Il Processore PD32

Introduzione all'Architettura Hardware e Programmazione Software

SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Pellegrini

Calcolatori Elettronici
Sapienza, Università di Roma

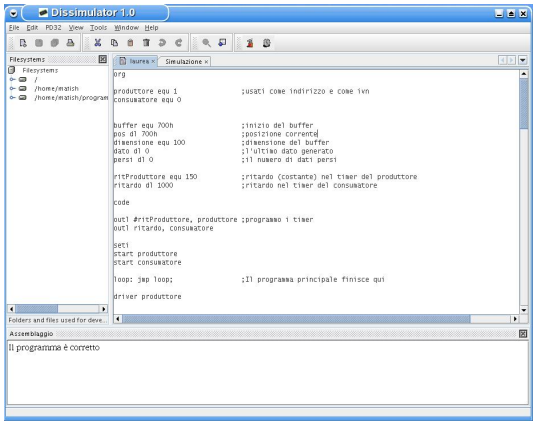
A.A. 2012/2013

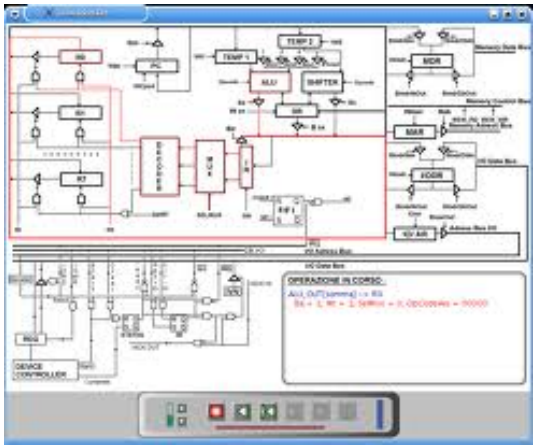
Obiettivo di questa parte del corso

- Introduzione alle basi dell'hardware del PD32
- Familiarizzare con il set di istruzioni PD32
- Imparare a scrivere dei programmi in Assembly PD32
- Imparare a scrivere dei device driver in Assembly PD32

Risorse utili

- DISsimulator
- MicroOpGen



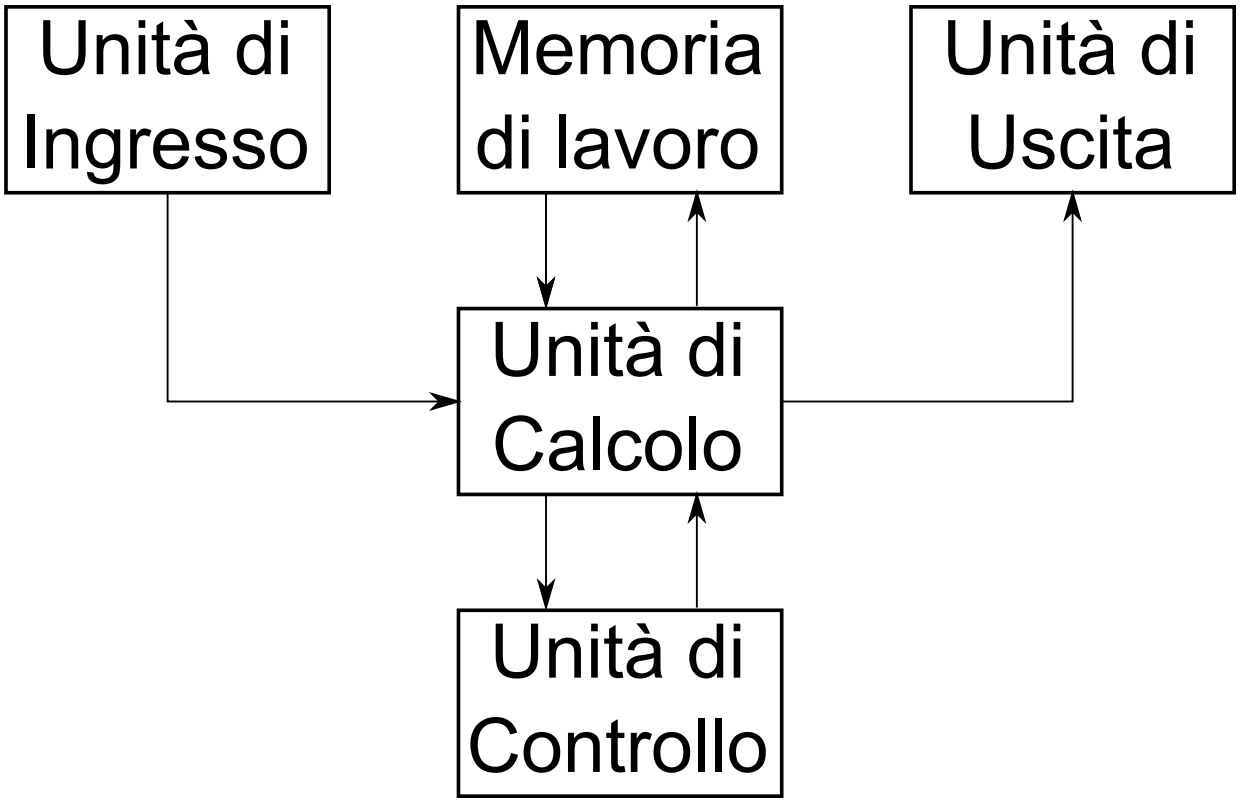


Architetture Hardware

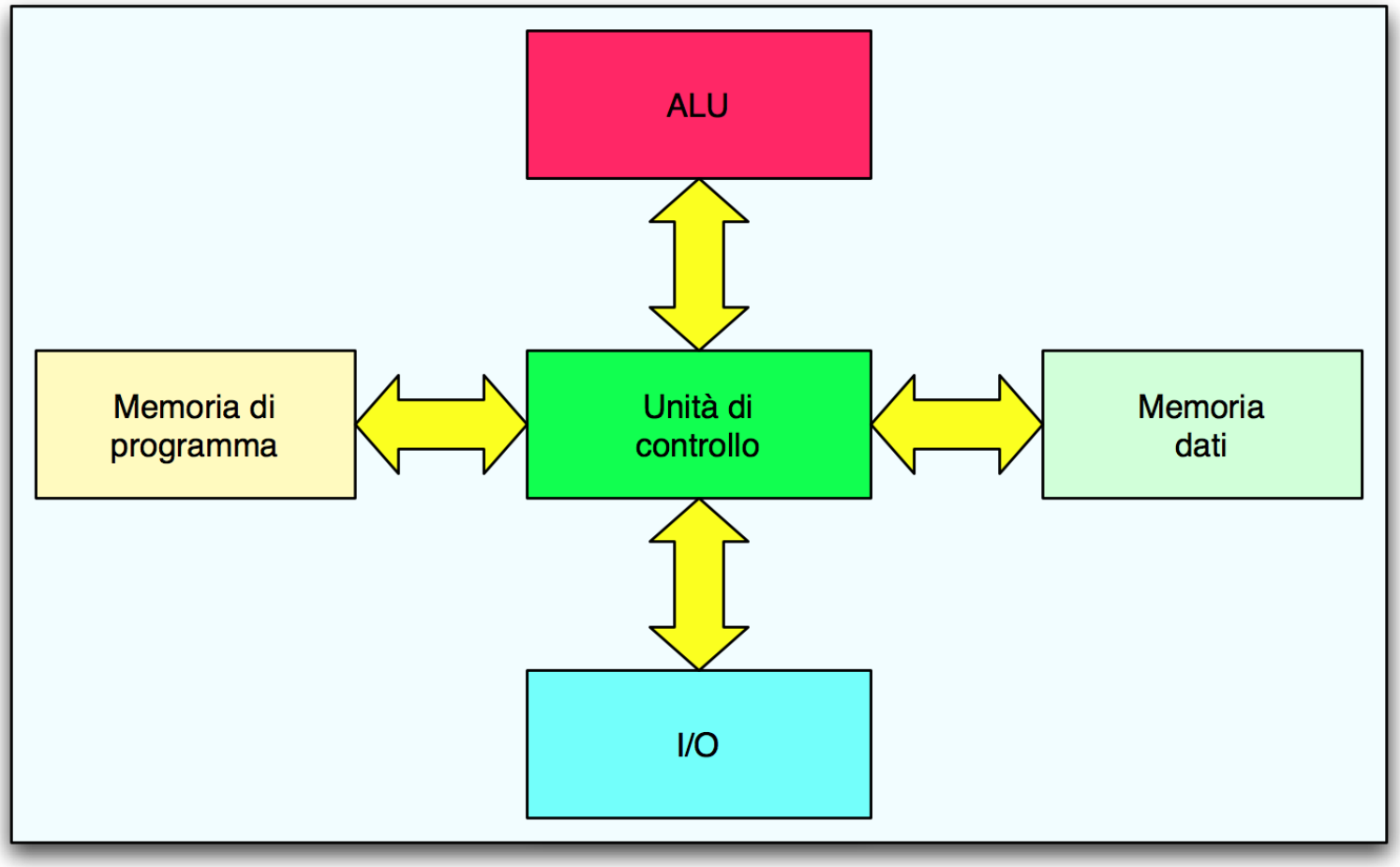
Un'architettura hardware descrive le relazioni tra le parti di un elaboratore elettronico.

- Ad esempio:
  - Come accede un processore alla memoria?
  - Come fa il processore a dialogare con le periferiche?
  - Come interagiscono dei processori tra loro?
- La prima "architettura" che compare nella storia si trova nella corrispondenza tra Charles Babbage ed Ada Lovelace, che descrive una macchina analitica
- Il più famoso modello di architettura è quello di John von Neumann, del 1945, per descrivere l'organizzazione logica degli elementi dell'EDVAC

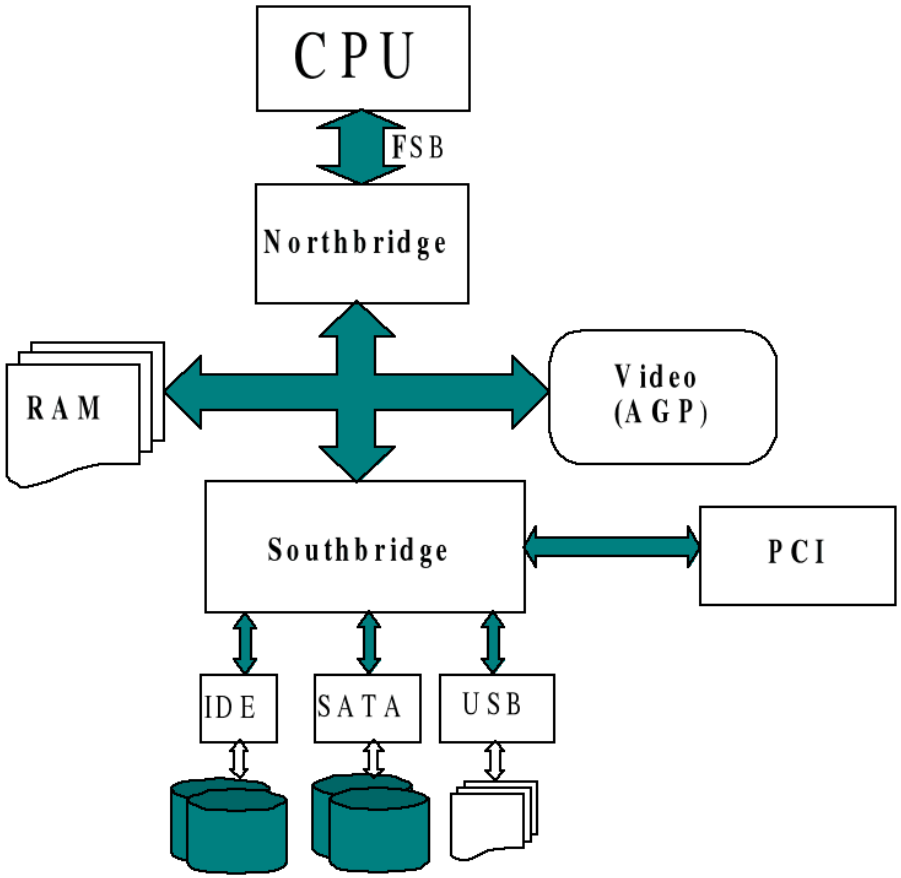
Architettura di von Neumann



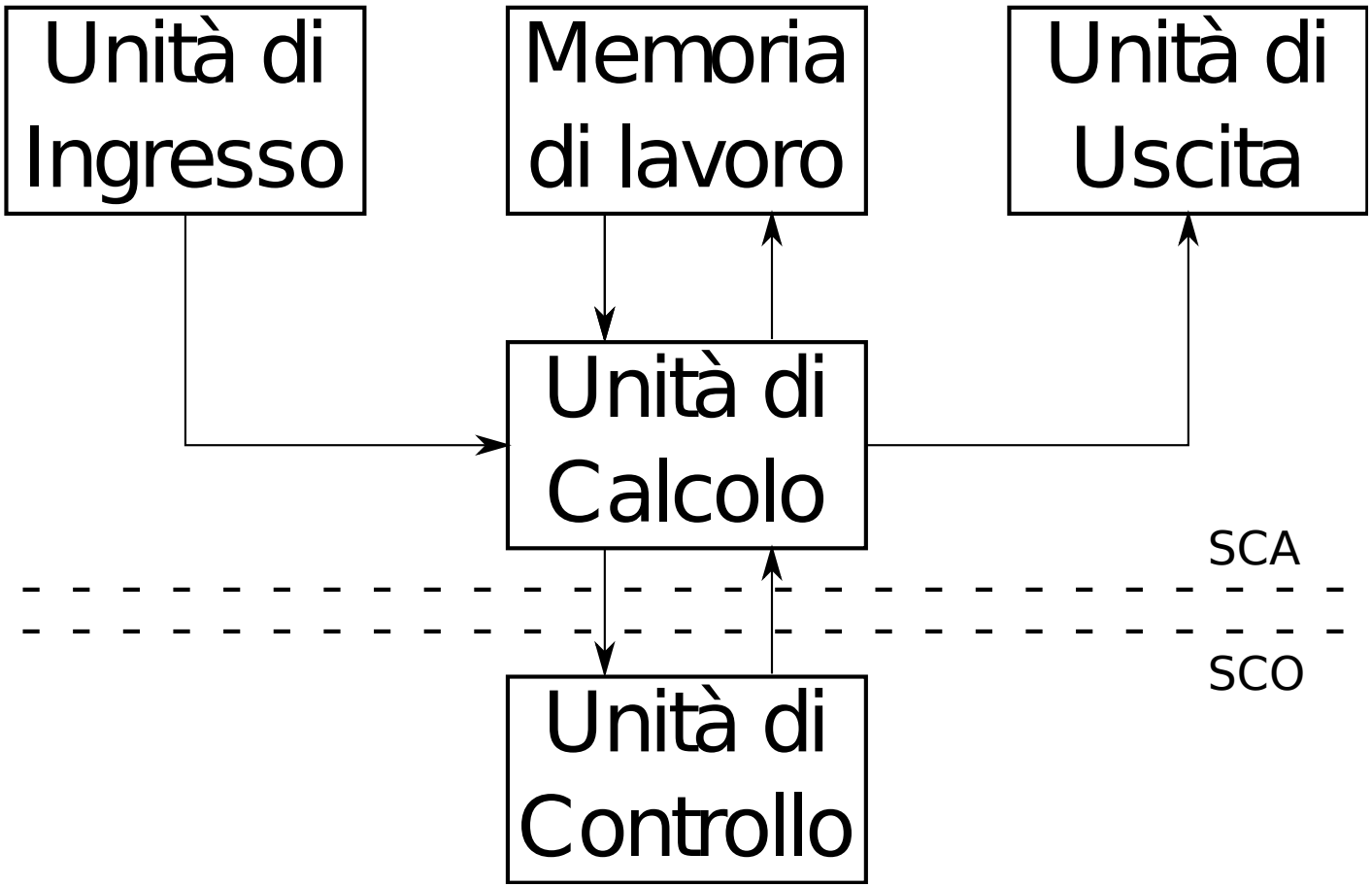
Architettura Harvard



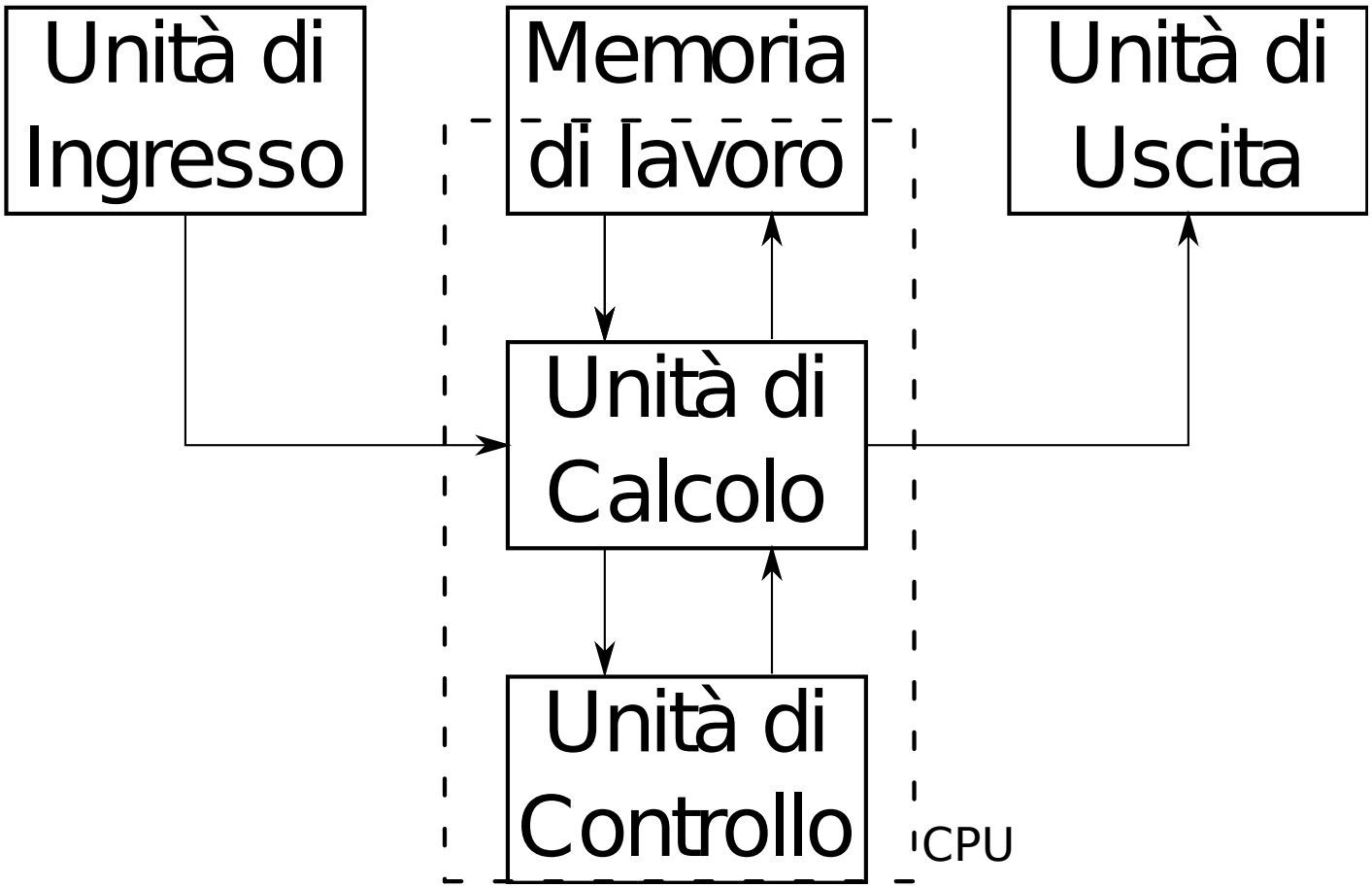
Architettura IBM



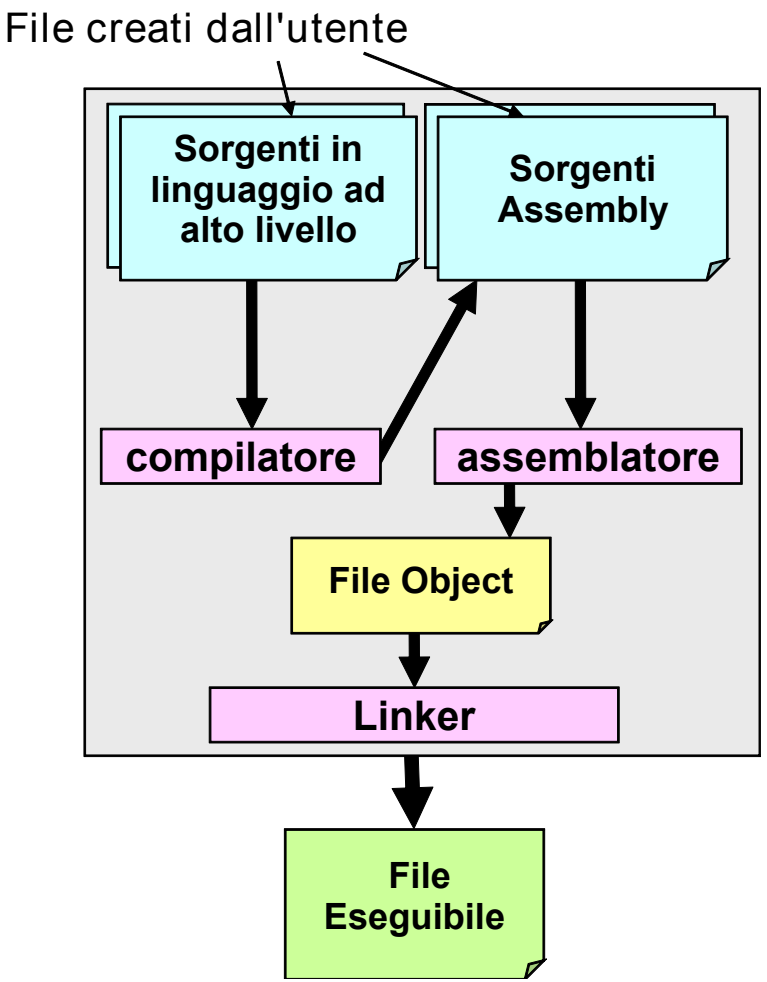
Architetture di von Neumann Rivisitata: SCA e SCO



Architetture di von Neumann Rivisitata: CPU



Generazione di un programma




```
a = b + c

movw b,R1
movw c,R2
addw R2,R1
movw R1,a

000101..0101001
1011101..010100
01011..11101010
010..1110101010
```

Generazione di un programma (2)

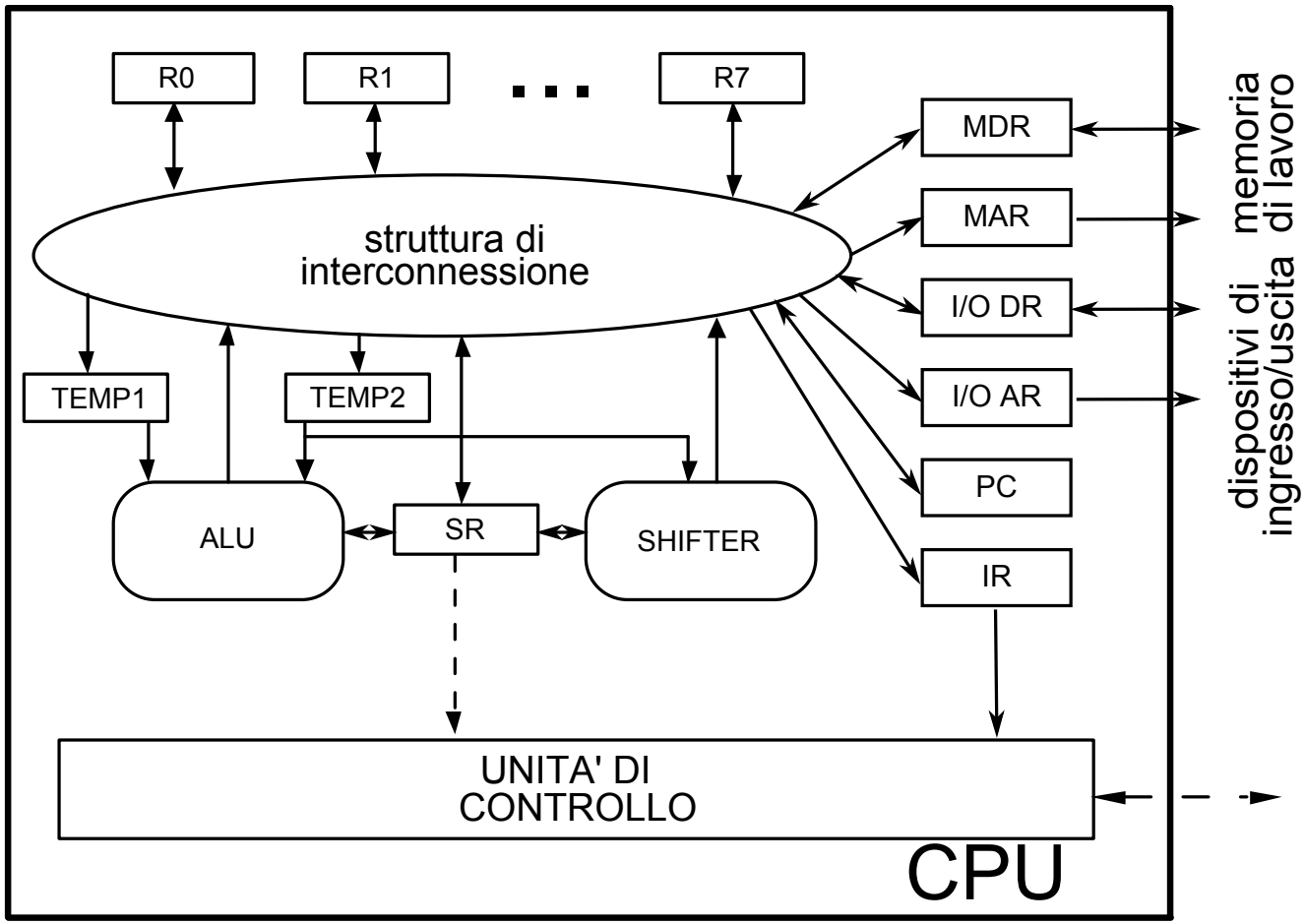
Il processo di generazione di un programma si snoda in tre fasi principali:

- **Compilazione**: Generazione automatica di codice Assembly da uno o più file scritti in linguaggi di alto livello
- **Assemblazione**: Generazione di uno o più file oggetto contenenti le rappresentazioni binarie delle istruzioni, a partire da codice Assembly
- **Collegamento**: Traduzione degli indirizzi e risoluzione di tutti i collegamenti tra *funzioni*, *variabili*, *etichette*.

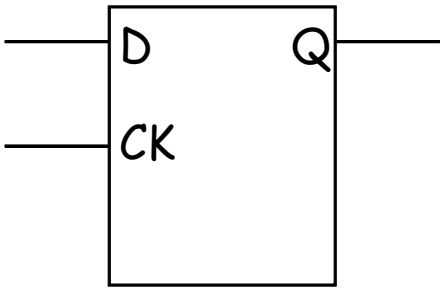
Il Processore Didattico a 32 bit

- È un processore "virtuale" dotato di registri a 32 bit
- Non esiste nella realtà, ma le sue funzionalità sono simulate tramite un programma
- Il set delle istruzioni è un sottoinsieme di quello dei processori Motorola
- Lo utilizzeremo per scoprire come funziona al suo interno un calcolatore

Il Processore Didattico a 32 bit

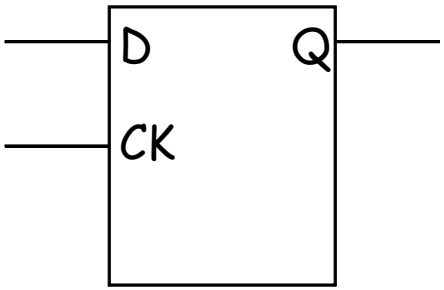


Memorizzare i dati nel processore

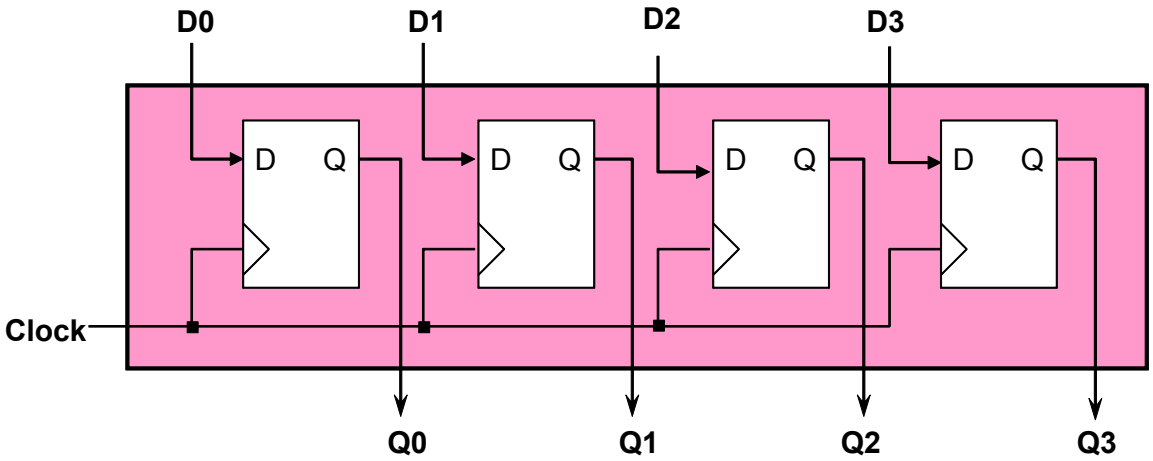


- Un flip-flop è un circuito basato su un bistabile
- Ha un ingresso dati, ed uno di abilitazione
- Usato come unità elementare di memorizzazione
- Combinandone più insieme, si possono costruire dei registri

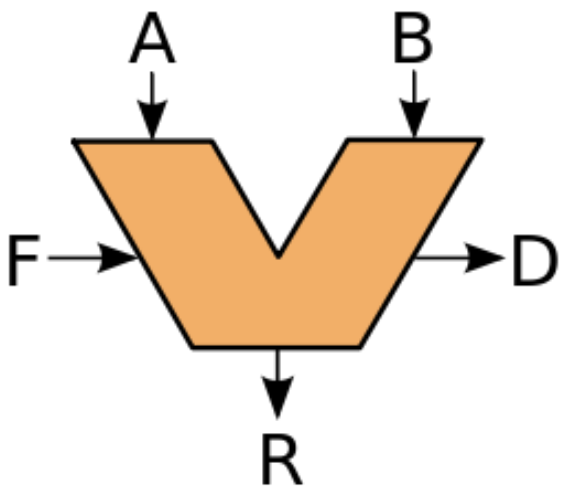
Memorizzare i dati nel processore



- Un flip-flop è un circuito basato su un bistabile
- Ha un ingresso dati, ed uno di abilitazione
- Usato come unità elementare di memorizzazione
- Combinandone più insieme, si possono costruire dei registri



Operazioni: la ALU

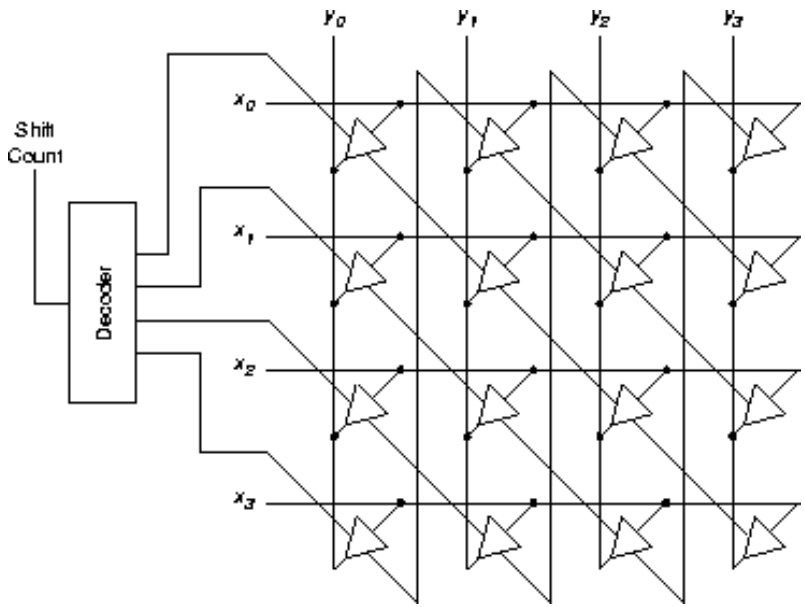


A, B: operandi
F: segnali dal SCO
D: segnali di stato
R: risultato

- L'Unità Logico-Aritmetica (ALU) è un componente fondamentale di ogni processore
- Essa implementa la maggior parte di operazioni artimetiche (somma, sottrazione, moltiplicazione, ...) e logiche (and, or, xor, ...)
- L'ALU recupera i dati dai registri del processore, processa i dati nell'accumulatore e provvede a salvare il risultato nel registro di uscita

Operazioni: lo Shifter

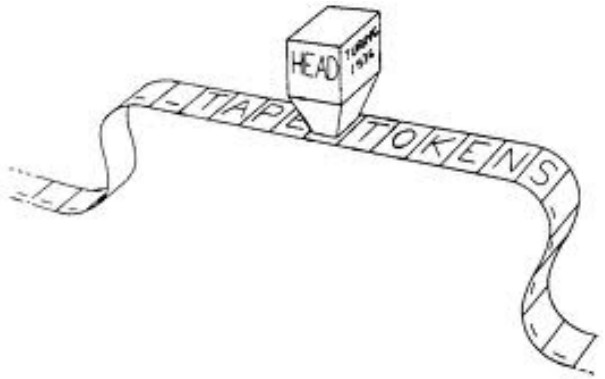
Uno shifter è un circuito digitale che può traslare i bit di una parola di un numero specificato di posizioni in un solo colpo di clock



PD32: Il Sottosistema di Calcolo (SCA)

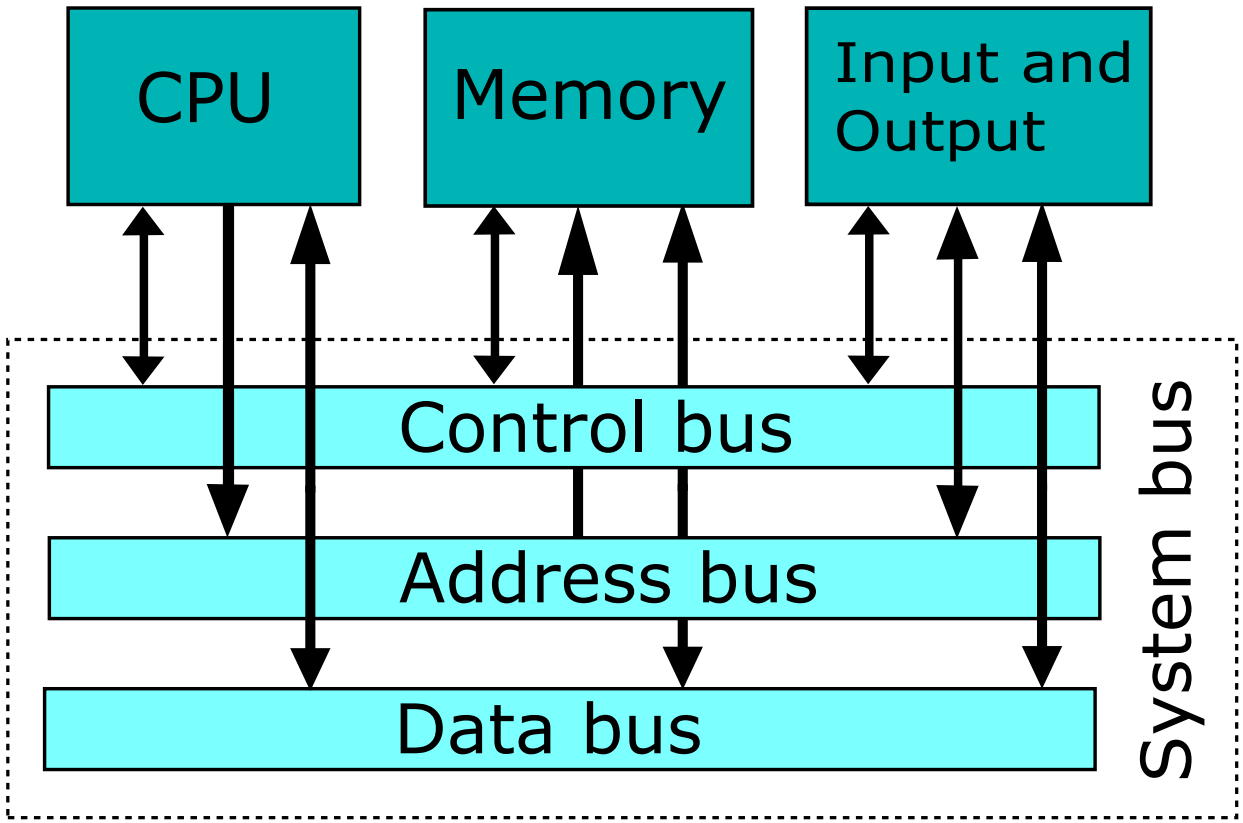
- Registri (basati su flip flop con segnali di enable)
  - Speciali
  - Generali
- Dispositivi di Calcolo
  - Shifter
  - ALU (somma e sottrazione)
- MUX
- Decodificatori
- Struttura di interconnessione: BUS

Modello di memoria



- La memoria può essere vista come un lunghissimo nastro, diviso in celle (*modello di memoria piatta*)
- Ogni cella di memoria viene identificata da un numero intero progressivo (*indirizzo*)
- Ciascuna cella ha dimensione pari ad un byte (8 bit)
- Virtualmente, una testina si muove sul nastro, per leggere o scrivere i dati nelle celle
- Una cella viene scritta necessariamente per intero

Trasferire dei dati: il BUS



Processamento delle istruzioni

Il processamento delle istruzioni (ciclo istruzione) prevede tre fasi:

- *Fetch*: l'istruzione viene prelevata dalla memoria
- *Decodifica*: viene identificato quale microprogramma del sottosistema di controllo (SCO) deve essere attivato
- *Esecuzione*: la *semantica* dell'istruzione viene effettivamente implementata. Può richiedere uno o più cicli macchina (se, ad es., si richiede l'interazione con la memoria o dispositivi di I/O).

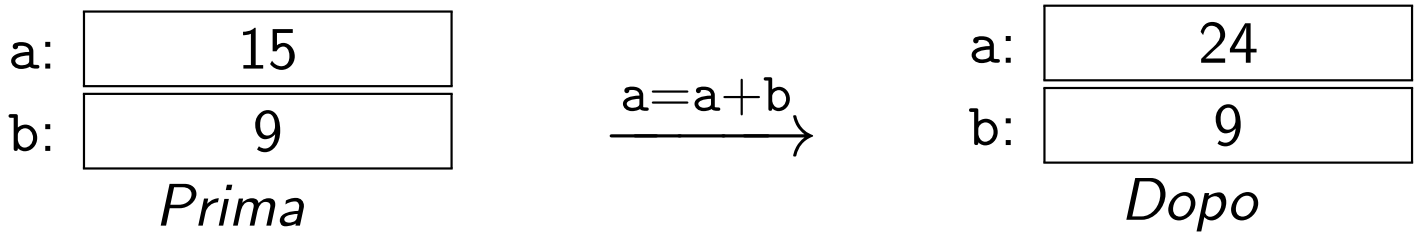
Un semplice esempio

Consideriamo l'istuzione `a=a+b` espressa in linguaggio di alto livello:

Somma

Memorizza nella variabile di nome `a` la somma dei valori contenuti nelle variabili di nome `a` e `b`

Nota: le variabili sono individuate da un nome simbolico deciso precedentemente nel programma



Un semplice esempio (2)

- Per eseguire questa istruzione è necessario:
  - Stabilire dove sono memorizzati i valori da sommare
  - Stabilire dove va scritto il risultato dell'operazione
  - Decidere quale operazione svolgere
- Nel PD32, gli operandi sono memorizzati nei registri interni alla CPU (registri visibili al programmatore)
- Il formato dell'istruzione è:

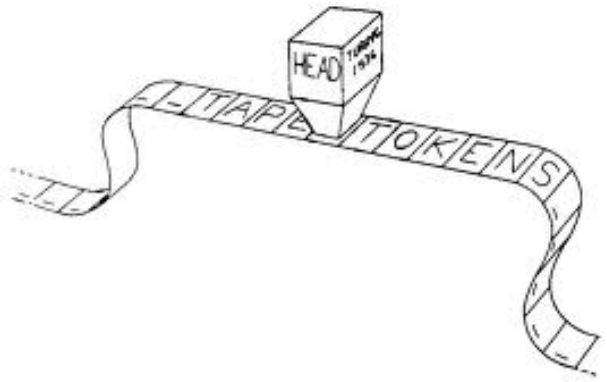
```
ADDs <sorgente> <destinazione>
```

- `s` può essere `B`, `W`, `L`
- Il campo destinazione è un registro che contiene il valore iniziale di un operando e sarà modificato:

```
ADDW R2, R1
```

(somma `R1` con `R2` (16 bit) e poni il risultato in `R1`)

Tenere traccia dell'esecuzione: il Program Counter

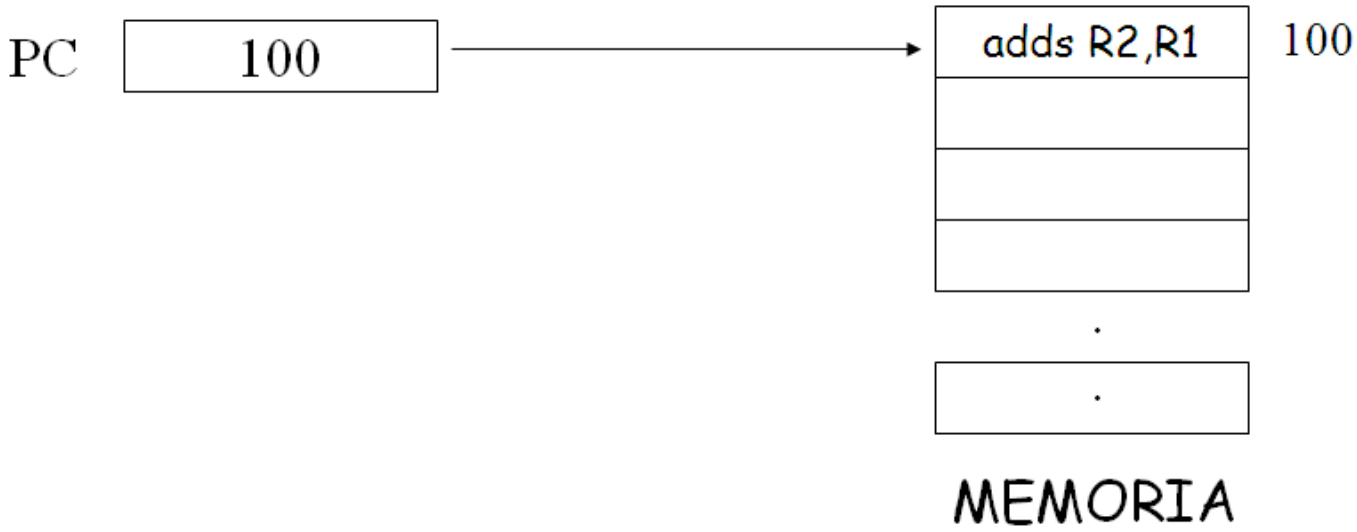


- Nell'architettura di von Neumann dati e programma si trovano nella stessa memoria
- Per conoscere l'indirizzo della prossima istruzione da eseguire, il processore utilizza un registro speciale (non visibile al programmatore): il *Program Counter* (PC)
- Ogni volta che un'istruzione viene eseguita, il PC viene aggiornato automaticamente per puntare a quella successiva
- Le istruzioni devono quindi essere contigue in memoria
- Il flusso d'esecuzione può non seguire il loro ordine in memoria (salti)

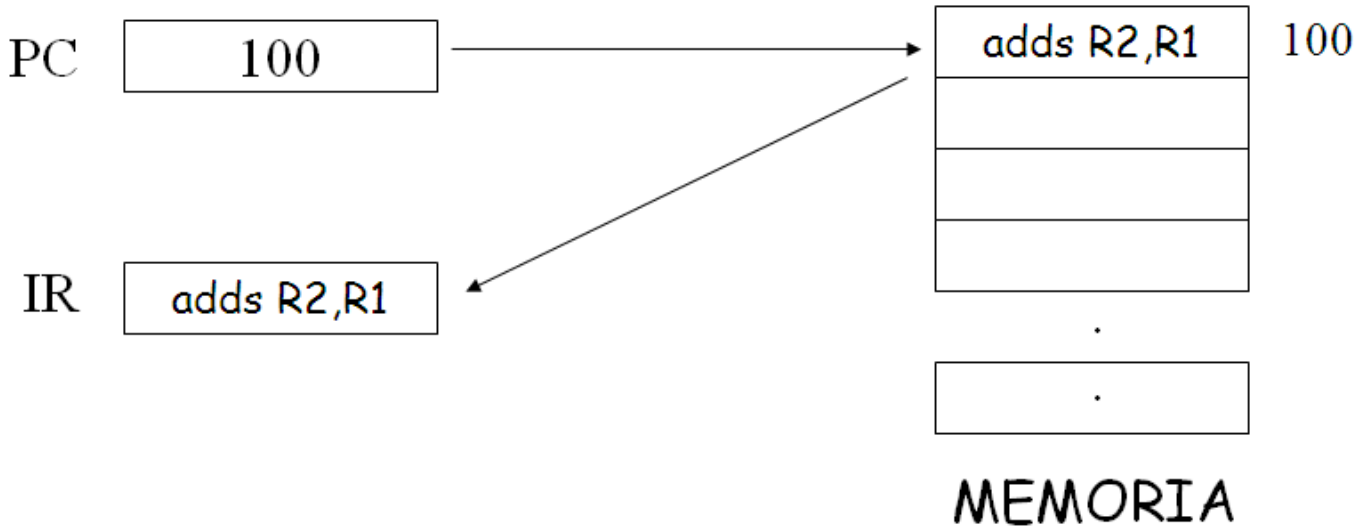
Supporto all'esecuzione di un'istruzione: l'Instruction Register

- Accedere alla memoria è un'operazione costosta
  - I registri dei processori sono realizzati con tecnologia molto più veloce, ma più costosta
  - La memoria, avendo dimensione più grande, è realizzata con tecnologie più economiche ma meno veloci
- L'esecuzione di un'istruzione è divisa in due fasi (decodifica, esecuzione)
- Accedere in memoria per eseguire ogni fase è troppo costoso
- Prima di essere eseguita, l'istruzione viene copiata (fase di fetch) in un registro speciale (non visibile al programmatore): l'*Instruction Register* (IR)

Esecuzione di un'istruzione

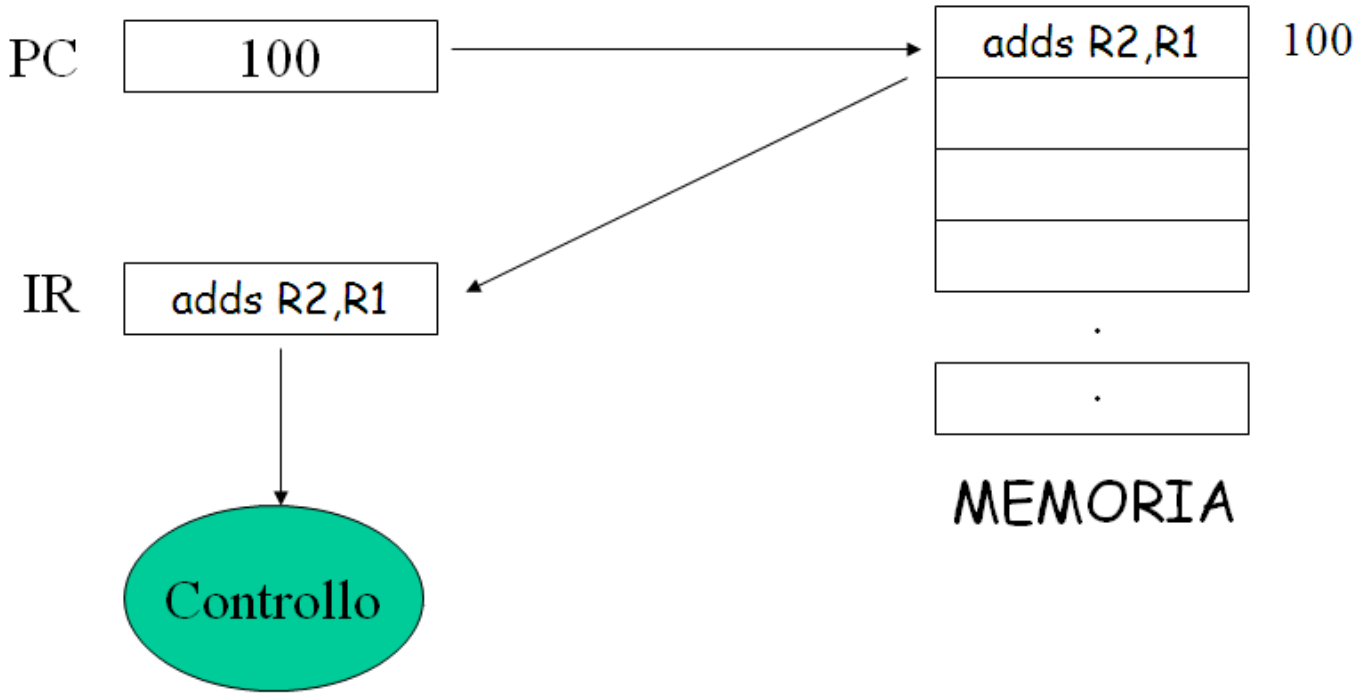


Esecuzione di un'istruzione: fase di fetch



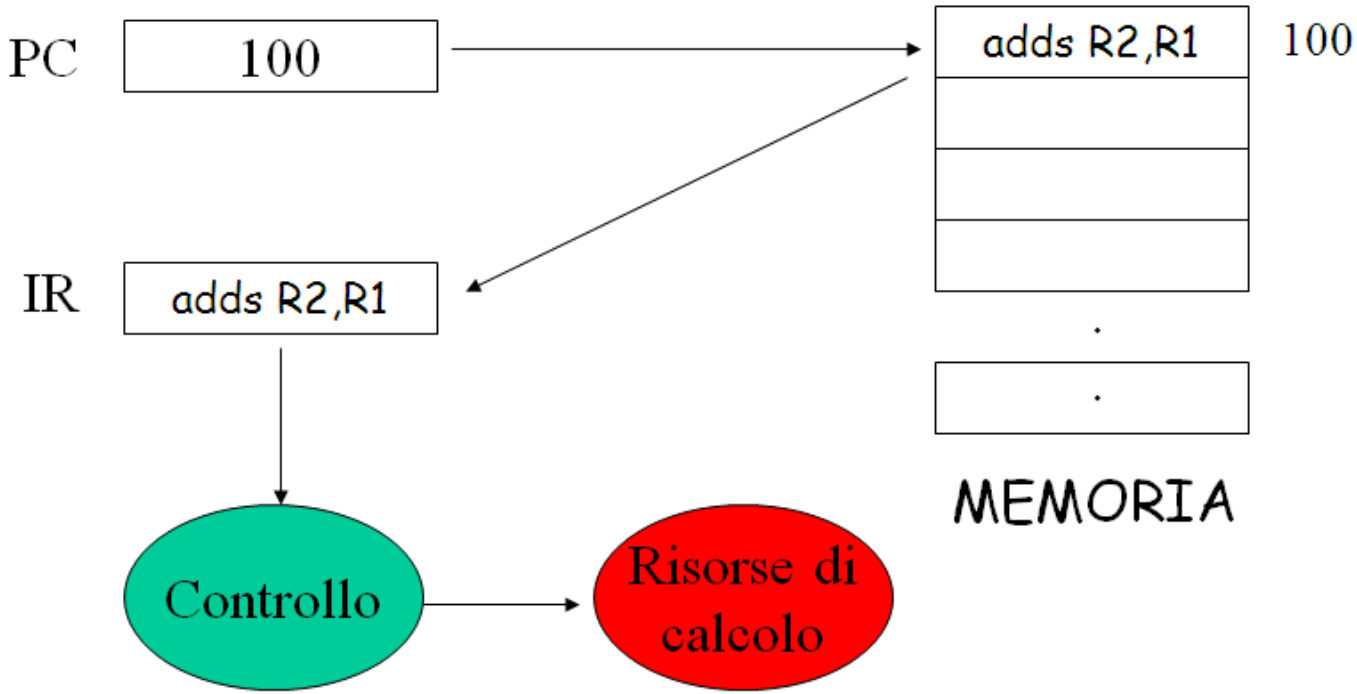
Durante la fase di fetch, l'istruzione viene copiata dalla memoria di lavoro nell'IR

Esecuzione di un'istruzione: fase di decodifica



Durante la fase di decodifica, l'istruzione viene interpretata per identificare il microprogramma che la possa eseguire

Esecuzione di un'istruzione: fase di esecuzione



Il SCO pilota il SCA per implementare la semantica dell'istruzione

CPU come interprete

- L'esecuzione di un programma può essere vista come la ripetizione dei seguenti passi che interpretano ed eseguono le istruzioni del programma contenute in memoria.

Ciclo Istruzione

fetch: `(PC)` $\rightarrow$ `IR`
incrementa `PC`
esegui istruzione in `IR`
vai al passo fetch

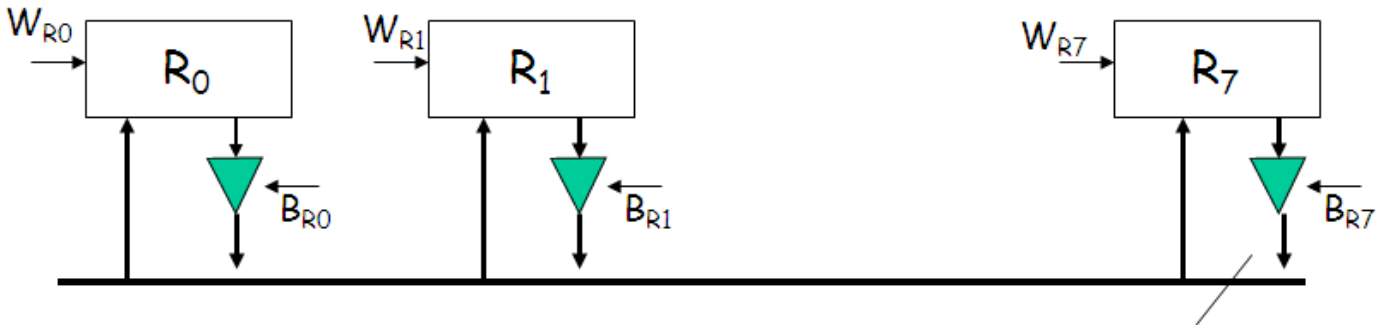
- La CPU interpreta le istruzioni che sono presenti nel suo IR
- L'esecuzione di un'istruzione può modificare (indirettamente) il contenuto di PC
- Tale schema è semplificato: per interagire con l'esterno, o per gestire situazioni anomale, tale ciclo deve poter essere interrotto

PD32: BUS interno

- Usato per il collegamento dei registri interni
- Operazioni che caratterizzano il BUS:
  - *Ricezione dati*: i bit presenti sul bus sono memorizzati in un registro
  - *Trasmissione dati*: il contenuto di un regsitro è posto sul bus
- Al più un solo registro può scrivere sul bus
  - Utilizzo di segnali di controllo opportunamente generati:
    - *Write Enable*
    - *Buffer Three-state*

PD32: BUS interno e segnali di controllo

Si può avere una sola scrittura per volta sul bus (controllo mediante B_i o W_i), pertanto con n registri, si avranno $2n$ segnali di controllo:



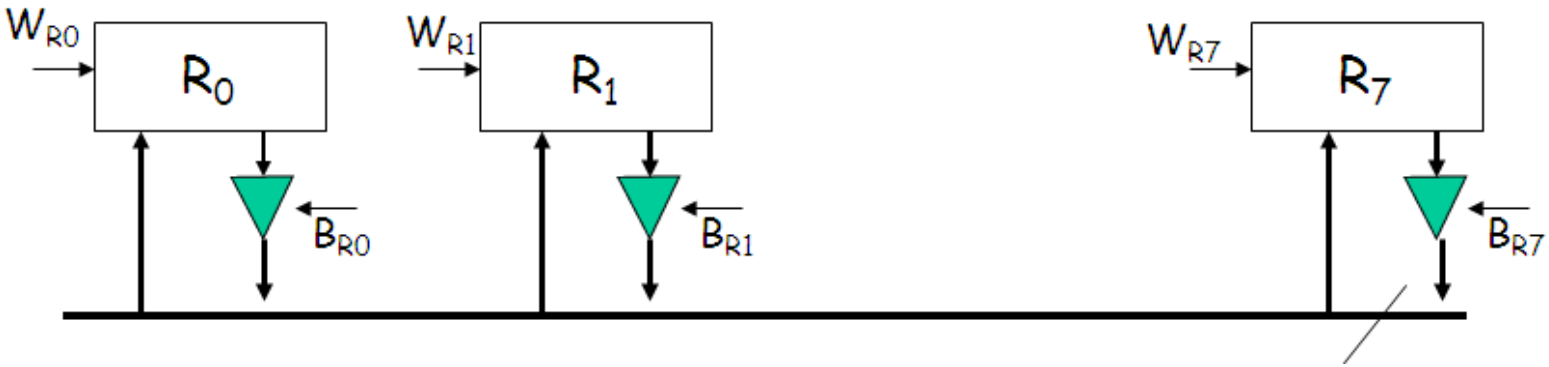
$W_i = 1$: si può leggere dal bus
$R_i = 1$: si può scrivere sul bus

PD32: Interazione con la memoria

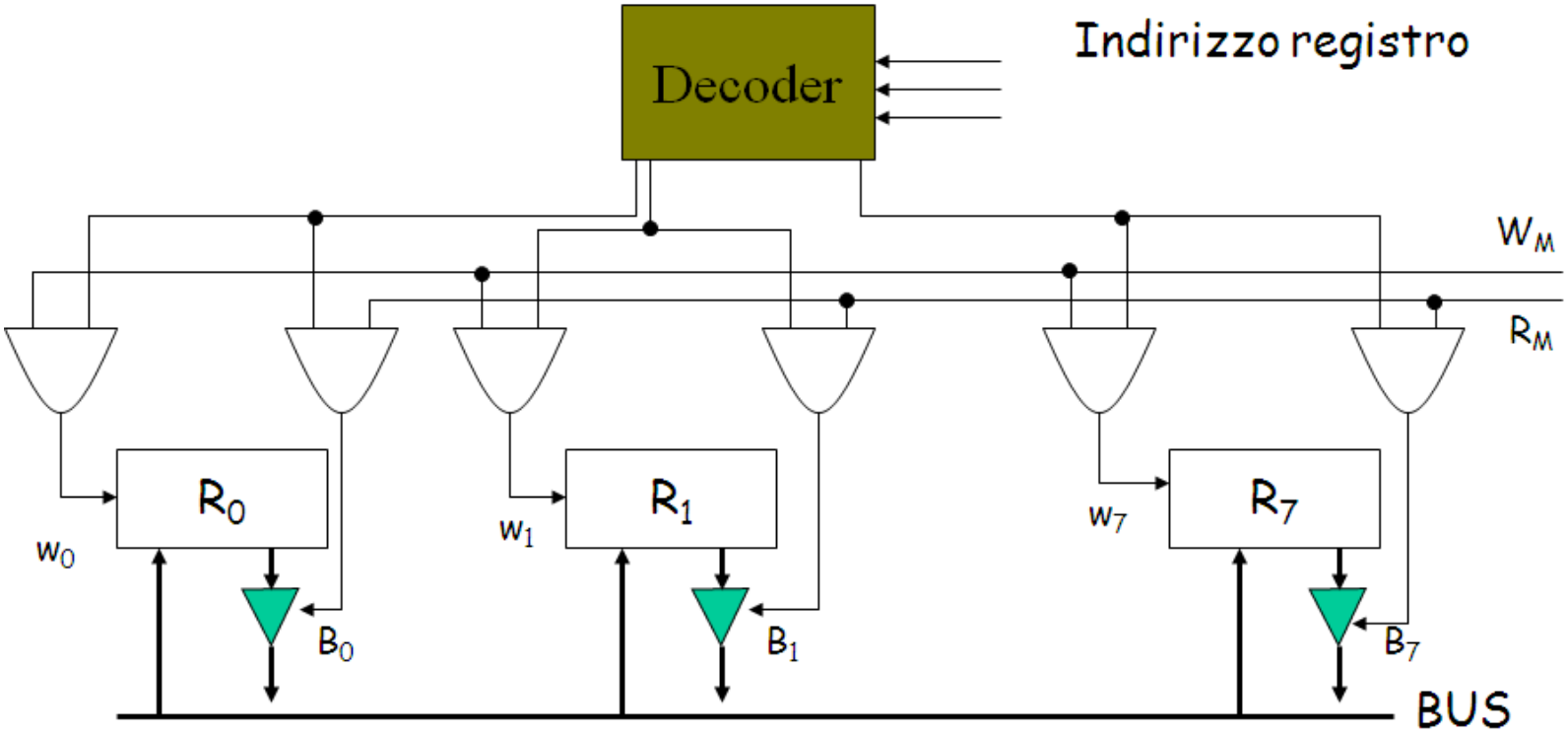
- La memoria contiene sia i dati che le istruzioni (architettura di von Neumann)
- Può essere sia scritta che letta
- È necessario quindi:
  - Prelevare istruzioni
  - Leggere dati
  - Scrivere dati
- È necessario inoltre instradare opportunamente i dati ricevuti dalla memoria verso i registri e viceversa

PD32: Banco dei registri

- Insieme di 8 registri generali (visibili dal programmatore), indicati da `R0` ad `R7`
- Sono controllati mediante segnali di abilitazione per:
  - Scrittura del registro (W_i)
  - Lettura e consegna sul bus interno del contenuto del regsitro (R_i)

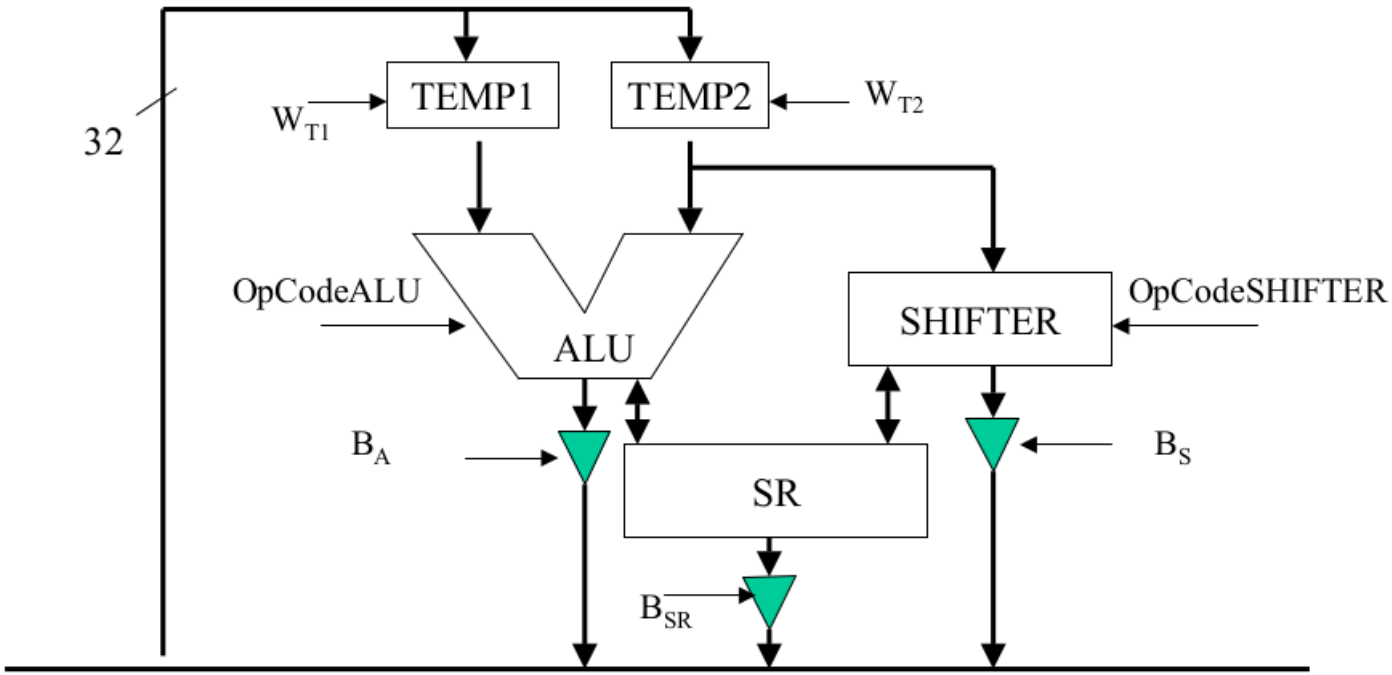


PD32: Banco dei registri (2)

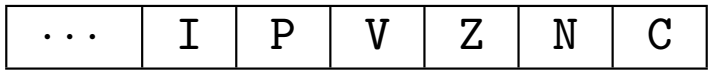


Lo Status Register (SR)

- Il registro SR è un registro speciale (non scrivibile direttamente dal programmatore)
- Contiene informazioni sull'esito dell'ultima operazione (es: zero, overflow)
- Usato anche come ingresso per alcune operazioni (es: salti condizionati)



Lo Status Register (SR)

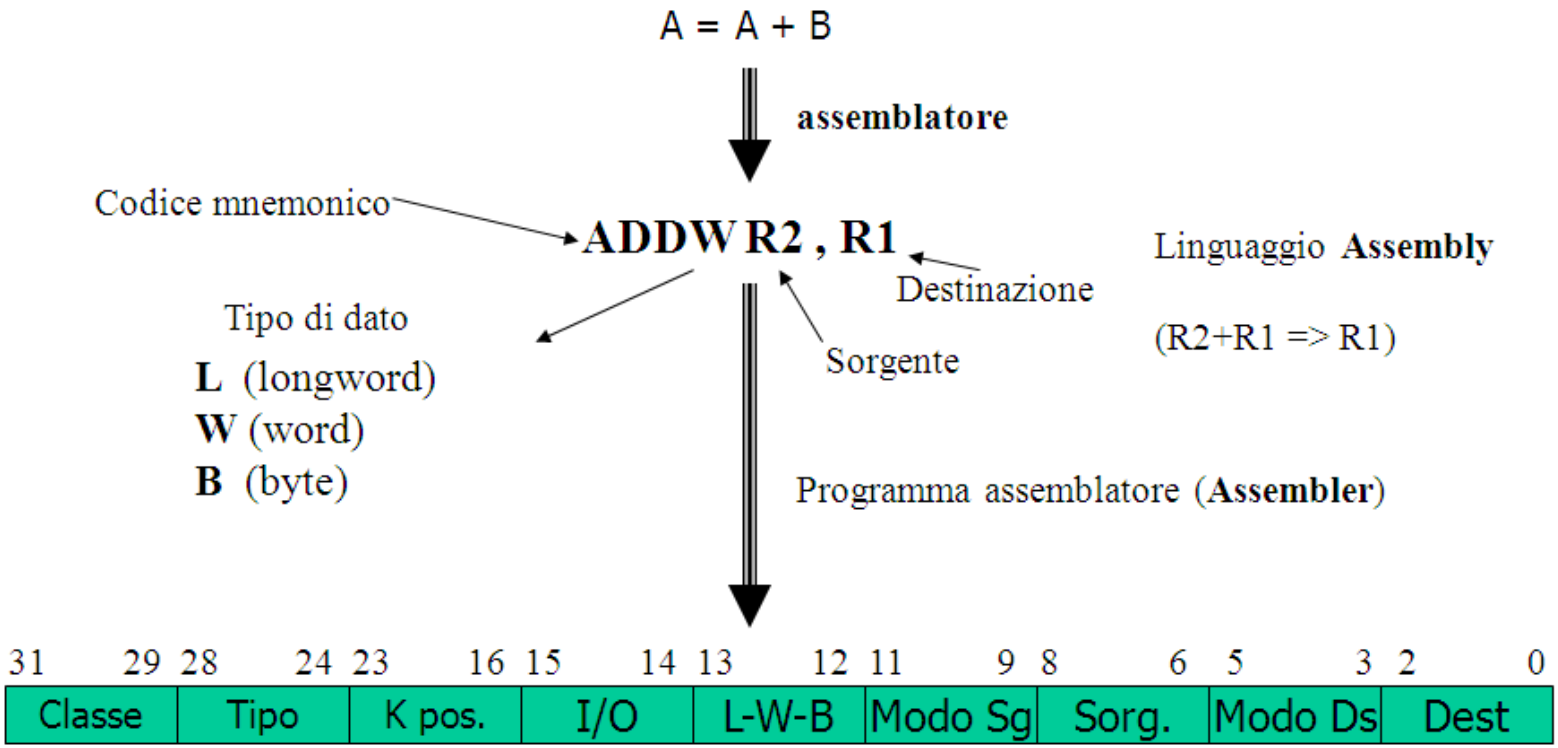


- **carry** (C): impostato a 1 se l'ultima operazione ha prodotto un riporto (addizione) o un prestito (sottrazione)
- **negative** (N): impostato a 1 se l'ultima operazione ha prodotto un risultato negativo (copia del bit più significativo)
- **zero** (Z): impostato a 1 se l'ultima operazione ha come risultato 0
- **overflow** (V): impostato a 1 se il risultato dell'ultima operazione supera la capacità di rappresentazione a 32 bit
- **parity** (P): impostato a 1 se nel risultato dell'ultima operazione c'è un numero pari di 1
- **interrupt enable** (I): indica che c'è la possibilità di interrompere ($I = 1$) o meno ($I = 0$) l'esecuzione del programma in corso

Rappresentare delle operazioni: le Istruzioni

- Sono organizzate in otto classi:
  1. Spostamento di dati
  2. Aritmetiche (somma e sottrazione) su interi
  3. Operazioni di tipo logico
  4. Rotazione e shift
  5. Operazioni sui bit di stato
  6. Controllo del flusso d'esecuzione del programma
  7. Controllo della macchina
  8. Ingresso/Uscita di dati

Rappresentare delle operazioni: le Istruzioni (2)

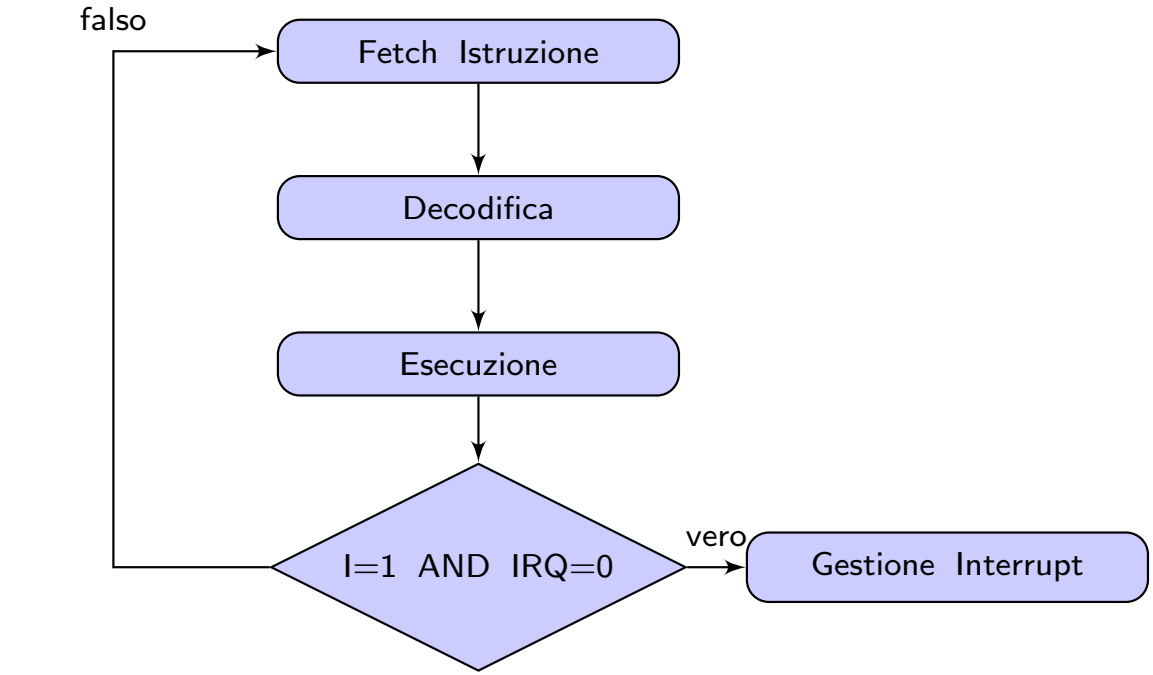


Nel formato istruzioni del PD32, ogni istruzione si rappresenta con 32 bit

Rappresentare delle operazioni: le Istruzioni (3)

Campo	**N° bit**	**Descrizione**
Classe	3	Indica la classe dell'istruzione
Tipo	5	L'operazione all'interno della classe
K Pos.	8	Utilizzato nelle istruzioni di shift e I/O
I/O	2	Codifica come recuperare l'indirizzo del device
L-W-B	2	Indica il formato del dato trattato dall'operazione
Modo Sg	3	Indica il modo di indirizzamento della sorgente
Sorg	3	Uno degli 8 registri generali `R0` – `R7`
Modo Ds	3	Indica il modo di indirizzamento della destinazione
Dest	3	Uno degli 8 registri generali `R0` – `R7`

Ciclo istruzione



Ciclo istruzione (2)

Ciclo istruzione del microprocessore:

Passo	Operazione
1	`PC` $\rightarrow$ `MAR`
2	`(MAR)` $\rightarrow$ `MDR`
3	`MDR` $\rightarrow$ `IR`; `PC` $+ 4 \rightarrow$ `PC`
4	Decodifica istruzione (*Decode*)
5	Esecuzione istruzoine (*Execute*)
6	Torna al passo 1

- Il tempo (in *nsec*) necessario ad eseguire l'intera sequenza delle operazioni costituisce il *ciclo di istruzione della CPU*
- La CPU esegue tale ciclo continuamente, finché non carica nel registro `IR` (ed esegue) l'istruzione `HALT`
- Un'eccezione a questa regola (*interruzione*) si verifica quando un dispositivo esterno chiede di trasferire dati dalla/alla CPU

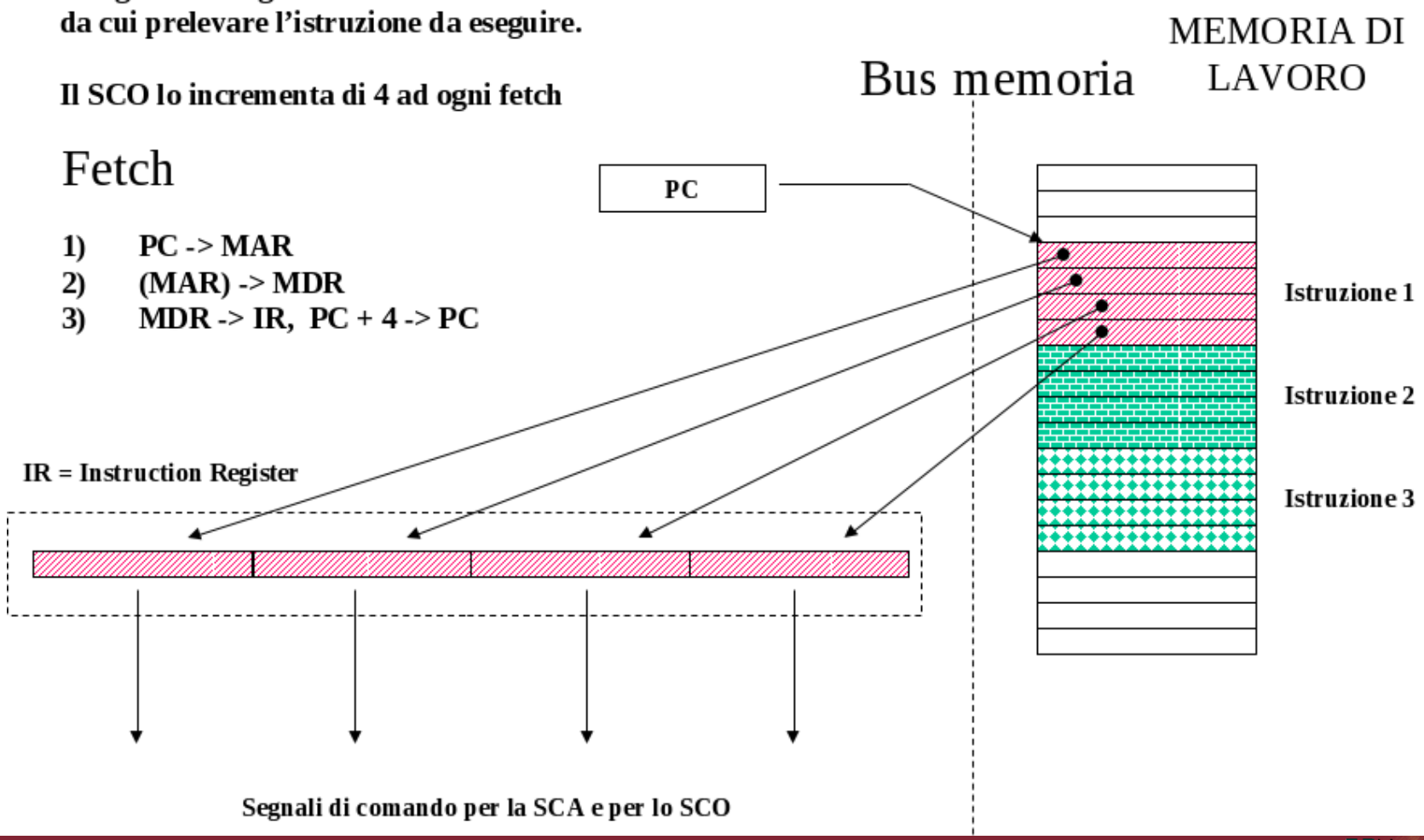
Ciclo istruzione: Fetch

Il registro "Program Counter" contiene l'indirizzo da cui prelevare l'istruzione da eseguire.

Il SCO lo incrementa di 4 ad ogni fetch

Fetch

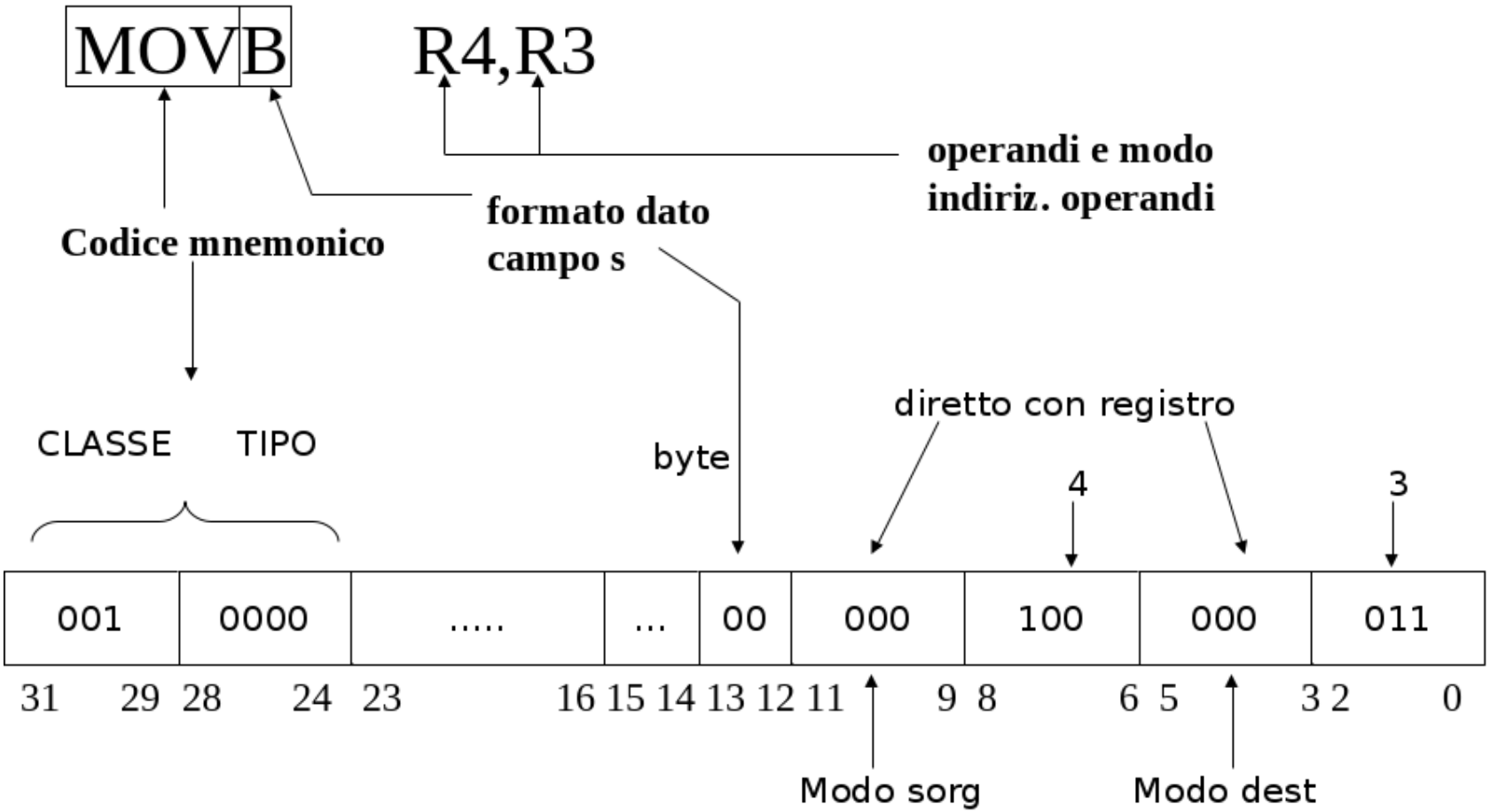
1) PC -> MAR
2) (MAR) -> MDR
3) MDR -> IR, PC + 4 -> PC



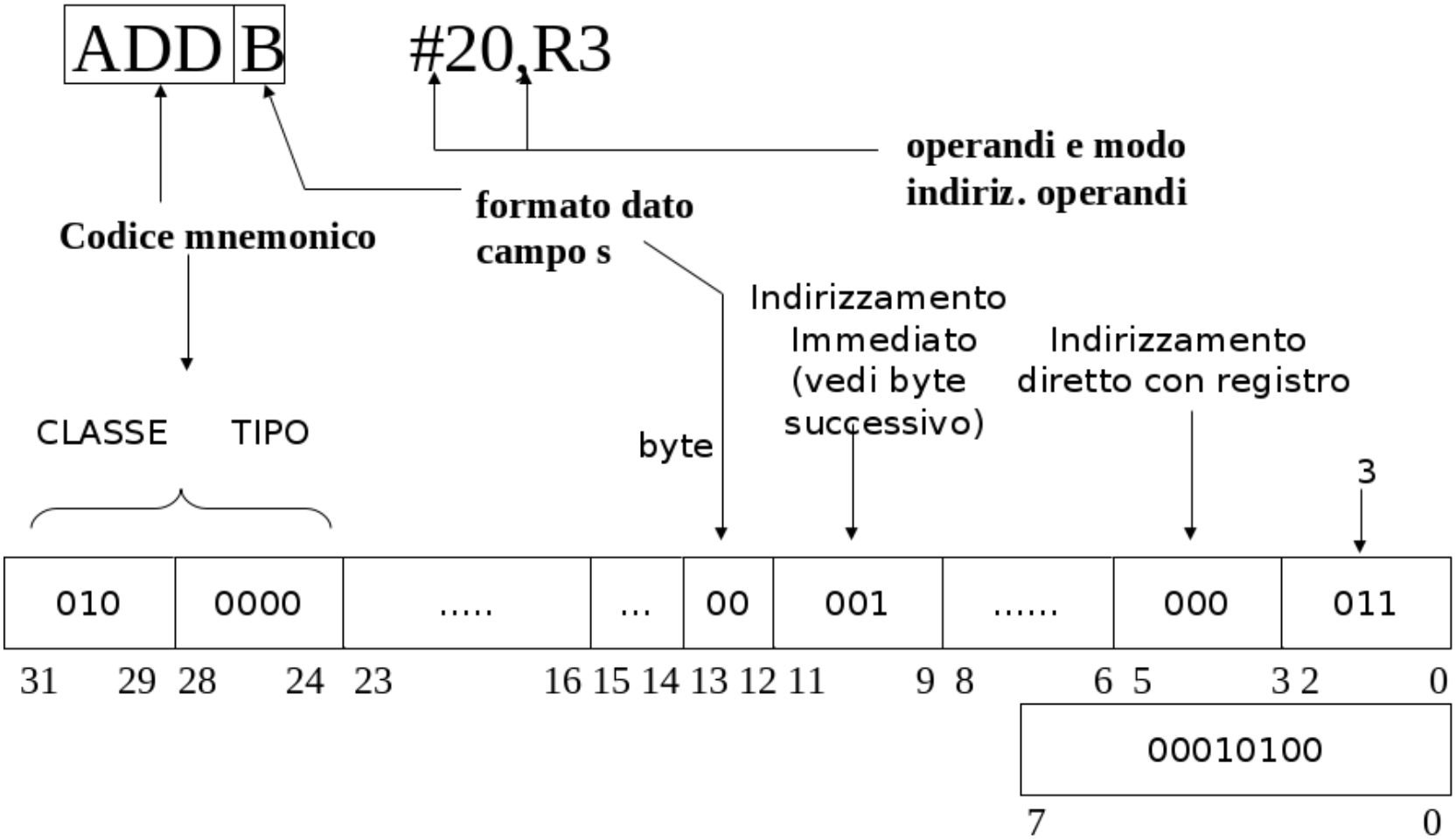
Alcune istruzioni Assembly

- `MOVB R1, R2`: copia il contenuto del primo byte di `R1` in `R2`
- `MOVW R1, (R2)`: copia il contenuto dei primi 2 byte di `R1` nei due byte di memoria il cui indirizzo iniziale è memorizzato in `R2`
- `MOVL (R1) ,R2`: copia in `R2` il contenuto dei 4 byte di memoria il cui indirizzo è specificato in `R1`
- `SUBs R1,R2`: sottrai il contenuto del primo, dei primi 2 o i 4 bytes del registro `R1` con il corrispondente in `R2`, il risultato memorizzalo in `R2`
- `ADDs #d,R2`: addiziona al contenuto del registro `R2` la quantità `d` di dimensione `s`

Traduzione istruzioni Assembly in linguaggio macchina



Traduzione istruzioni Assembly in linguaggio macchina



Modalità di Indirizzamento

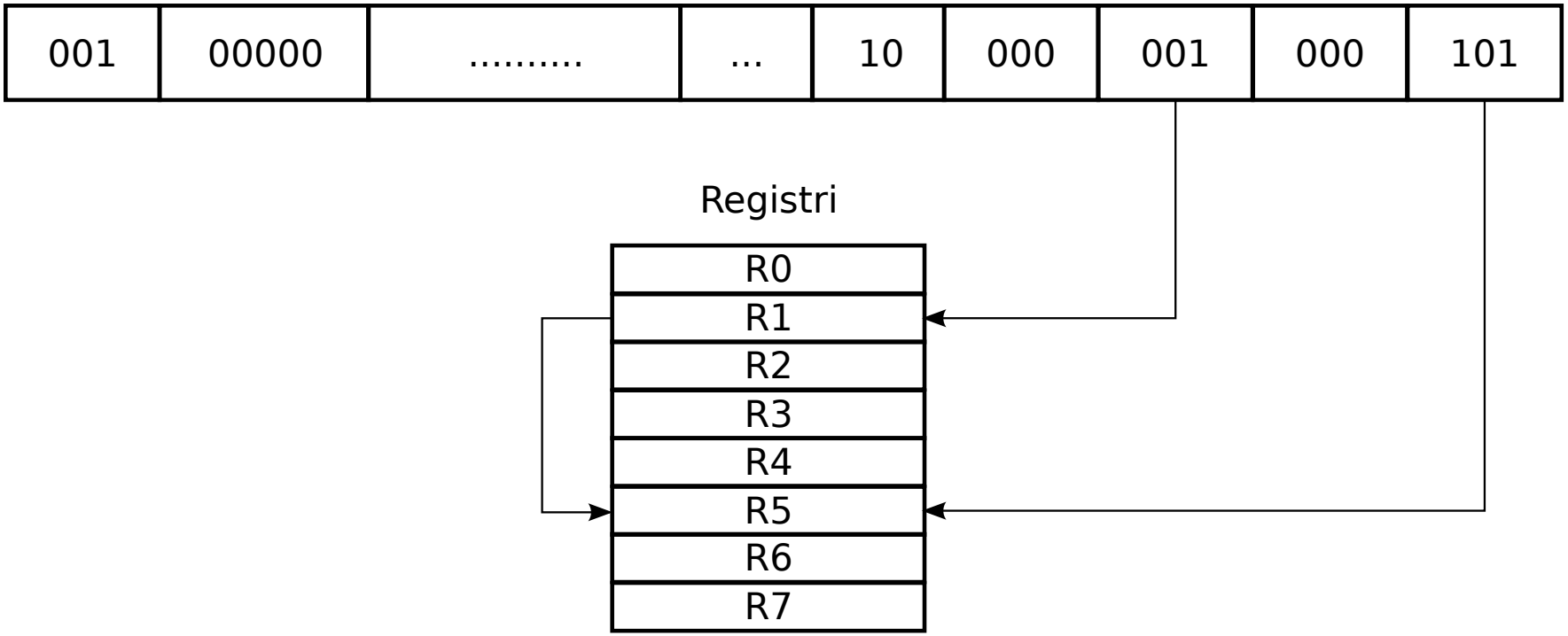
- L'indirizzamento consiste nell'identificazione della posizione degli operandi
  - Possono trovarsi nei registri (`R0` – `R7`)
  - In memoria di lavoro (la posizione è stabilita dall'indirizzo di memoria in cui è memorizzato
- Chiamiamo la posizione di un operando Effective Address (EA)
  - EA può essere pertanto un registro o una locazione di memoria
- Il valore di EA deve essere noto al tempo di esecuzione del programma (run time), può però non essere noto al momento della sua scrittura (compile time). Ciò consente di ottenere una grande flessibilità

Modalità di Indirizzamento (2)

- Modi diretti
  - Diretto con registro
  - Immediato
  - Assoluto
- Modi indiretti
  - Indiretto con registro
  - Indiretto con spiazzamento
  - Relativo
  - Indiretto con predecremento
  - Indiretto con postincremento

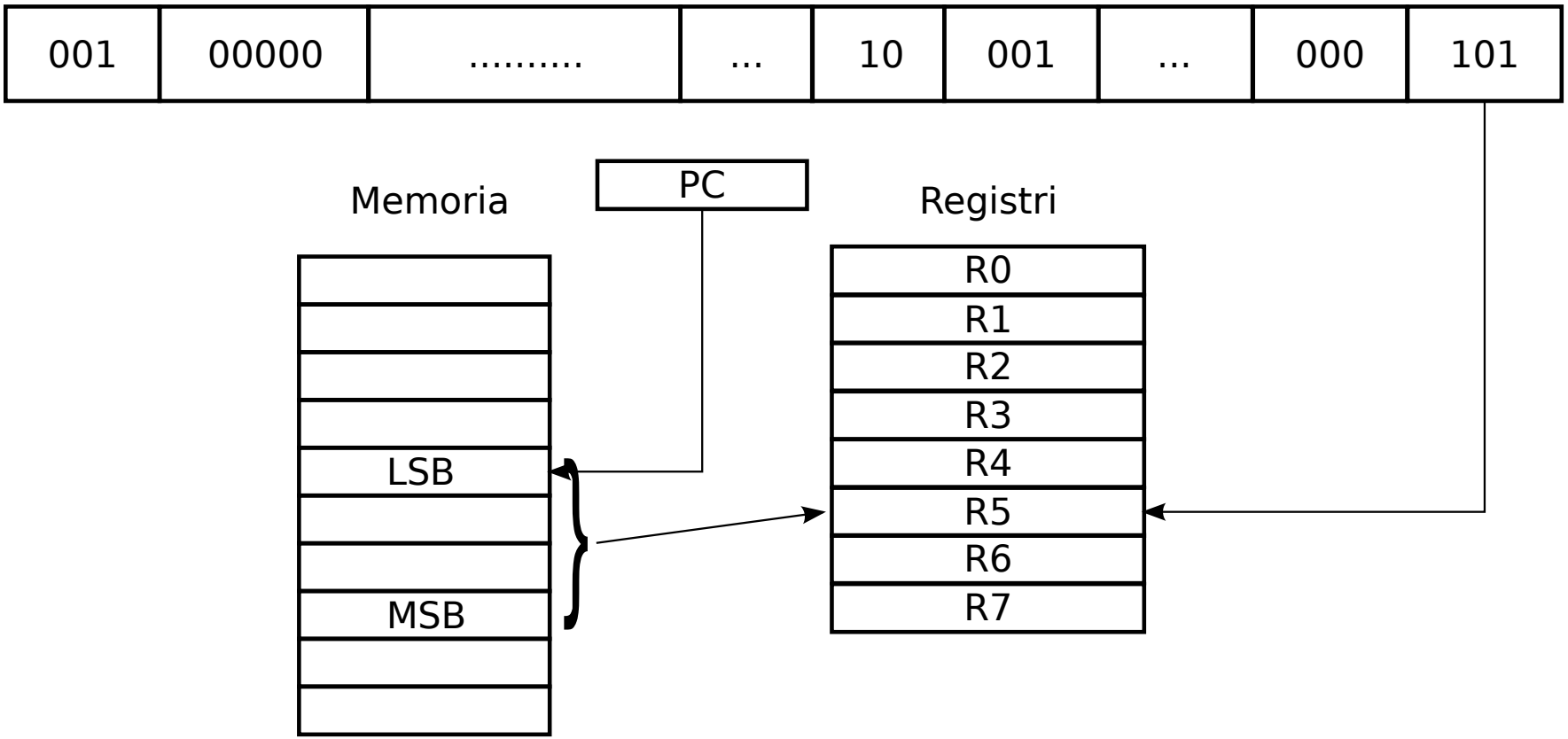
Indirizzamento: a registro

Esempio: `MOVL R1, R5`:



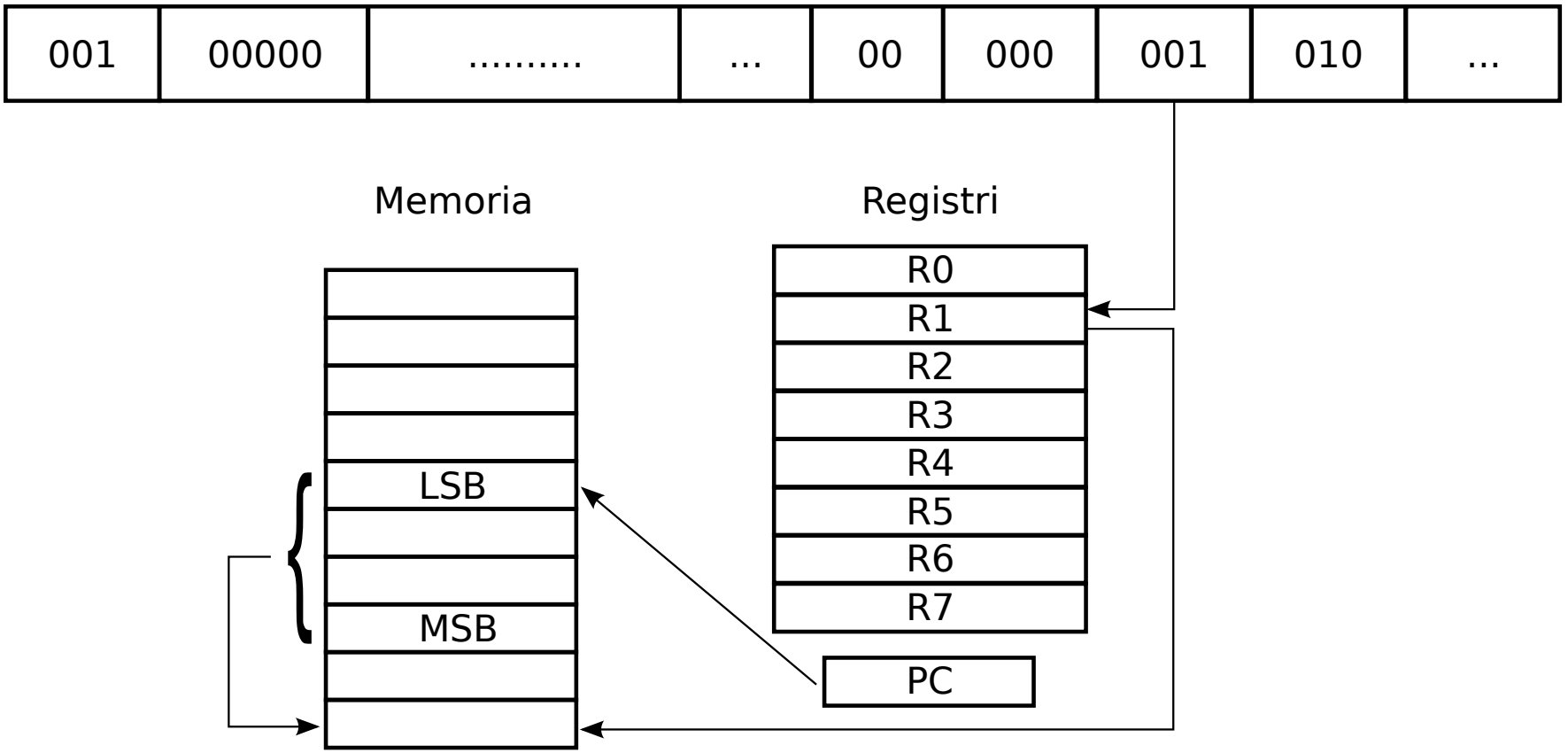
Indirizzamento: immediato

Esempio: `MOVL #0, R5`:



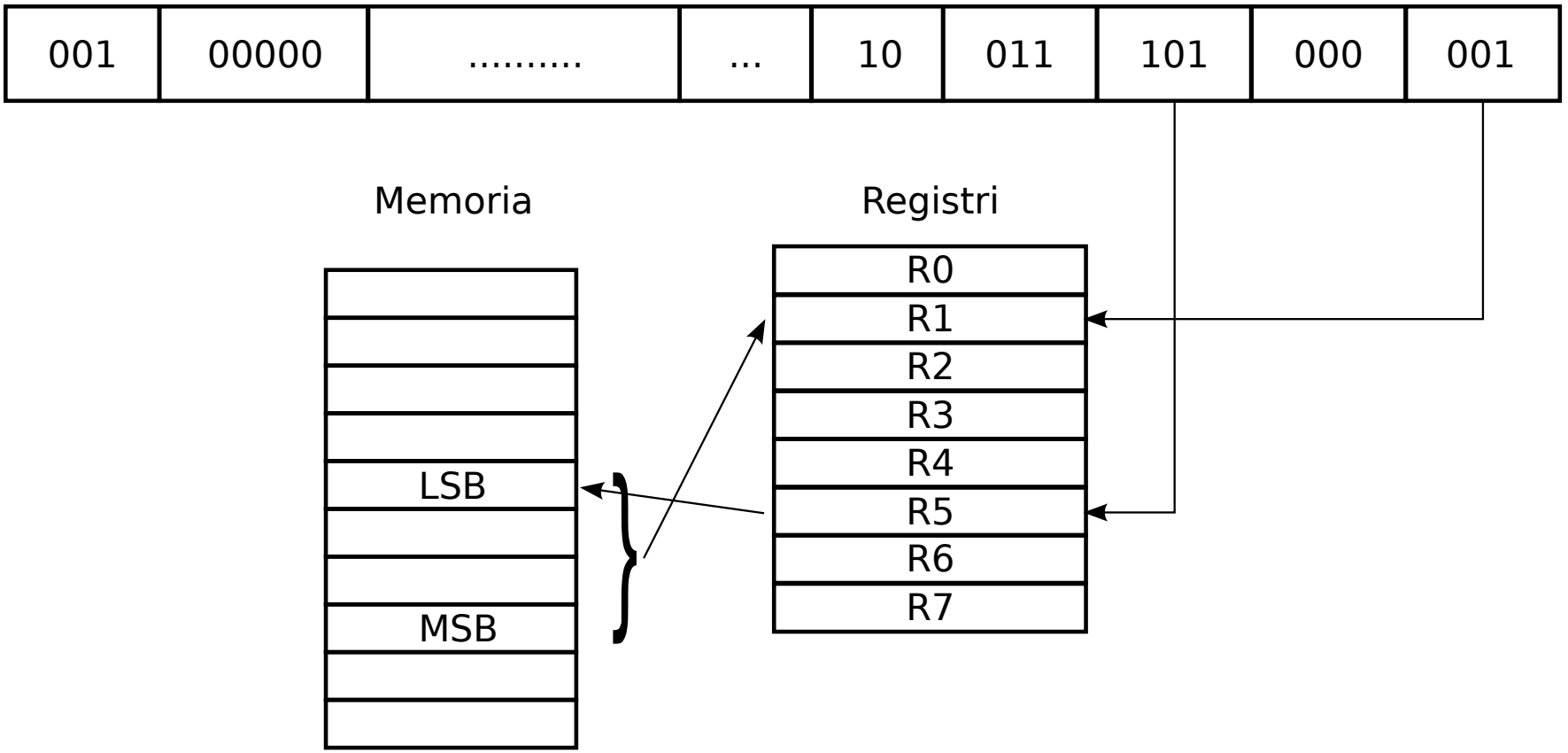
Indirizzamento: assoluto

Esempio: `MOVB R1, 1280h`:



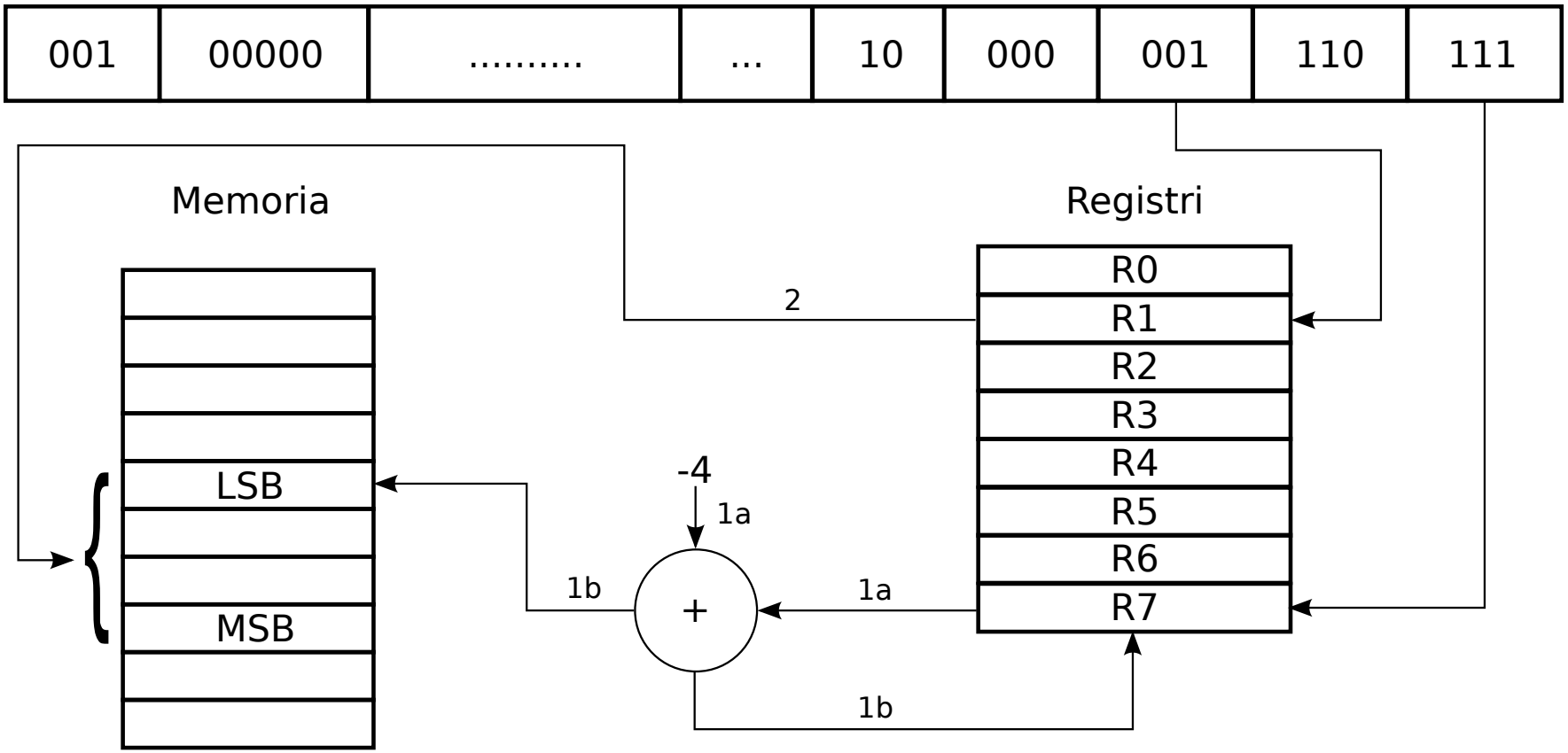
Indirizzamento: indiretto con registro

Esempio: `MOVL (R5), R1`:



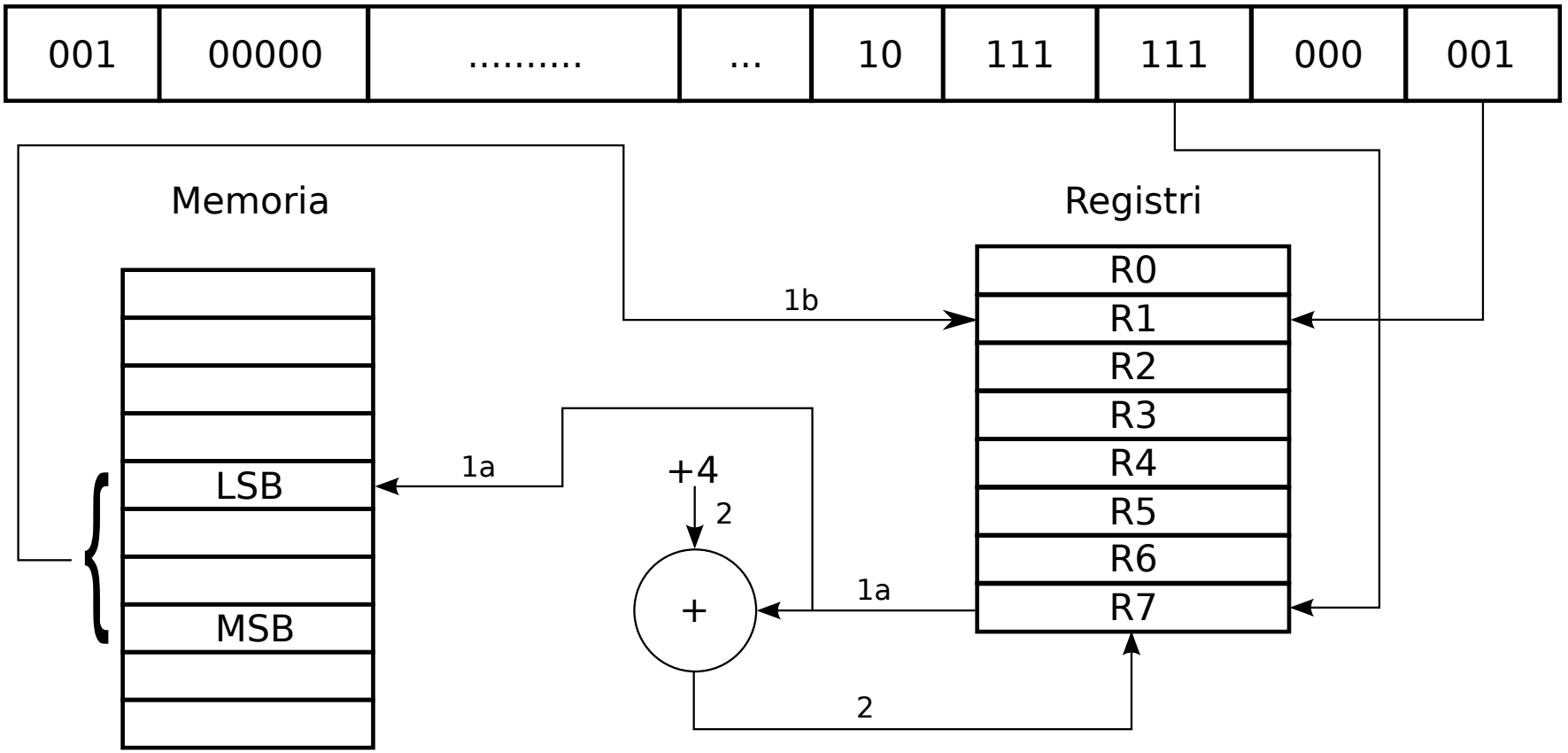
Indirizzamento: indiretto con registro e predercemento

Esempio: `MOVL R1, -(R7)`:



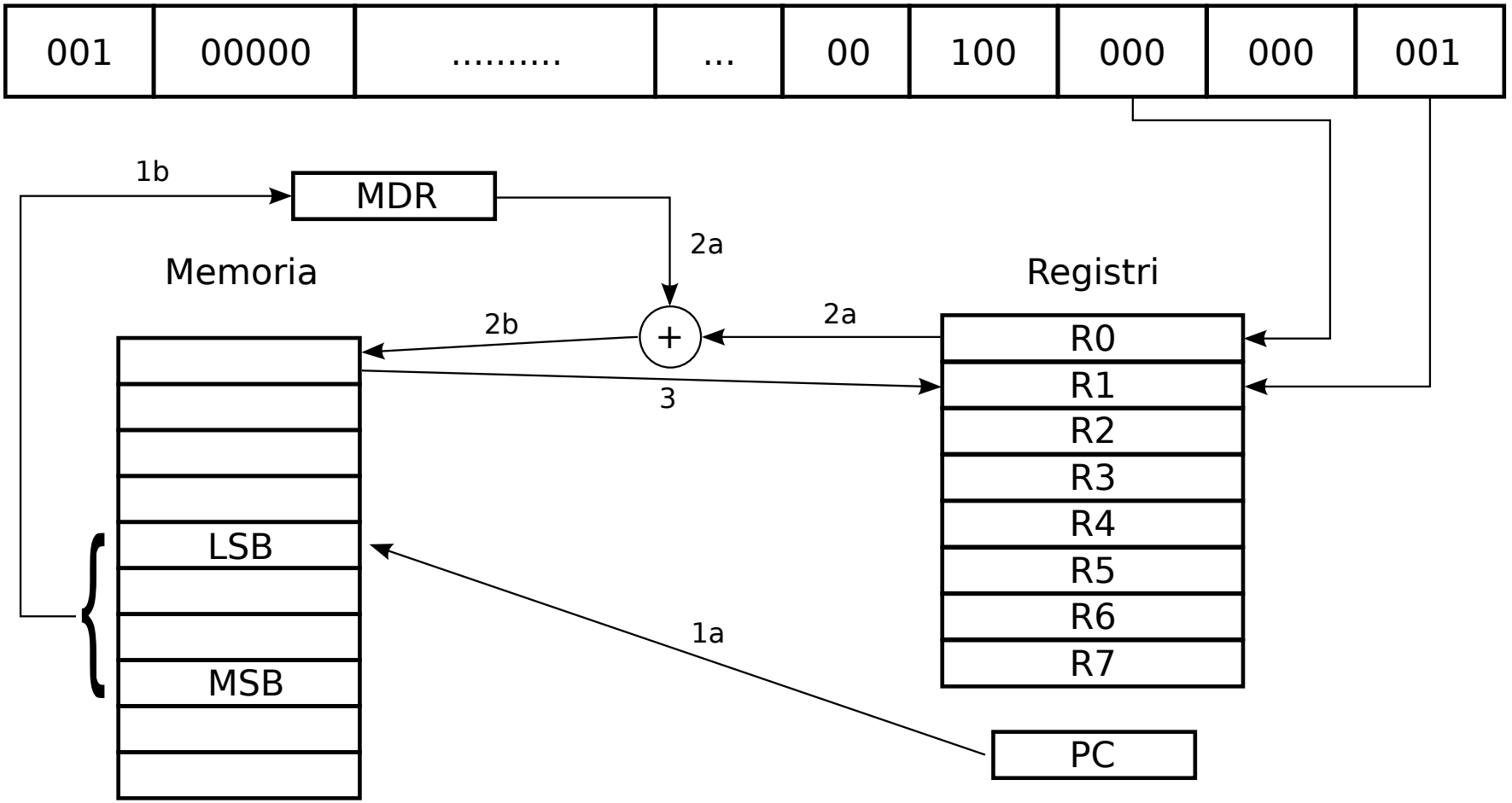
Indirizzamento: indiretto con registro e postincremento

Esempio: `MOVL (R7)+, R1`:



Indirizzamento: con spiazzamento

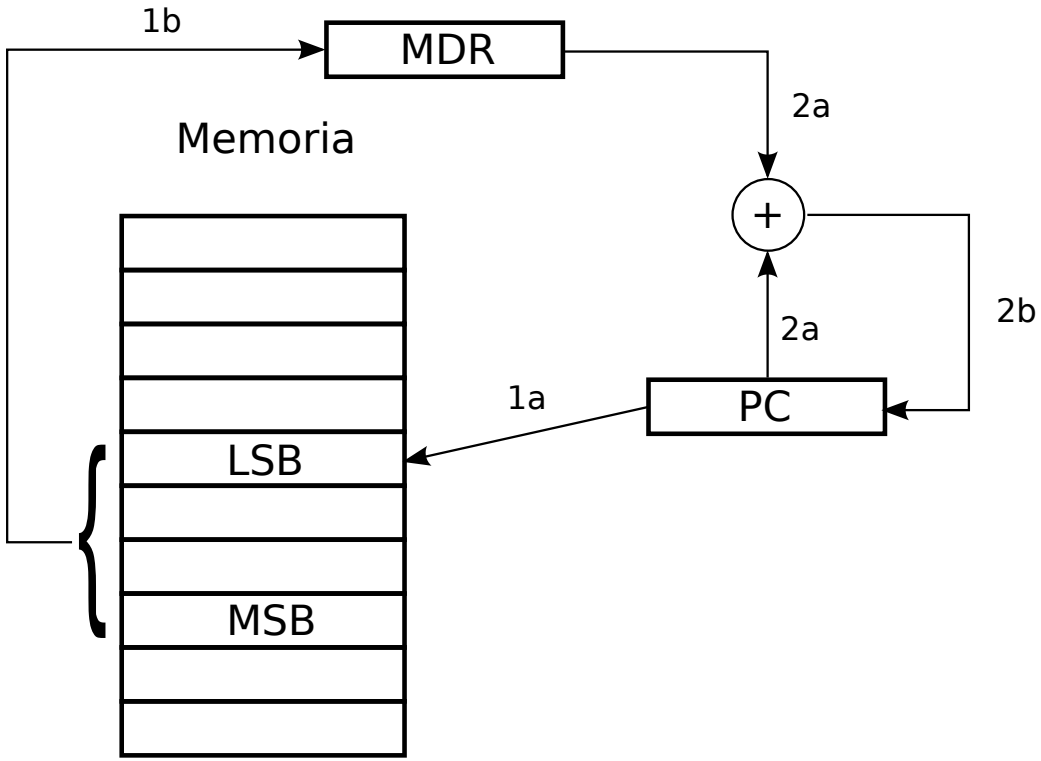
Esempio: `MOVB D(R0), R1`:



Indirizzamento: relativo

Esempio: `JMP LABEL(PC):`

110	00000	.....	...	10	...	...	101	...



Modalità di Indirizzamento: tabella riassuntiva

Nome	**Codifica**	**Descrizione**
modo 0	000	diretto con registro
modo 1	001	immediato
modo 2	010	assoluto
modo 3	011	indiretto con registro
modo 4	100	indiretto con spiazzamento
modo 5	101	relativo
modo 6	110	indiretto con predecremento
modo 7	111	indiretto con postincremento

Modalità di Indirizzamento: un esempio

```
1  ; indirizzamento immediato
2  movl #20, r1 ; r1 = 20
3
4
5  ; indirizzamento a registro
6  addl r1, r1 ; r1 = 40
7
8
9  ; indirizzamento assoluto
10 movb #0FFh, 800h ;mem[0x800]=0xFF
11
12
13 ; indirizzamento con registro
14 movl #800h,r2 ;r2=0x800
15 movb #0EEh, (r2) ;mem[r2]=0xEE
16
```

Modalità di Indirizzamento: un esempio (2)

```
17 ; con predecremento
18 movb #0FFh, -(r2) ;r2=0x800-0x1=0x7FF, mem[0x7FF]=0xFF
19
20
21 ; con postincremento
22 movb #0AAh, (r2)+ ;mem[0x7FF]=0xAA, r2=0x800
23
24
25 ; con spiazzamento
26 movb #0FFh, 8(r2) ;mem[0x808]=0xFF, r2=0x800
```

Classe 1: Istruzioni di movimento dati

Tipo	Codice	Operandi	C N Z V P I	Commento
0	MOVs	S, D	- - - - - -	La sorgente S viene posta nella destinazione D
1	MOVFRSR	D	- - - - - -	Il contenuto del registro SR viene posto nella destinazione D
2	MOVTOSR	S	- - - - - -	Il contenuto della sorgente S viene posto nel registro SR
3	MVLs	S, D	- - - - - -	Equivalente a MOVs, ma senza estensione del segno se D è un registro
0	PUSH	S	- - - - - -	Sinonimo di MOVL S, -(R7)
0	POP	D	- - - - - -	Sinonimo di MOVL (R7)+, D
1	PUSHSR	–	- - - - - -	Sinonimo di MOVFRSR -(R7)
2	POPSR	–	- - - - - -	Sinonimo di MOVTOSR (R7)+

Istruzioni `MOVs`

Sono usate per copiare:

- Registro–Registro: `MOVs R1, R2`
- Registro–Memoria: `MOVs R1, (R2)`
- Memoria–Registro: `MOVs (R1), R2`
- Memoria–Memoria: `MOVs (R1), (R2)`

In generale, le istruzioni di copia memoria–memoria non sono supportate da tutte le architetture. Alcuni processori forniscono delle istruzioni speciali per effettuare queste operazioni.

Classe 2: Istruzioni aritmetiche

Tipo	Codice	Operandi	C N Z V P I	Commento
0	ADDs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	La sorgente S viene sommata al registro R ed il risultato posto nel registro R destinazione (S + R → R)
1	ADCs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Come ADC, ma somma anche il bit di carry (S + R + C → R)
2	CMPs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Effettua una sottrazione R - sorg, ma senza salvare il risultato in R
3	NEGs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	La sorgente S viene complementata ed il risultato salvato nel registro R (0 - S → R)
4	SUBs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Viene sottratta la sorgente S dal registro R ed il risultato posto nel registro R destinazione (R - S → R)
5	SBBs	S, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Come SUB, ma sottrae anche il bit di carry (R - S - C → R)

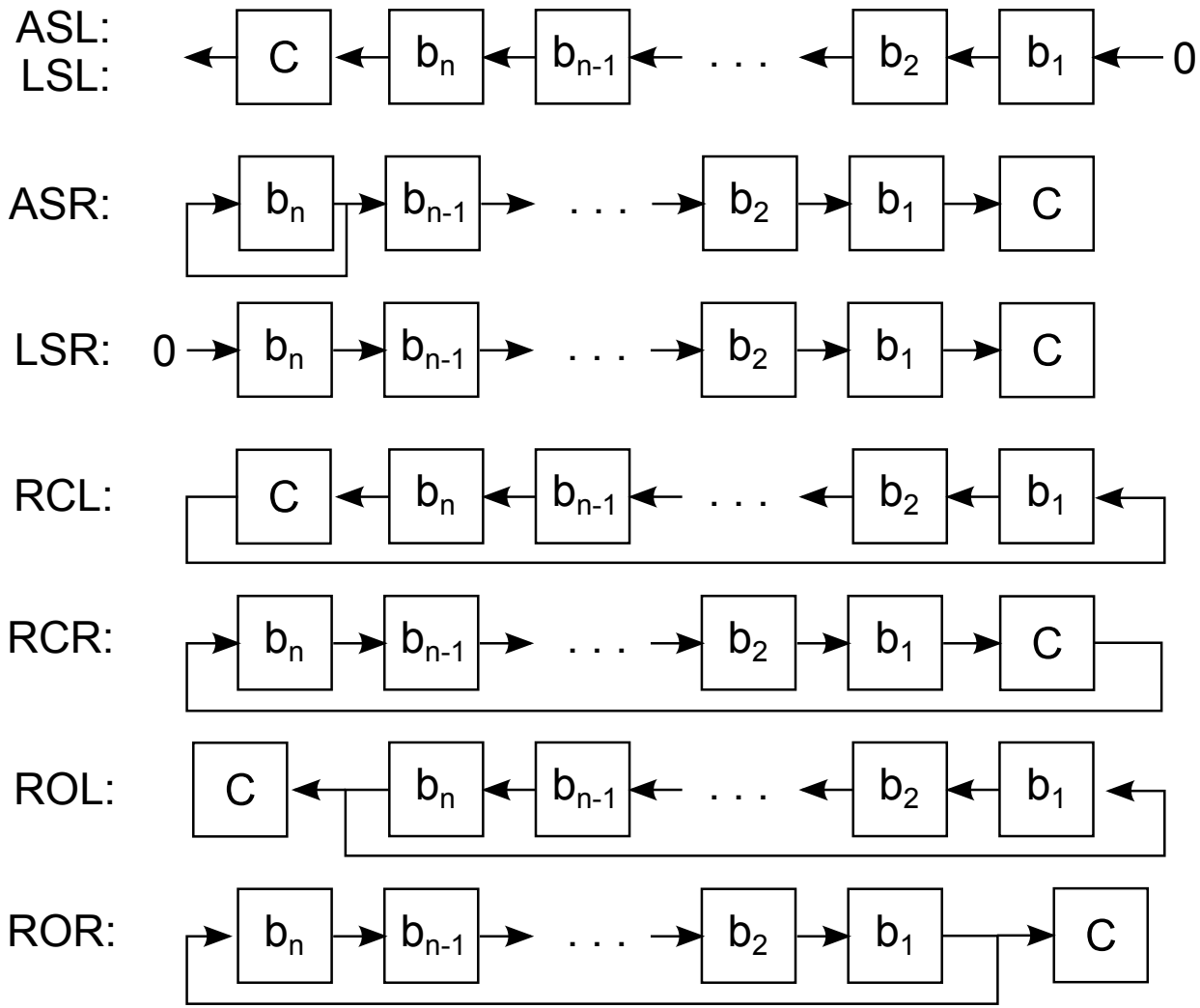
Classe 3: Istruzioni di tipo logico

Tipo	Codice	Operandi	C N Z V P I	Commento
0	ANDs	S, R	0 ⇕ ⇕ 0 ⇕ -	Il risultato del bitwise AND tra S e D viene memorizzato in R
1	ORs	S, R	0 ⇕ ⇕ 0 ⇕ -	Il risultato del bitwise OR tra S e D viene memorizzato in R
2	XORs	S, R	0 ⇕ ⇕ 0 ⇕ -	Il risultato del bitwise XOR tra S e D viene memorizzato in R
3	NOTs	S, R	0 ⇕ ⇕ 0 ⇕ -	La sorgente S viene complementata a 1 ed il risultato è posto in R

Classe 4: Istruzioni di rotazione e shift

Tipo	Codice	Operandi	C N Z V P I	Commento
0	ASLs	K, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Shift aritmetico a sinistra di K posti del dato contenuto in R
1	ASRs	K, R	⇕ ⇕ ⇕ ⇕ ⇕ -	Shift aritmetico a destra di K posti del dato contenuto in R
2	LSLs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Shift logico a sinistra di K posti del dato contenuto in R
3	LSRs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Shift logico a destra di K posti del dato contenuto in R
4	RCLs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Ruota a sinistra di K posti il dato contenuto in R
5	RCRs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Ruota a destra di K posti il dato contenuto in R
6	ROLs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Ruota a sinistra di K posti il dato contenuto in R
7	RORs	K, R	⇕ ⇕ ⇕ 0 ⇕ -	Ruota a destra di K posti il dato contenuto in R

Operazioni di rotazione e shift



Classe 5: Istruzioni su bit di condizione

Tipo	Codice	Operandi	C N Z V P I	Commento
0	CLRC	–	0 - - - - -	0 → C
1	CLRN	–	- 0 - - - -	0 → N
2	CLRZ	–	- - 0 - - -	0 → Z
3	CLRV	–	- - - 0 - -	0 → V
4	CLRP	–	- - - - 0 -	0 → P
5	CLRI	–	- - - - - 0	0 → I
10	SETC	–	1 - - - - -	1 → C
11	SETN	–	- 1 - - - -	1 → N
12	SETZ	–	- - 1 - - -	1 → Z
13	SETV	–	- - - 1 - -	1 → V
14	SETP	–	- - - - 1 -	1 → P
15	SETI	–	- - - - - 1	1 → I

Classe 6: Istruzioni di controllo del programma

Tipo	Codice	Operandi	C N Z V P I	Commento
0	JMP	D	- - - - - -	Salta all'indirizzo specificato dalla destinazione D
1	JSR	D	- - - - - -	Salva PC nello *stack* e salta all'indirizzo specificato dalla destinazione D
2	RET	–	- - - - - -	Salva in PC l'elemento affiorante nello *stack*
3	RTI	–	- - - - - -	Ritorno da interruzione: ripristina SR e tt PC dallo *stack*
10+c	Jc	D	- - - - - -	Salta all'indirizzo specificato dalla destinazione D se il flag c = 1
18+c	JNc	D	- - - - - -	Salta all'indirizzo specificato dalla destinazione D se il flag c = 0

Classe 7: Istruzioni di I/O

Tipo	Codice	Operandi	C N Z V P I	Commento
0	INs	dev, R	- - - - - -	Copia il dato dal buffer del device dev in R
1	OUTs	dev, R	- - - - - -	Copia il dato da (R) nel buffer del device dev
2	START	dev	- - - - - -	Azzera il flip-flop READY del dispositivo e avvia l'operazione di I/O
3	CLEAR	dev	- - - - - -	Azzera il flip-flop READY del dispositivo
4	JR	dev, R	- - - - - -	Se READY=1, va all'indirizzo (R)
5	JNR	dev, R	- - - - - -	Se READY=0, va all'indirizzo (R)
6	SETIMS	dev	- - - - - -	Abilita il dispositivo dev ad inviare interruzioni (1 $\rightarrow$ IMS)
7	CLRIMS	dev	- - - - - -	Impedisce al dispositivo dev di

Classe 8: Istruzioni di controllo macchina

Tipo	Codice	Operandi	C N Z V P I	Commento
0	HALT	–	- - - - - -	Pone il processore in uno stato di attesa, da cui si esce alla ricezione di un'interruzione o all'arrivo di RESET
1	NOP	–	- - - - - -	No Operation: il processore non fa nulla, al termine dell'esecuzione il flusso procede verso l'istruzione successiva
2	RESET	–	0 0 0 0 0 0	Forza a 0 il registro SR e pone in PC il valore contenuto sull'I/O DB