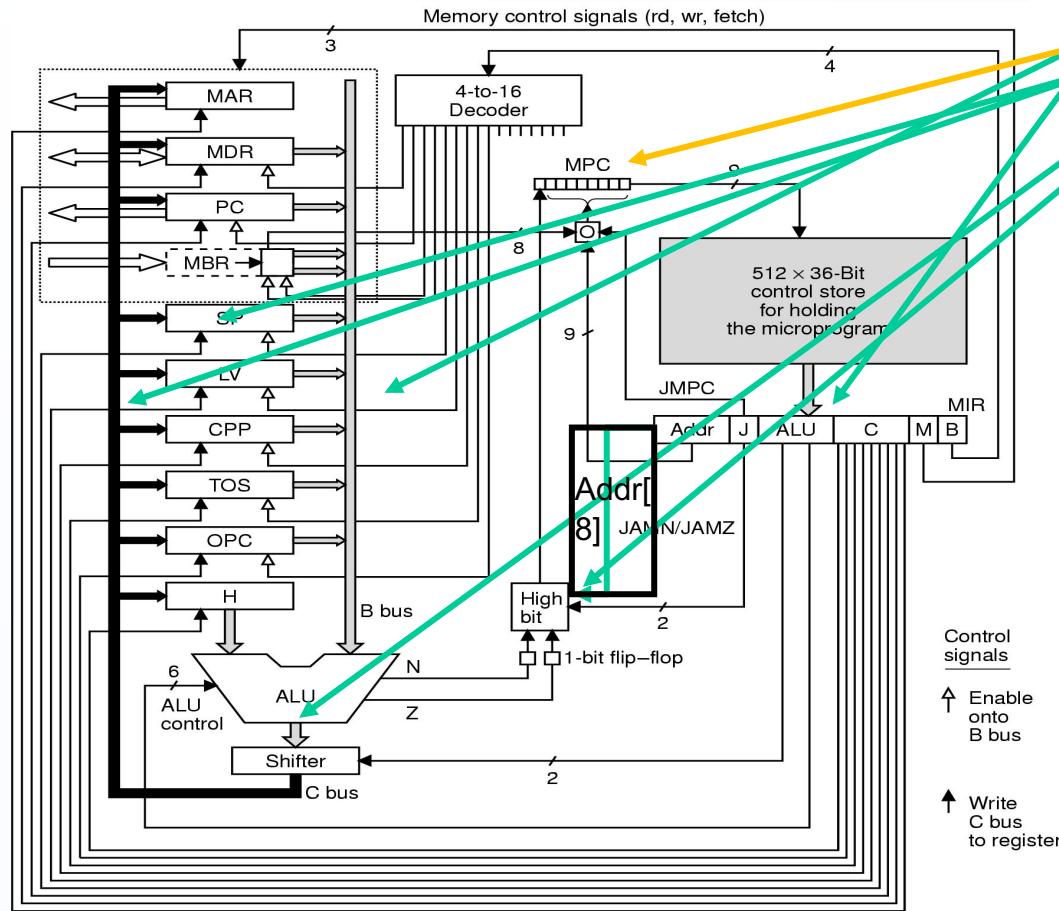


- **Addr**: indirizzo della microistruzione seguente
 - **JAM**: determina come selezionare l'istruzione successiva
 - **ALU**: bit di controllo dell'ALU e dello shifter
 - **C**: seleziona i registri su cui memorizzare il valore del bus C
 - **Mem**: controlla le operazioni sulla memoria
 - **B**: seleziona l'input per il bus B
 - **TOTALE: 36 bit per istruzione**

Architettura del Mic-1



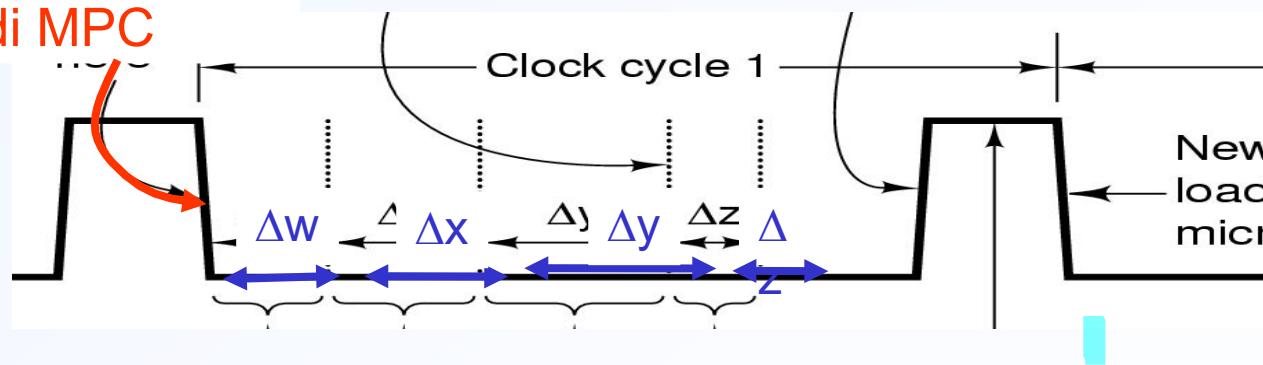
Organizzazione dei sottocicli



Sul fronte di discesa del clock il MIR viene caricato in base al valore di MPC, questi si propagano per controllare i bus e l'ALU, quindi si memorizza i valori nei registri sul fronte di salita del ciclo successivo

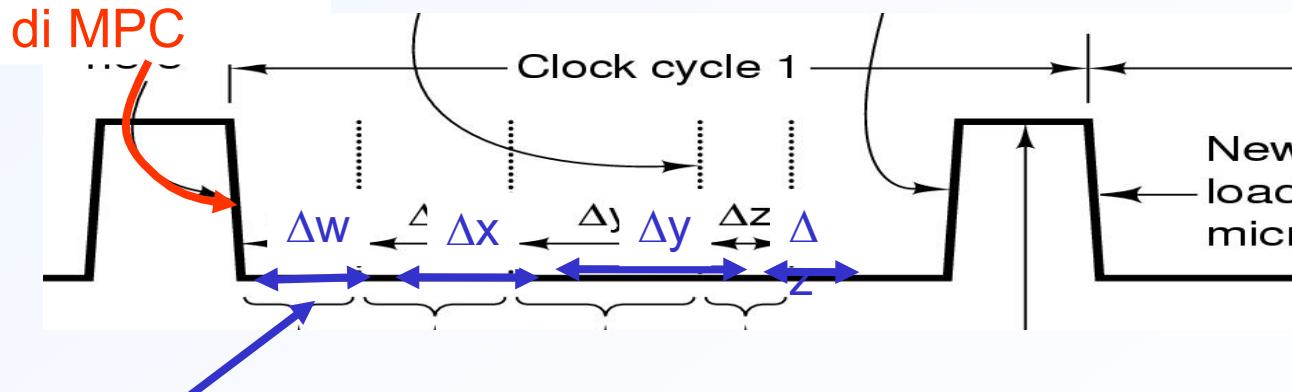
Organizzazione dei sottocicli

MIR viene caricato
in base al valore
di MPC



Organizzazione dei sottocicli

MIR viene caricato
in base al valore
di MPC

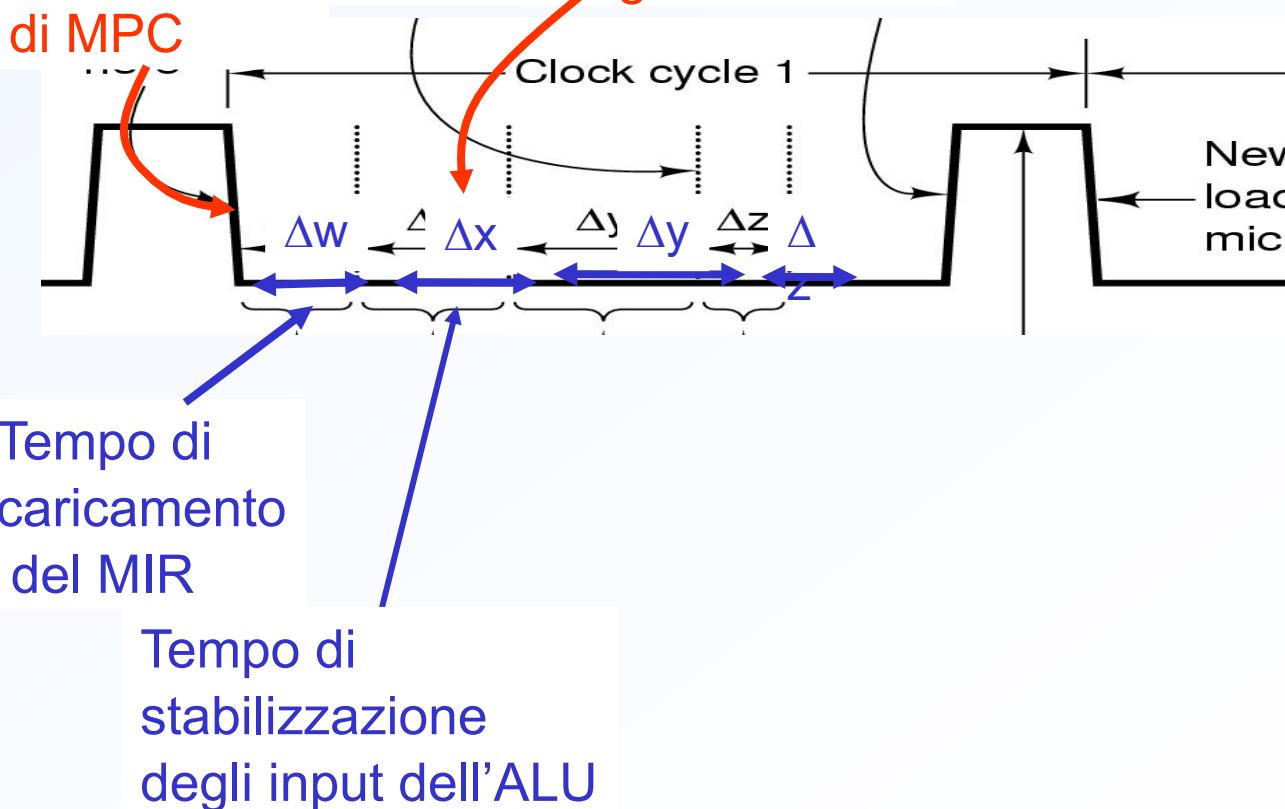


Tempo di
caricamento
del MIR

Organizzazione dei sottocicli

MIR viene caricato
in base al valore
di MPC

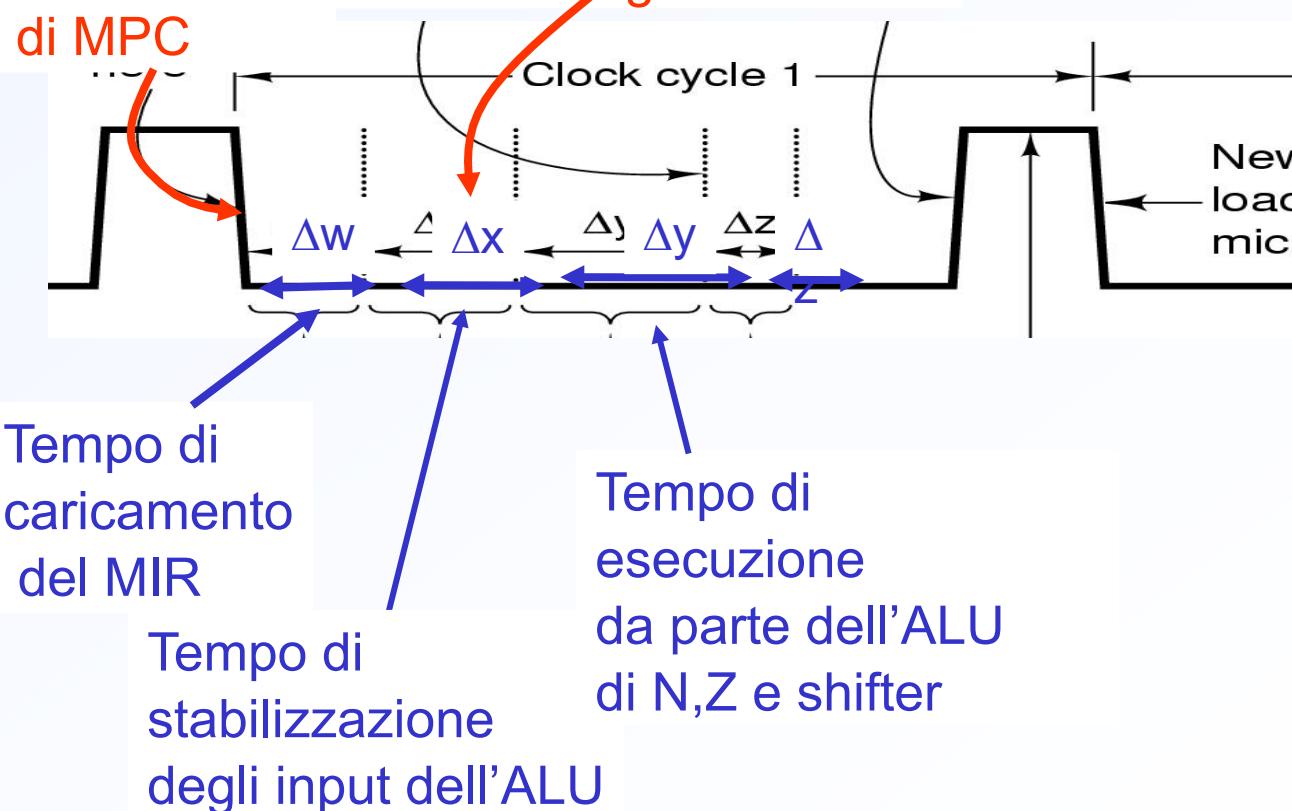
I segnali si propagano
nel data path: un registro
viene scritto sul bus B e
l'ALU sa che operazione
eseguire



Organizzazione dei sottocicli

MIR viene caricato
in base al valore
di MPC

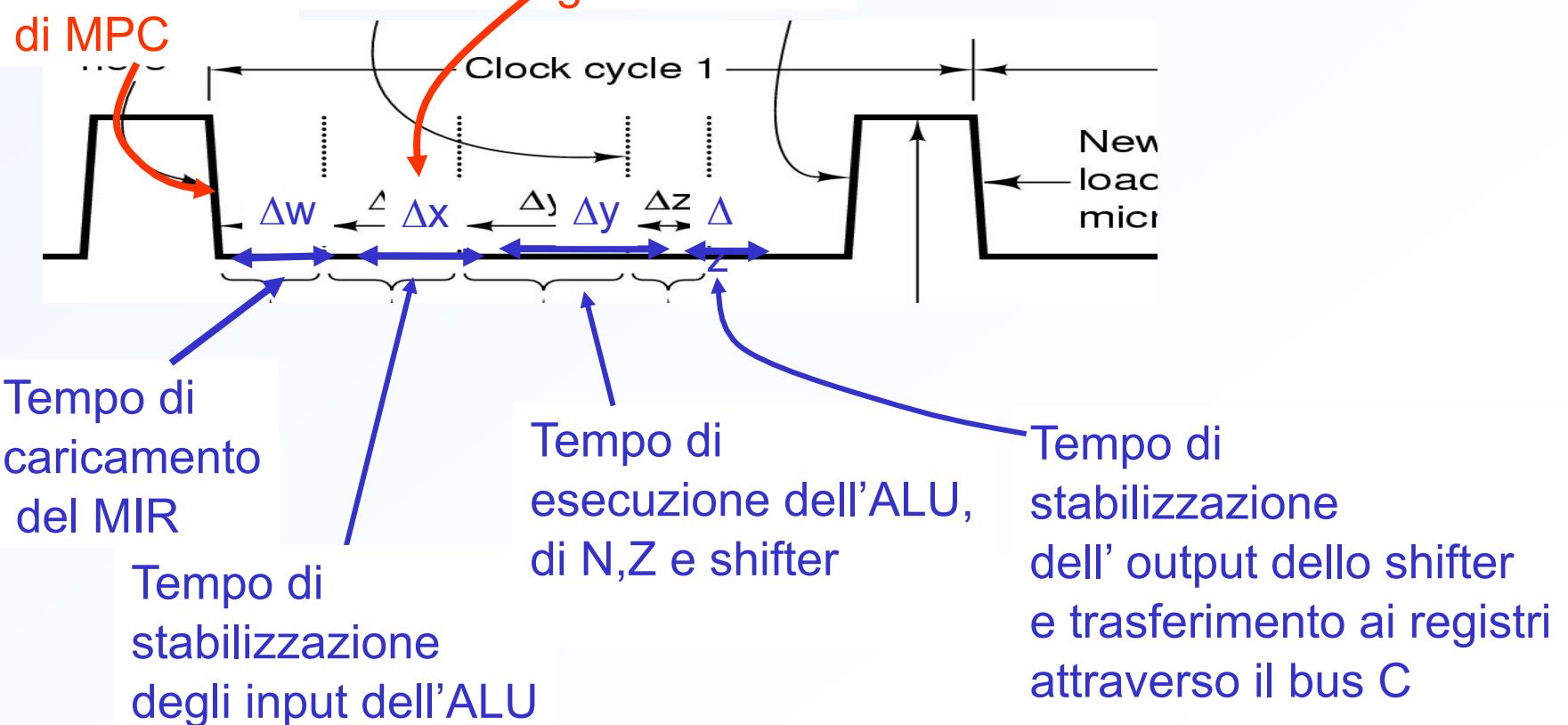
I segnali si propagano
nel data path: un registro
viene scritto sul bus B e
l'ALU sa che operazione
eseguire



Organizzazione dei sottocicli

MIR viene caricato
in base al valore
di MPC

I segnali si propagano
nel data path: un registro
viene scritto sul bus B e
l'ALU sa che operazione
eseguire

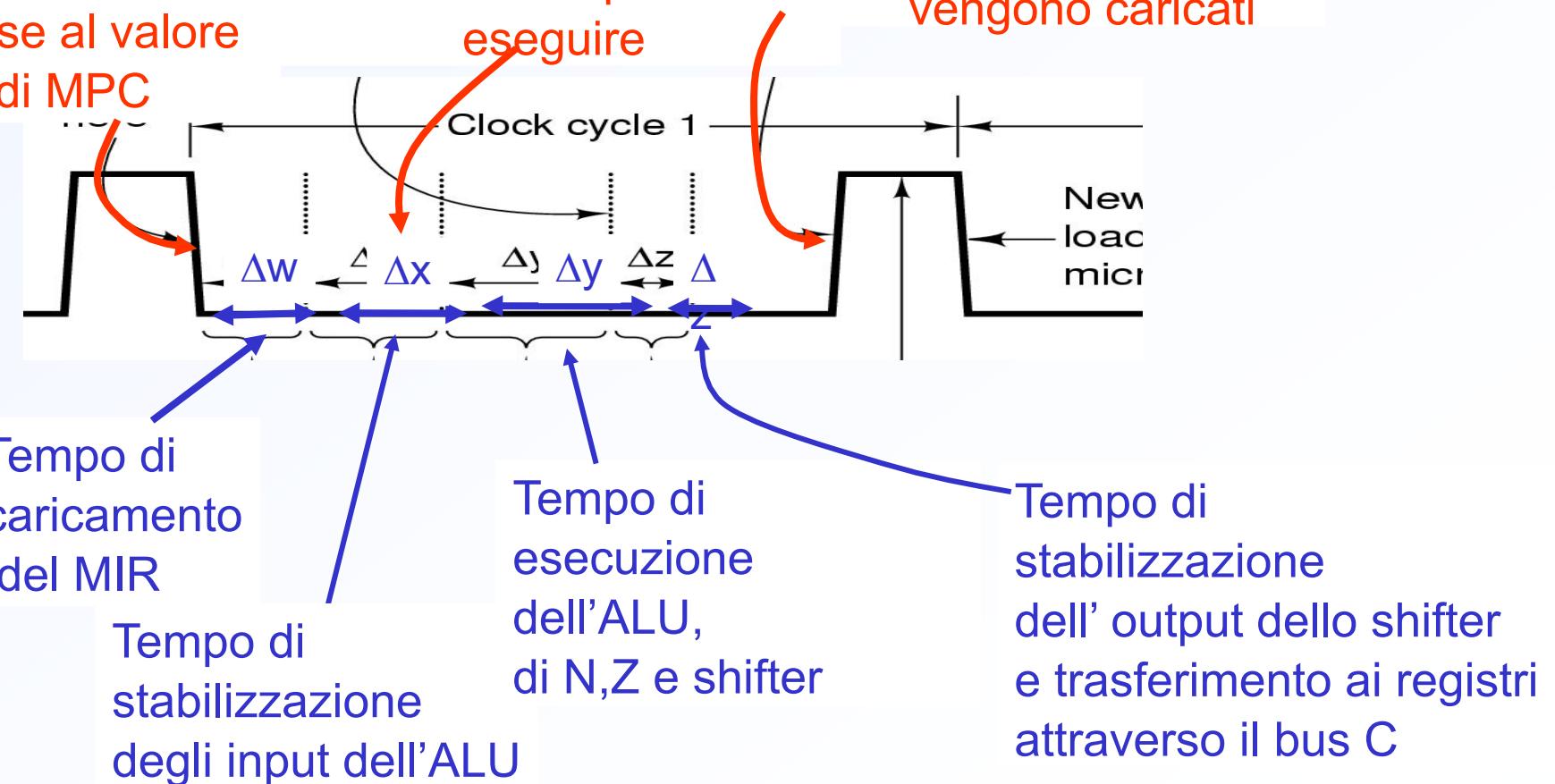


Organizzazione dei sottocicli

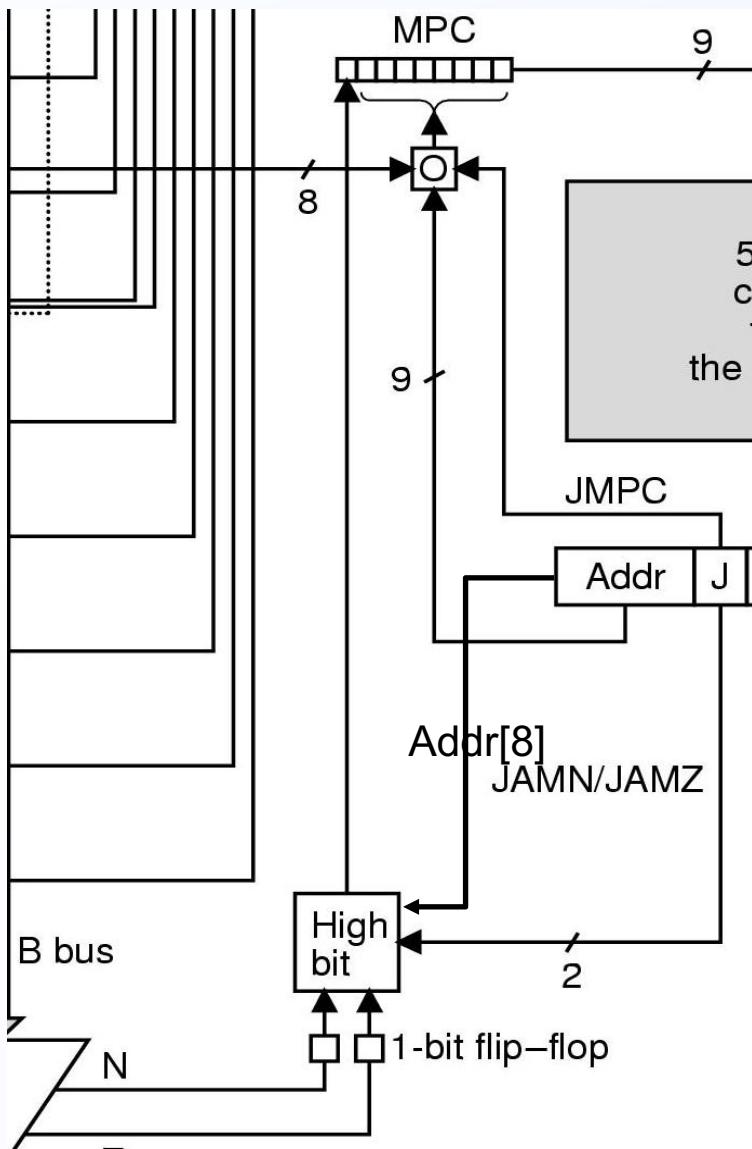
MIR viene caricato
in base al valore
di MPC

I segnali si propagano
nel data path: un registro
viene scritto sul bus B e
l'ALU sa che operazione
eseguire

I registri ed i flip-flop
vengono caricati

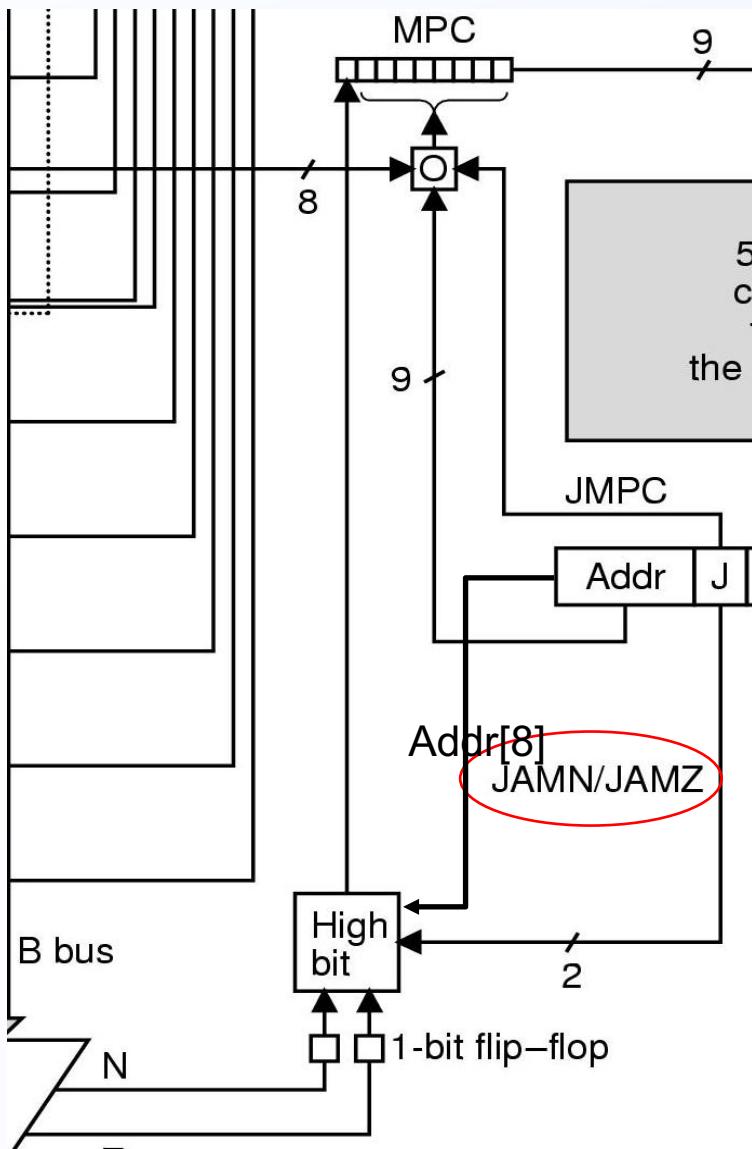


Determinare l'istruzione successiva



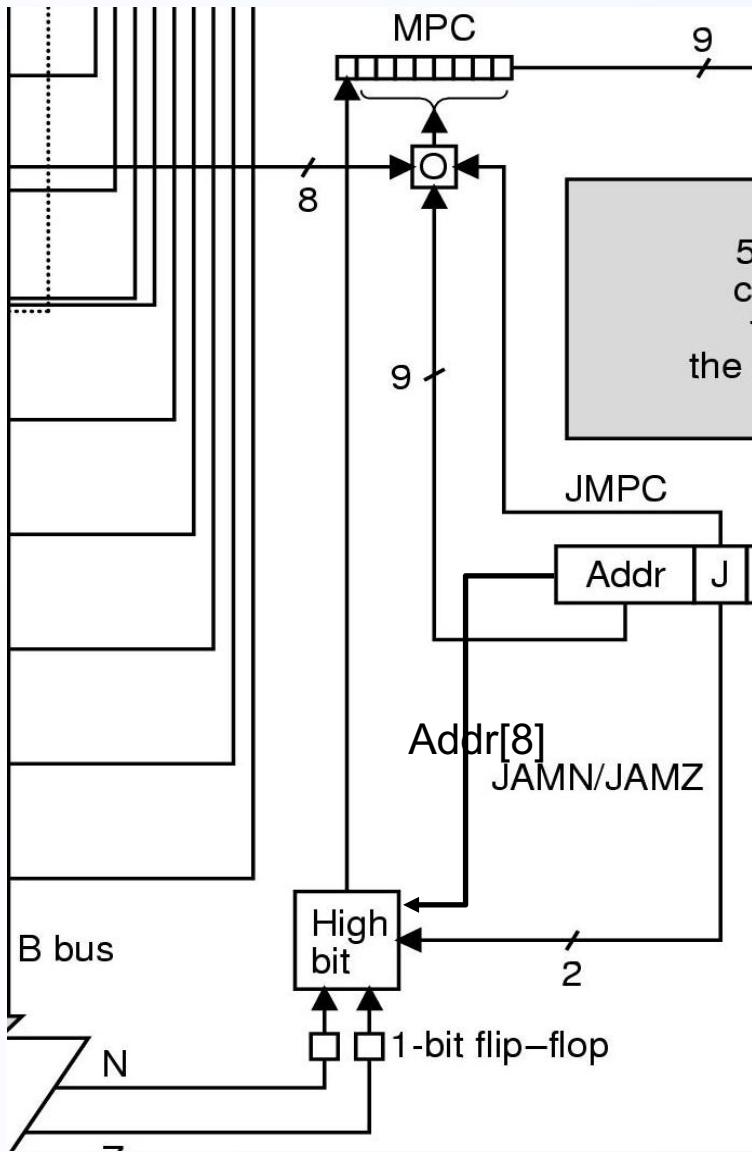
- Le microistruzioni non sono memorizzate nella ROM come compaiono sul testo del microinterprete
- Addr è copiato su MPC
- Se J è 000 non si fa altro e MPC contiene l'indirizzo della microistruzione successiva
- Se J assume altri valori è necessario effettuare il calcolo dell'istruzione successiva in base al valore di J e di N e Z (memorizzati su opportuni flip-flop per garantire la correttezza dei valori utilizzati)

Determinare l'istruzione successiva



- Se JAMN o JAMZ hanno valore diverso da 0 si procede `High bit` imposta il bit più alto di MPC con il risultato della seguente espressione booleana:
 - $(\text{JAMZ and } Z) \text{ or } (\text{JAMN and } N) \text{ or } \text{Addr}[8]$
- Questo significa che MPC:
 - ha il valore di Addr oppure
 - ha il valore di Addr con il bit più significativo in or con 1

Determinare l'istruzione successiva



- **(JAMZ and Z) or (JAMN and N) or Addr[8]**
- **Esempio:**
 - supponiamo $\text{Addr} \leq 0xFF$ (altrimenti abbiamo come risultato Addr stesso!)
 - supponiamo $\text{JAMZ} = 1$ (analogo per JAMN)
 - allora **MPC è uguale a $\text{Addr} + 0x100$ (es $0x92 + 0x100 = 0x192$)**
 - Nota: $0x100 = 256$ (notazione esadecimale)

Perchè?

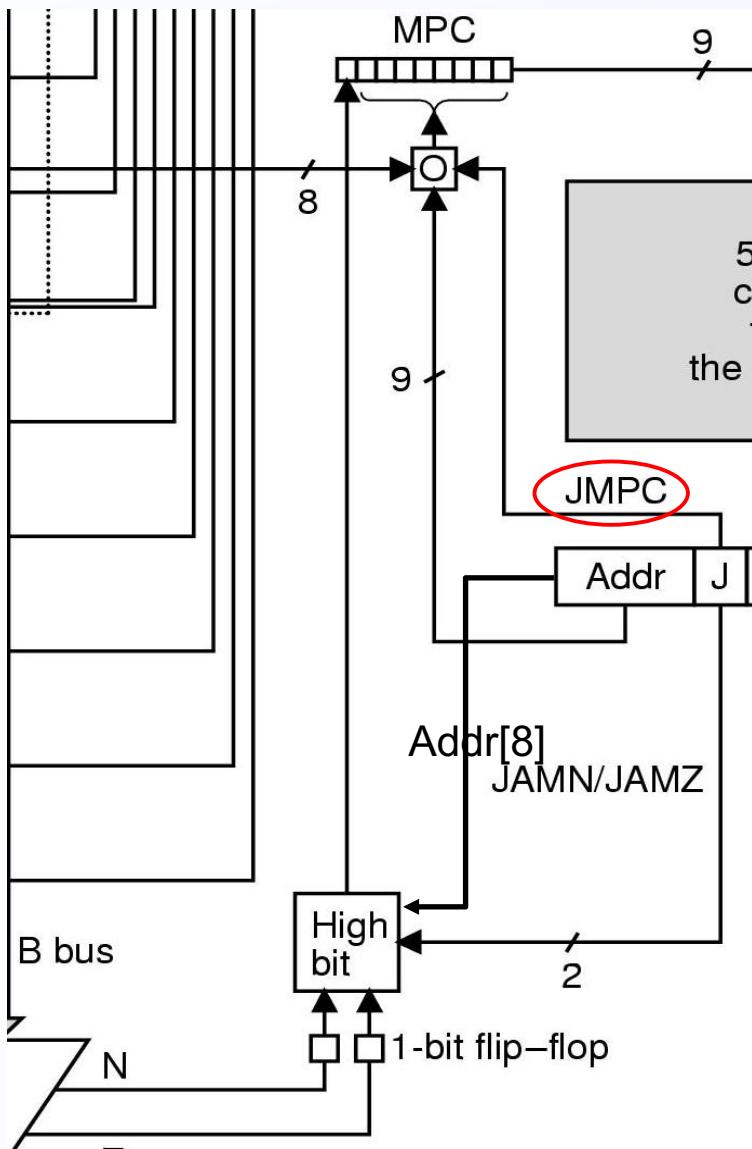
- 1) Ogni microistruzione contiene l'indirizzo dell'istruzione successiva
- 2) Un'istruzione di salto condizionato solitamente viene interpretata come: istruzione sucessiva se la condizione non è vera, salto ad un certo indirizzo se la condizione è vera
- 3) Quindi una microistruzione di salto condizionato dovrebbe specificare 2 indirizzi: uno è l'indirizzo dell'istruzione successiva (vedi punto 1) se la condizione non è vera, l'altro è l'indirizzo dell'istruzione a cui saltare se la condizione è vera (vedi punto 2)
- 4) Le microistruzioni di salto condizionato necessitano di 36 + 9 bit (i 9 sono dati dal punto 3)

Perché?

- 5) Per questioni di efficienza questo non è accettabile perchè si preferiscono microistruzioni tutte con lo stesso formato
- 6) Non sarebbe accettabile in termini di costi uniformare tutte le istruzioni a contenere sempre due indirizzi (dove la maggior parte delle volte is secondo non sarebbe utilizzato)

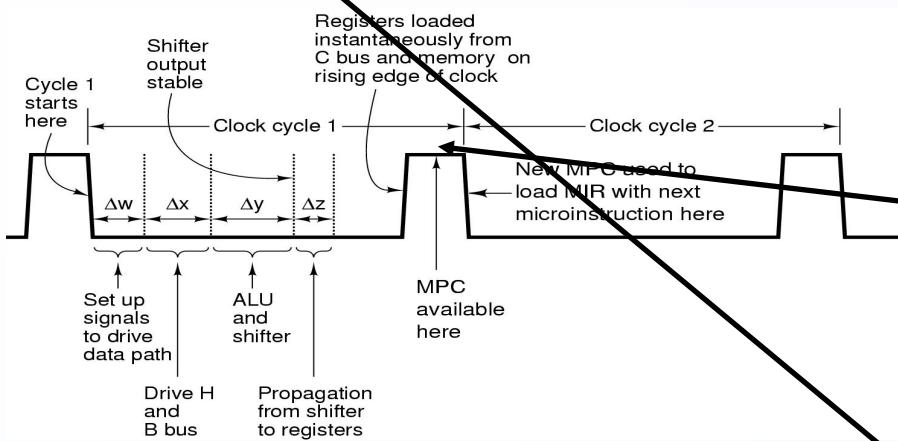
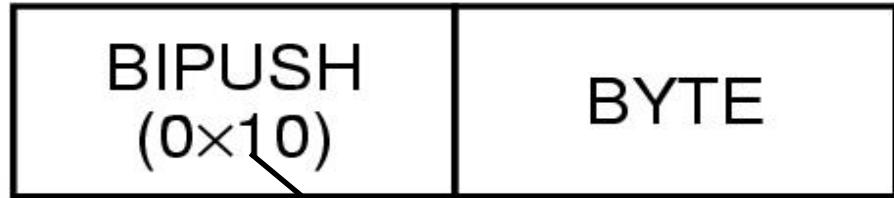
Soluzione: i due indirizzi vengono specificati come 'X' e 'X + costante', nel nostro caso X è Addr e costante = 256

Determinare l'istruzione successiva

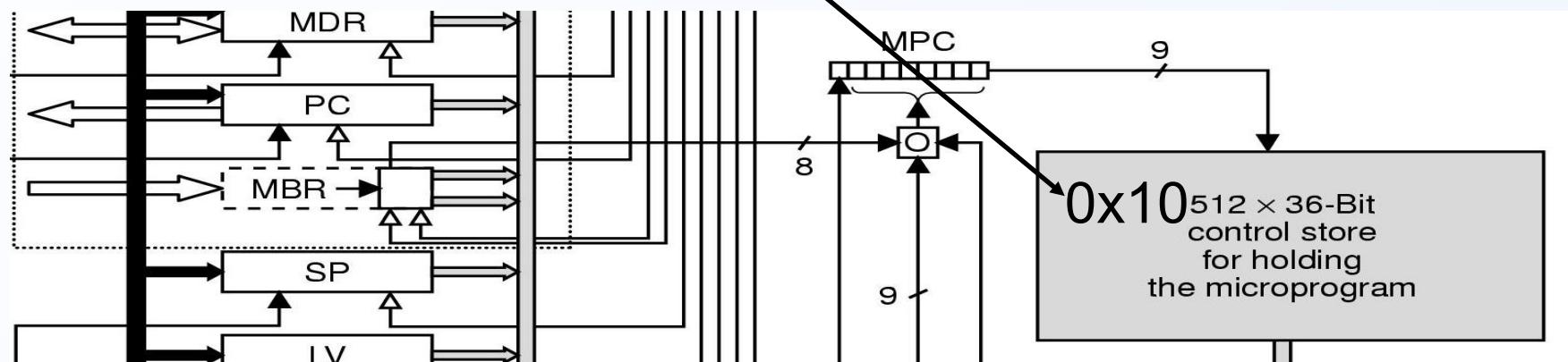


- Se JMPC vale 0 allora Addr è copiato in MPC
- Se JMPC ha valore 1 allora si esegue l'OR tra gli 8 bit meno significativi di Addr con MBR, il risultato è inviato a MPC
- In generale quando JMPC vale 1 in Addr è memorizzato 0x000 oppure 0x100
- Implementa un salto a più vie, viene utilizzato per saltare all'indirizzo contenuto in MBR (*opcode* dell'istruzione)
- Cioè le microistruzioni sono memorizzate a partire dalla posizione data dal loro *opcode*

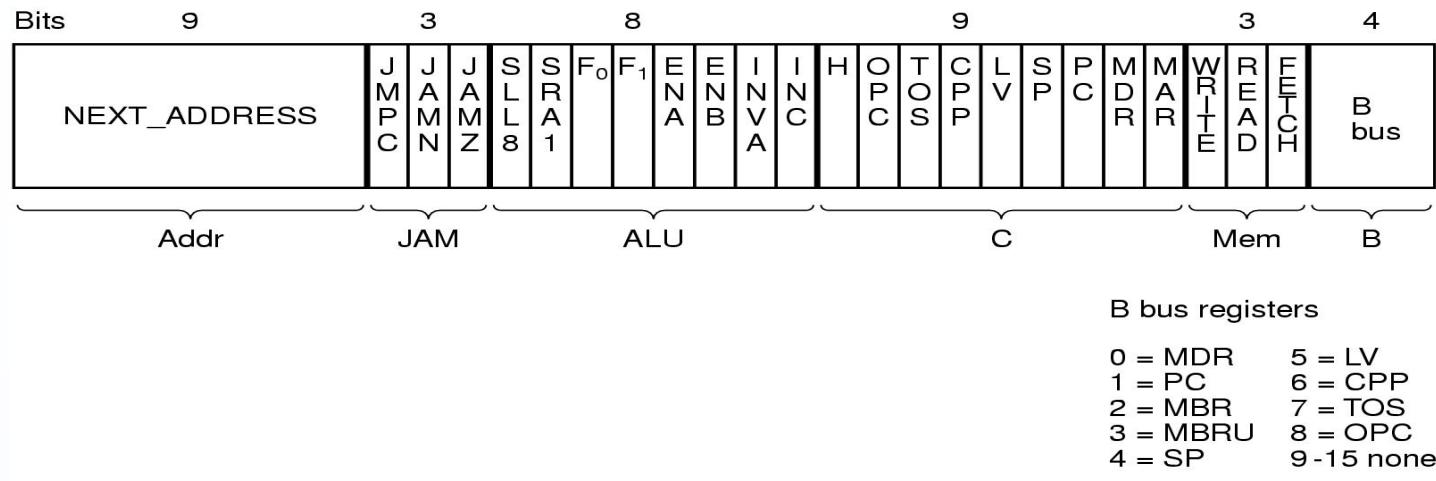
Determinare l'istruzione successiva



- Esempio, l'istruzione ISA IJVM **BIPUSH** è codificata dal byte **0x10**
- Nel control store la sequenza delle microistruzioni che interpretano **BIPUSH** inizia all'indirizzo **0x10**
- È quindi importante che MPC venga caricato solo dopo che MBR, N e Z siano pronti (cioè dopo il fronte di salita del ciclo successivo)



Esempio di microistruzione



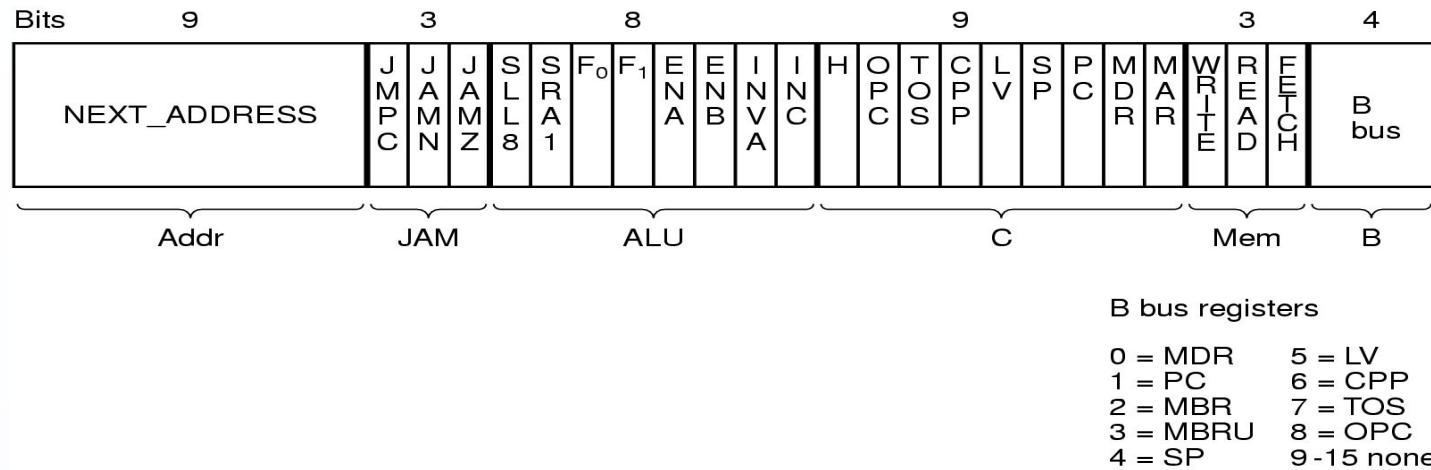
xxxxxxxxx 000 00 110110 000001001 010 0100

Cosa fa questa microistruzione?

- decrementa SP: $SP = SP - 1$
- assegna il valore di SP a MAR
- fa partire una operazione di lettura

MAR = SP = SP-1; rd

MAL (Micro Assembly Language)



xxxxxxxxx 000 00 111100 000000010 000 0100

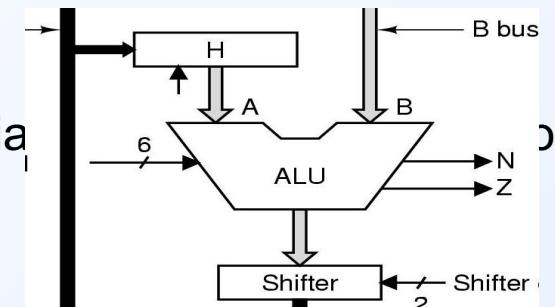
Notazione:

- MDR = H + SP
- MDR = SP + H

è equivalente ma è preferibile che "H" sia l'operando di sinistra

- MDR = SP + MDR

non è legale (le operazioni binarie hanno necessariamente l'operando di sinistra)



Istruzioni MAL fra registri

- SOURCE $\in \{\text{MDR}, \text{PC}, \text{MBR}, \text{MBRU}, \text{SP}, \text{LV}, \text{CPP}, \text{TOS}, \text{OPC}\}$
- DEST $\in \{\text{MAR}, \text{MDR}, \text{PC}, \text{SP}, \text{LV}, \text{CPP}, \text{TOS}, \text{OPC}, \text{H}\}$
- Queste istruzioni possono essere modificate aggiungendo “ $<< 8$ ”, come per $\text{H} = \text{MBR} << 8$
- Salto esplicito: GOTO 0xyyyy...
- In assenza di GOTO, il MAL assume il salto alla micro-istruzione seguente
- Comandi rd, wr, fetch

DEST = H
DEST = SOURCE
DEST = \bar{H}
DEST = $\overline{\text{SOURCE}}$
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE - H
DEST = SOURCE - 1
DEST = -H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = -1

Istruzioni MAL di salto

- Per i salti condizionati può essere necessario esaminare un registro per stabilire, ad esempio se è zero:

TOS = TOS

- Per comodità di lettura in MAL si usano due registri *immaginari* Z e N:

Z = TOS

if (Z) goto L1; else goto L2

Z = TOS; if (Z) goto L1; else goto L2

- Notazione per l'uso di JMPC:

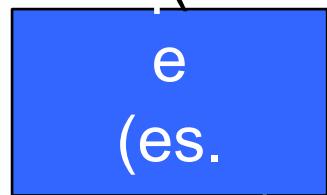
goto(MBR OR valore)

comunica di usare *valore* come NEXTADDRESS e di mettere a 1 il bit JMPC; se *valore* è 0 è sufficiente scrivere:

goto(MBR)

MAL vs IJVM

Programma in
linguaggio di alto
livello
↓
(es. Java)



↓
Programma
in IJVM



Microinterprete
scritto in MAL



Microinterprete

0101011001
0010101011
101101010101
0110011101
1010110010

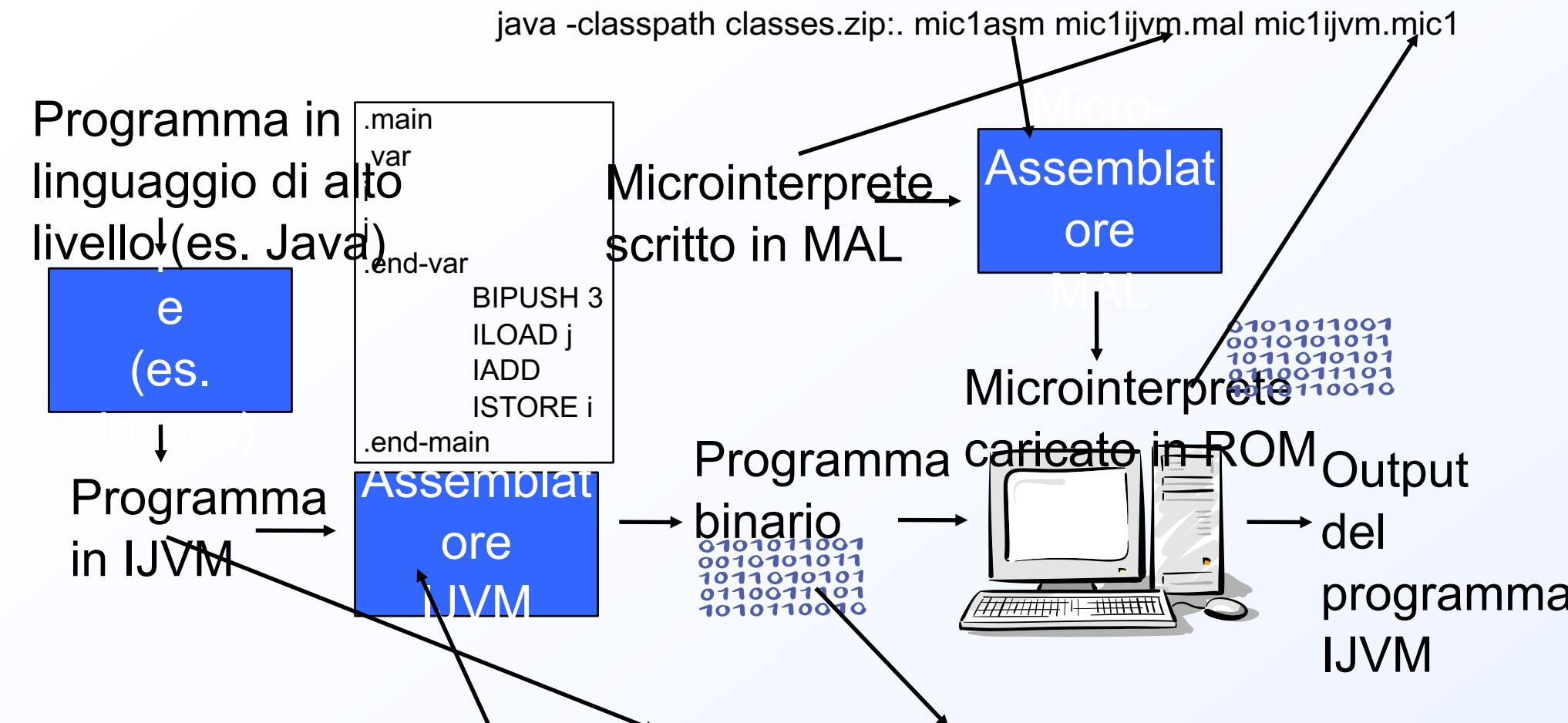
→ Programma
binario
0101011001
0010101011
101101010101
0110011101
1010110010



→ Programma caricato in ROM
Output
del
programm
IJVM

- Il microinterprete (livello 1) interpreta, pilotando il data-path, il programma scritto in IJVM (livello 2)
- Sia MAL che IJVM sono assemblati da opportuni programmi (l'assemblatore e il microassemblatore)

MAL vs IJVM



`java -classpath classes.zip:. mic1sim mic1ijvm.mic1 assegnamento.ijv`

IJVM (liv. MAC) e MAL (liv. MIC)

Bytecode
IJVM
(binario)

Memoria

