

**IL LIVELLO DI
MICROARCHITETTURA
E
MACROARCHITETTURA
(Prima Parte)**

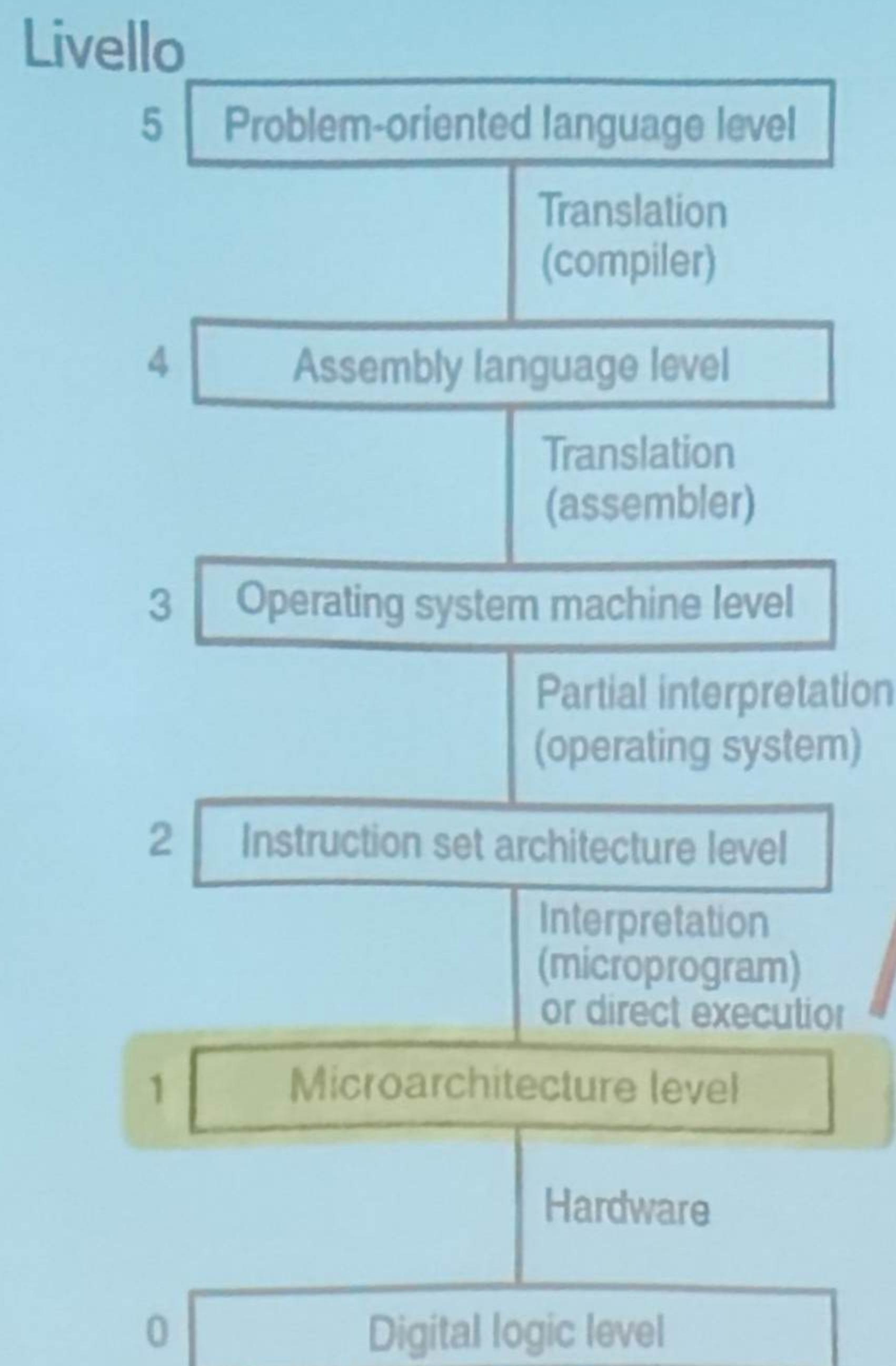
Argomenti

- IL LIVELLO DI MICROARCHITETTURA
 - Un esempio di microarchitettura
 - modello di esecuzione
 - data path (o percorso dati)
 - formato delle microinstruzioni
 - microarchitettura Mic-1
 - Esempio di ISA: IJVM
 - Lo stack
 - Il modello della memoria
 - Insieme delle istruzioni
- IL LIVELLO DI MACROARCHITETTURA
 - Overview del livello ISA
 - Tipi di dati

Obiettivi

- Analizzare il livello di microarchitettura e del funzionamento interno attraverso il data path.
- Comprendere il modello di memoria e l'insieme delle istruzioni.
- Introdurre il livello ISA.

IL LIVELLO DI MICROARCHITETTURA



- Il livello di Microarchitettura fornisce una visione astratta per il livello ISA (Instruction Set Architecture).
- Dipende non solo dall'ISA ma anche dagli obiettivi di costo e di prestazioni della macchina che si intende costruire.

Esempio di microarchitettura

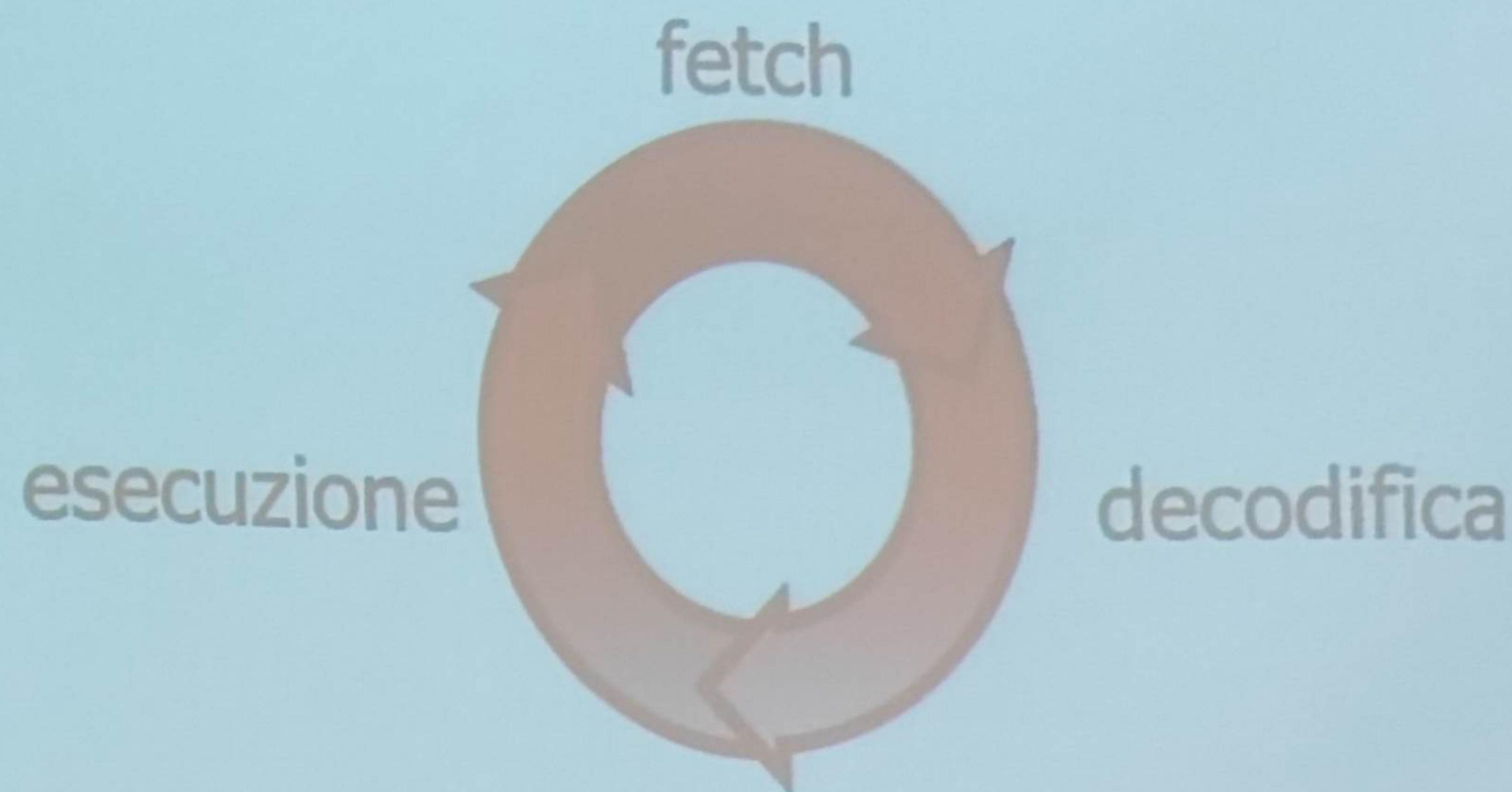
- Partiamo da un esempio poiché non esistono principi generali di costruzione di una ISA (Instruction Set Architecture).
- Utilizzeremo il sottoinsieme intero della Java Virtual Machine (JVM) che chiameremo **IJVM**.
- Utilizzeremo un **micropogramma** memorizzato in una ROM.

Esempio di microarchitettura

- Il microprogramma si compone di una sequenza di microistruzioni che utilizzano variabili che costituiscono lo **stato della macchina**.
- Il **Program Counter** (PC) contiene il riferimento della prossima microistruzione da eseguire.
- Ogni microistruzione di compone di:
 - **Codice operativo** (Opcode) che identifica il tipo di istruzione.
 - Operando/i su cui si applicherà l'istruzione.

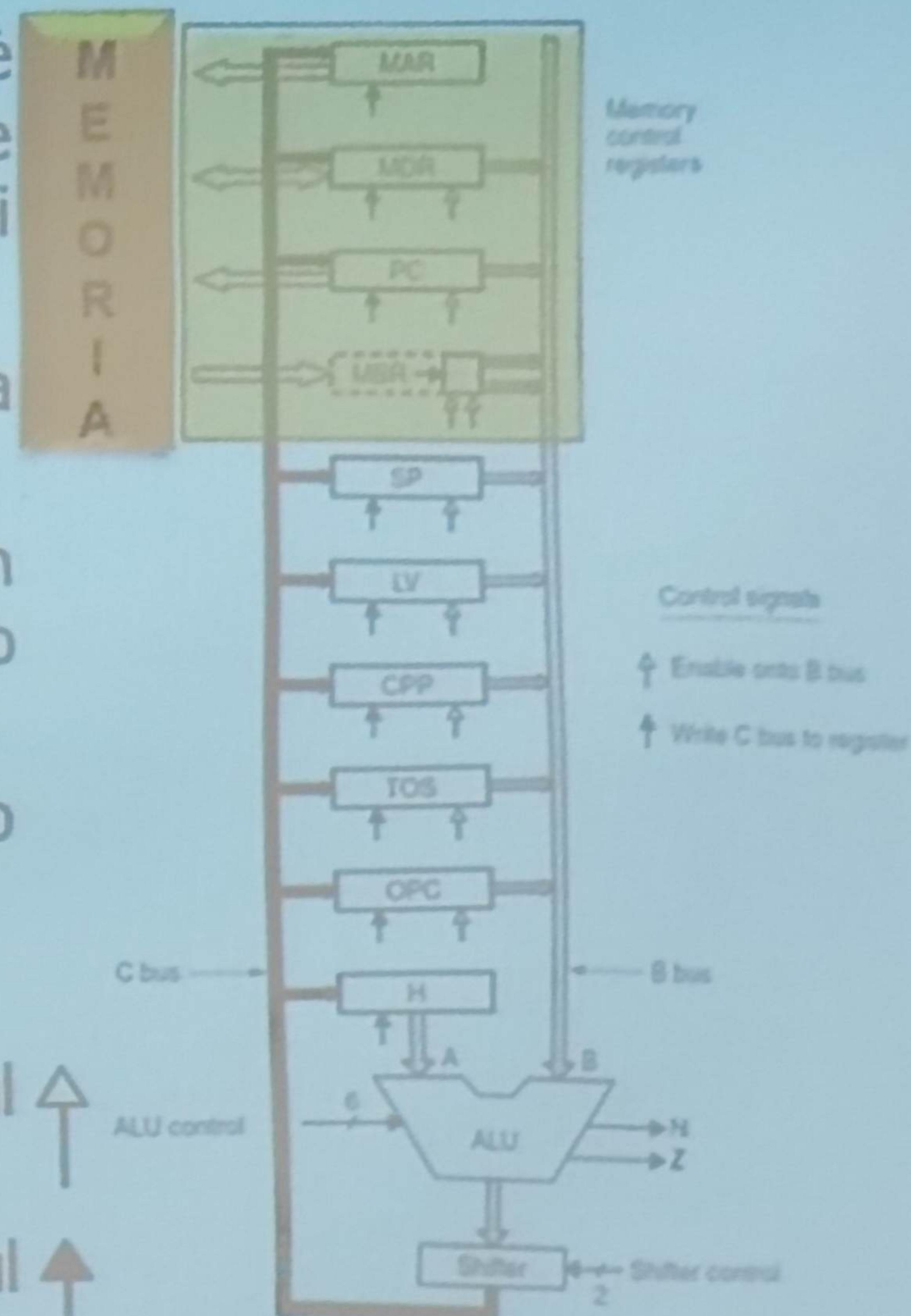
Il modello di esecuzione

- Il modello di esecuzione adottato è basato su tre fasi:
 - **fetch**, caricamento in memoria dell'istruzione.
 - **decodifica**, riconoscimento del tipo di istruzione da eseguire.
 - **esecuzione**, svolgimento del compito.



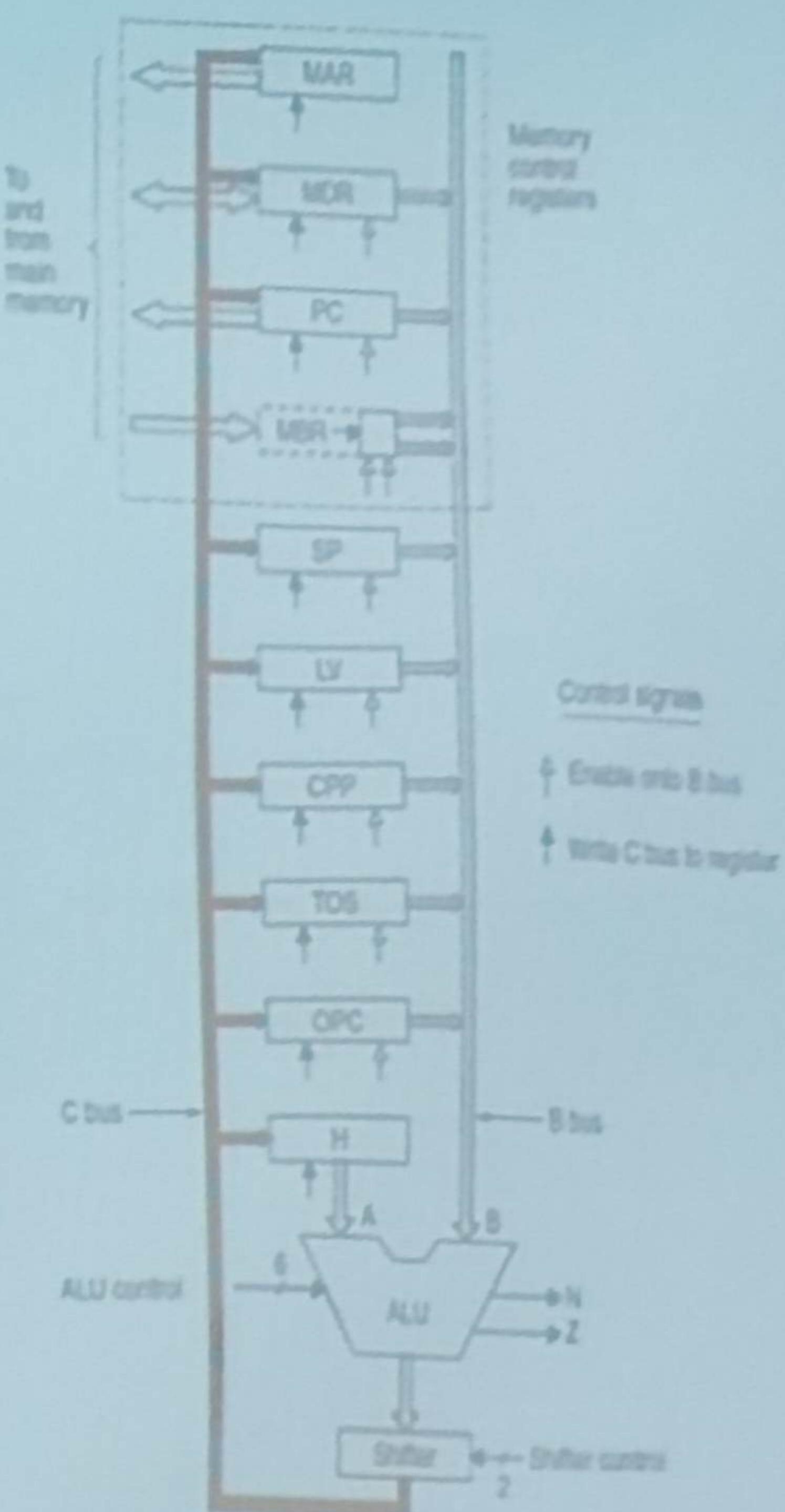
Il data path (o percorso dati)

- Il **data path** (o percorso dati) è quella parte della CPU che contiene l'ALU, i registri interni, gli input e gli output.
- Un insieme di registri controlla l'accesso in memoria.
- Si compone di due bus: **B** e **C** (in quanto l'operando **A** è quello contenuto nel registro **H**).
- Alla base dell'ALU troviamo uno shifter.
- Esistono due segnali di controllo:
 - abilita la scrittura del registro sul bus **B**.
 - scrive il contenuto del bus **C** sul registro.



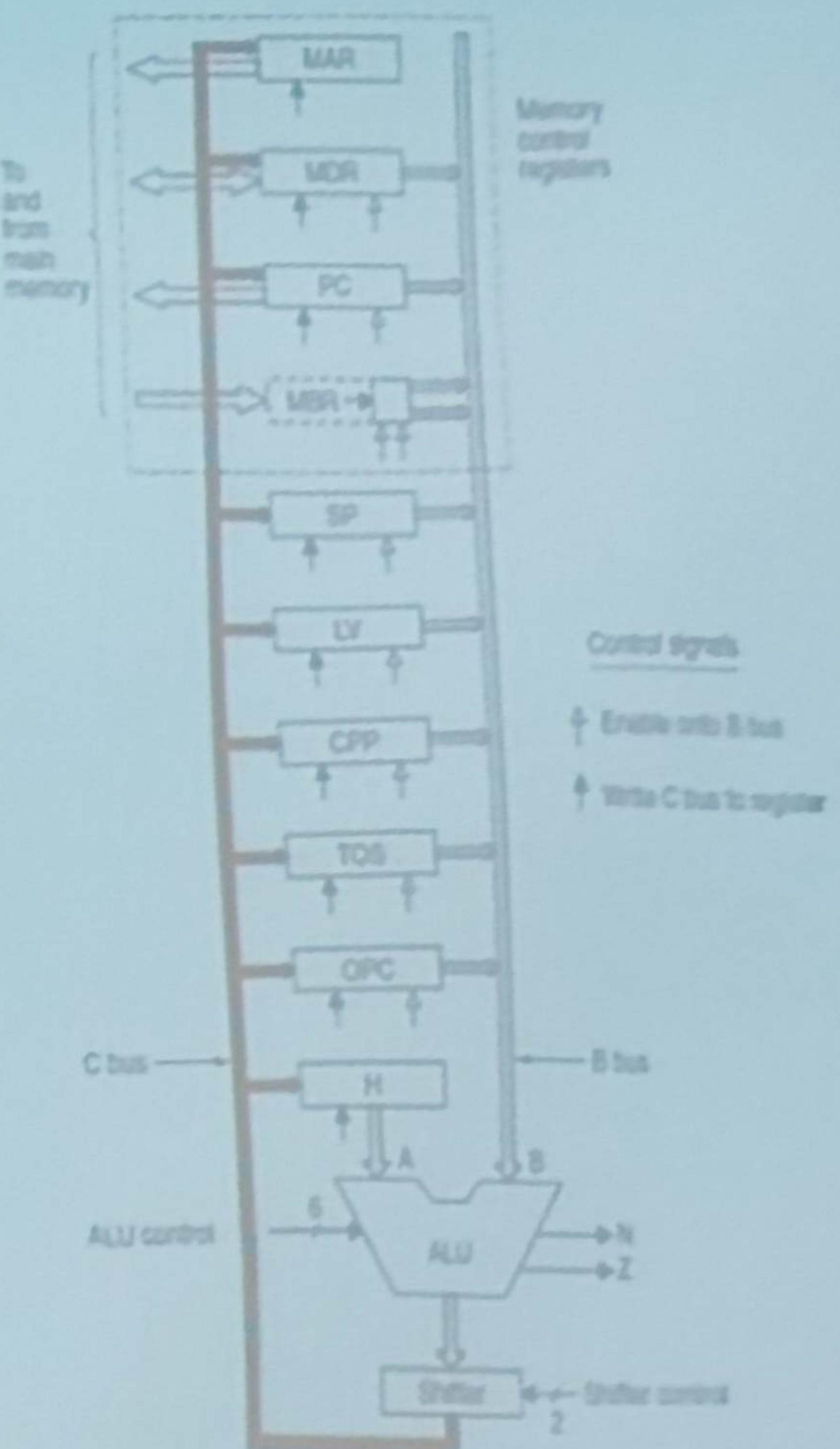
Il data path (o percorso dati)

- I registri sono:
 - Memory Address Register (**MAR**).
 - Memory Data Register (**MDR**).
 - Program Counter (**PC**).
 - Memory Byte Register (**MBR**), è un byte nello stream di istruzioni che provengono dalla memoria.
 - Stack Pointer (**SP**).
 - Local Variabile (**LV**), il riferimento nello stack alla base delle variabili locali.



Il data path (o percorso dati)

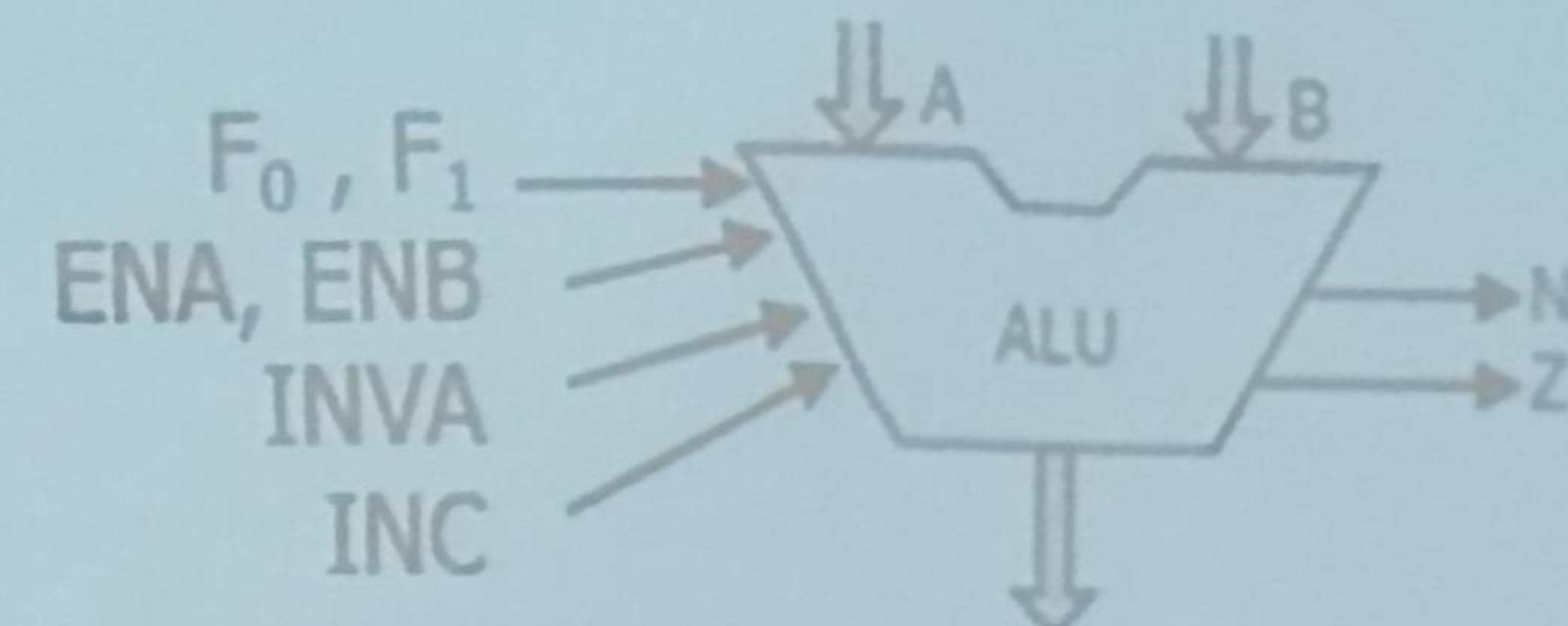
- Gli altri registri sono:
 - Constant Pool (**CPP**).
 - Top word On the Stack (**TOS**).
 - Op Code register (**OPC**): registro temporaneo, può contenere l'ultima istruzione eseguita prima di un salto.
 - Holding (**H**).



Arithmetic Logic Unit

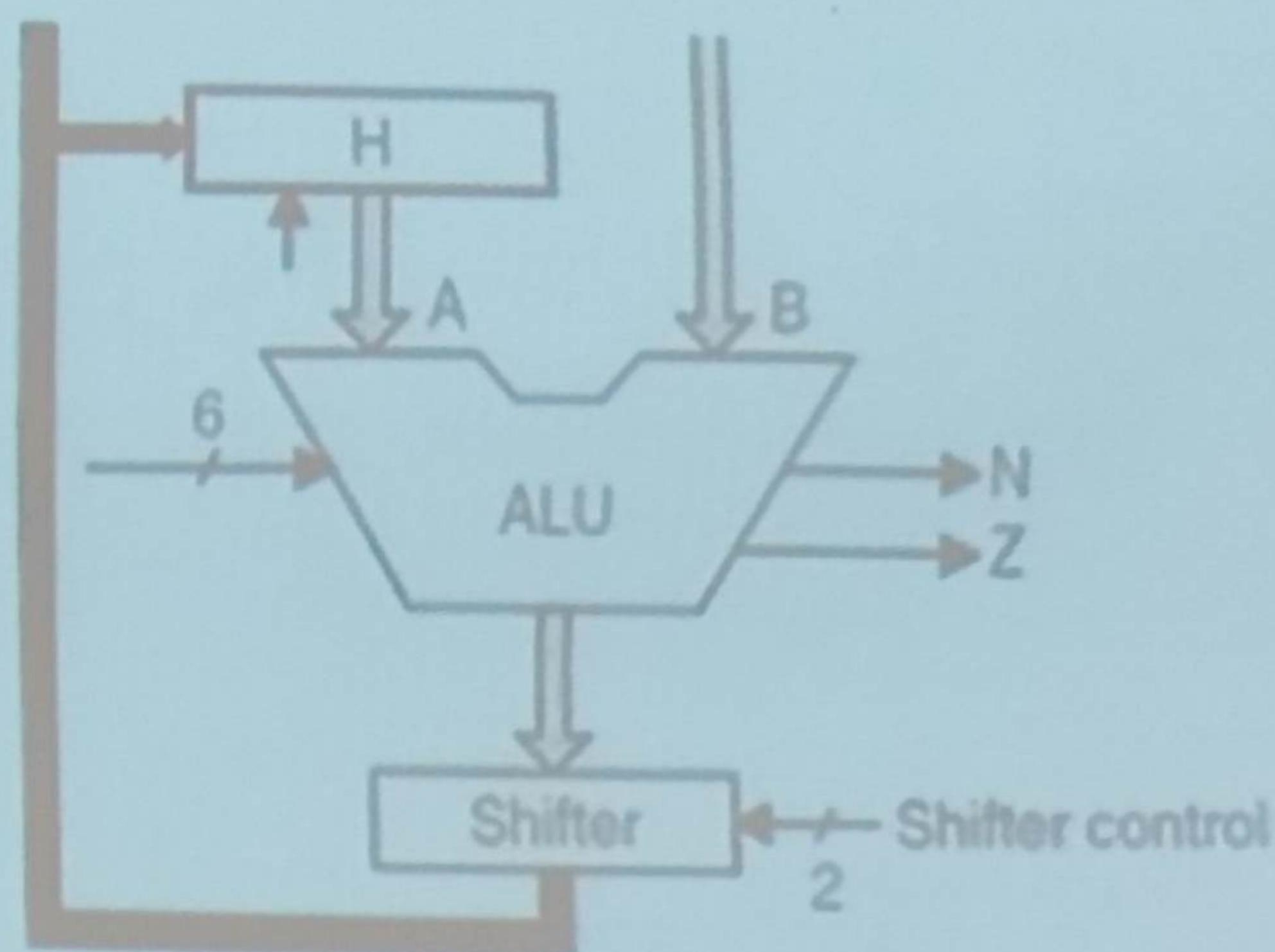
- La ALU ha sei linee di controllo:
 - F_0 e F_1 selezionano il tipo di funzione.
 - ENx , abilita il valore della variabile x altrimenti lo annulla.
 - $INVA$, esegui una sottrazione della variabile A .
 - INC , incrementa.

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



Arithmetic Logic Unit

- La ALU agisce su due operandi: uno viene dal registro **H** e l'altro dal bus **B**.
- È possibile “spostare” l'operando da B ad A, applicando la funzione che restituisce l'operando B e successivamente memorizzando il risultato del bus **C** in **H**.



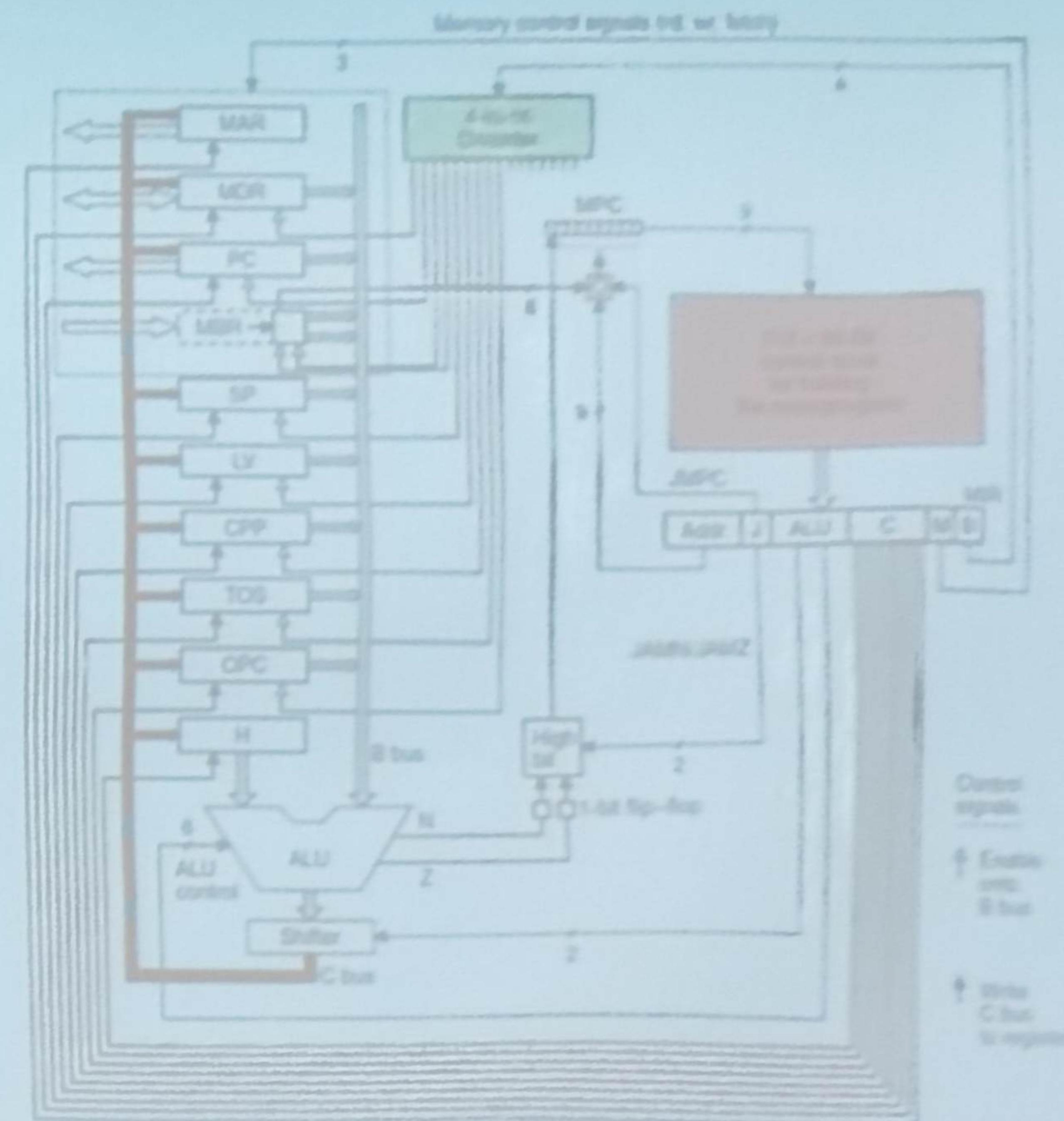
- Lo shifter può far scorrere i bit/byte (aritmetico/logico) del risultato verso destra o sinistra.

Il formato delle microistruzioni

- **Addr**, è l'indirizzo della potenziale successiva microistruzione.
 - **JAM**, determina come viene selezionata la prossima microistruzione.
 - **ALU**, seleziona le funzioni dell'ALU e dello shifter
 - **C**, seleziona quali registri sono scritti dal bus C.
 - **Mem**, seleziona la funzione in memoria.
 - **B**, seleziona quale registro è scritto sul bus B.

Microarchitettura Mic-1

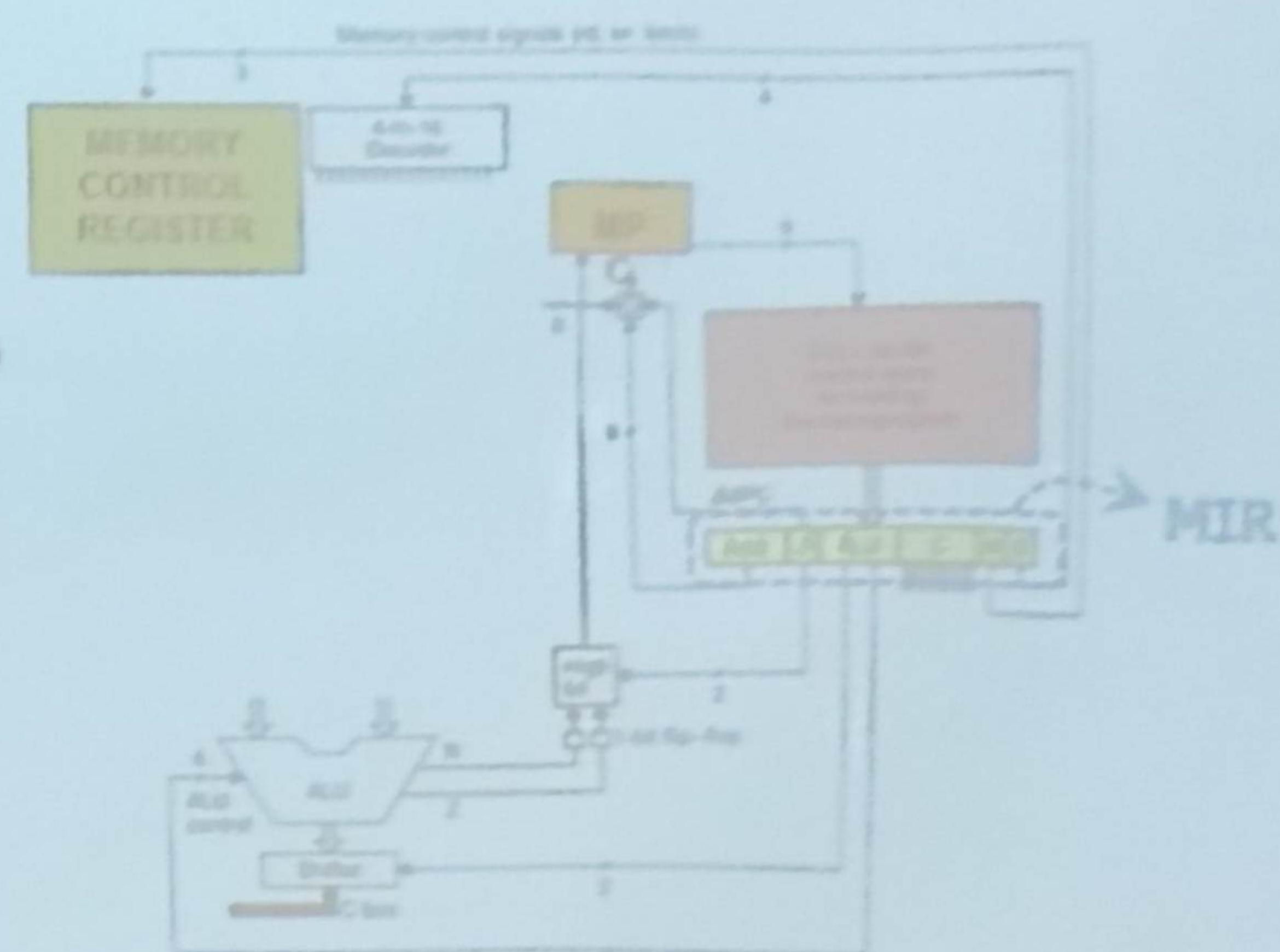
- La memoria di controllo memorizza le microistruzioni.
- Il decoder decodifica il tipo di istruzione in base al opcode.



Microarchitettura Mic-1

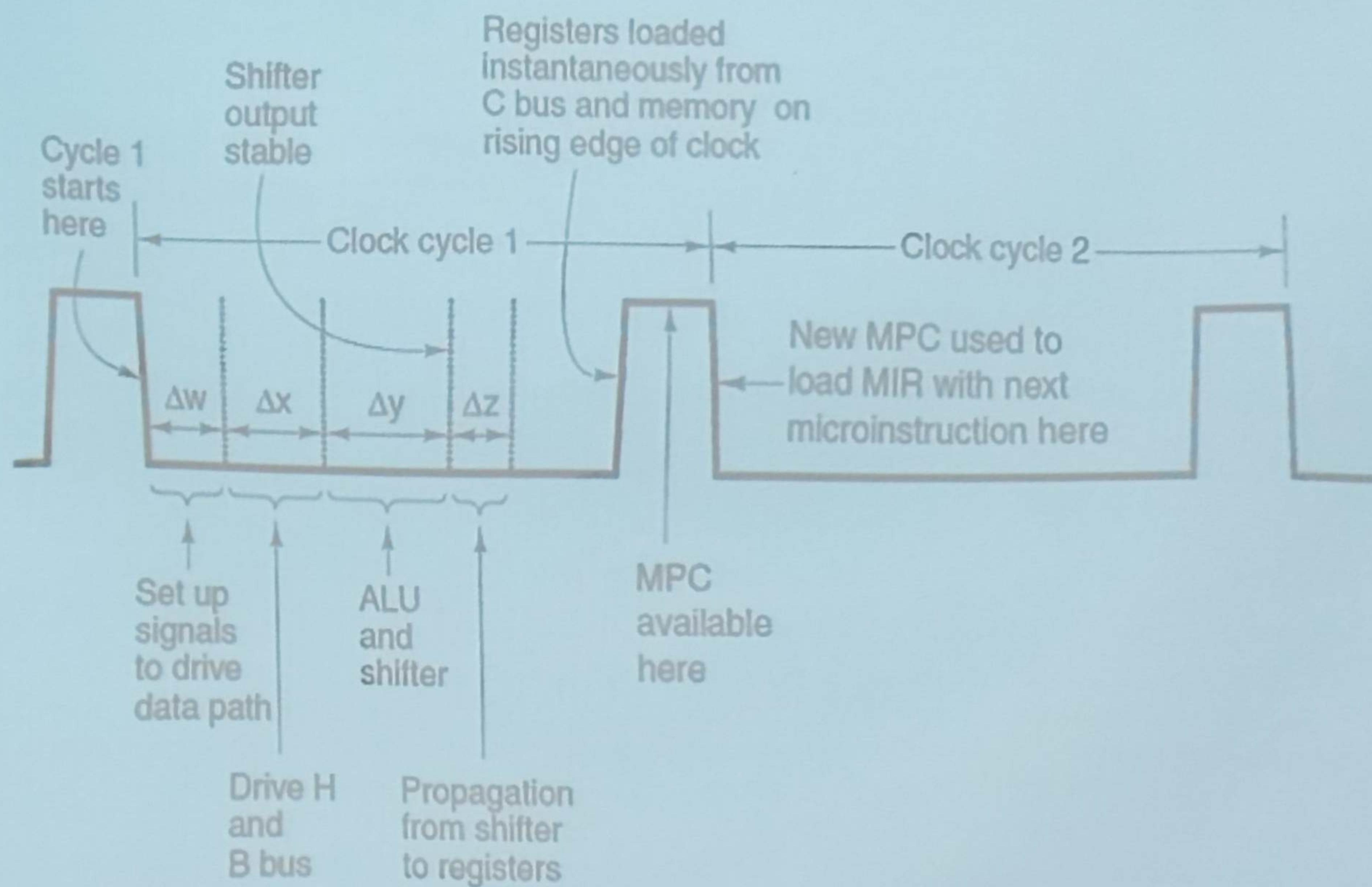
- Le istruzioni in memoria centrale sono eseguite nell'ordine di memorizzazione (a meno di salti).
- ciascuna microistruzione determina la successiva istruzione (non hanno necessariamente una memorizzazione lineare).

- La memoria di controllo ha un proprio registro indirizzo (MicroProgram Counter, **MPC**) e dati (MicroInstruction Register, **MIR**)



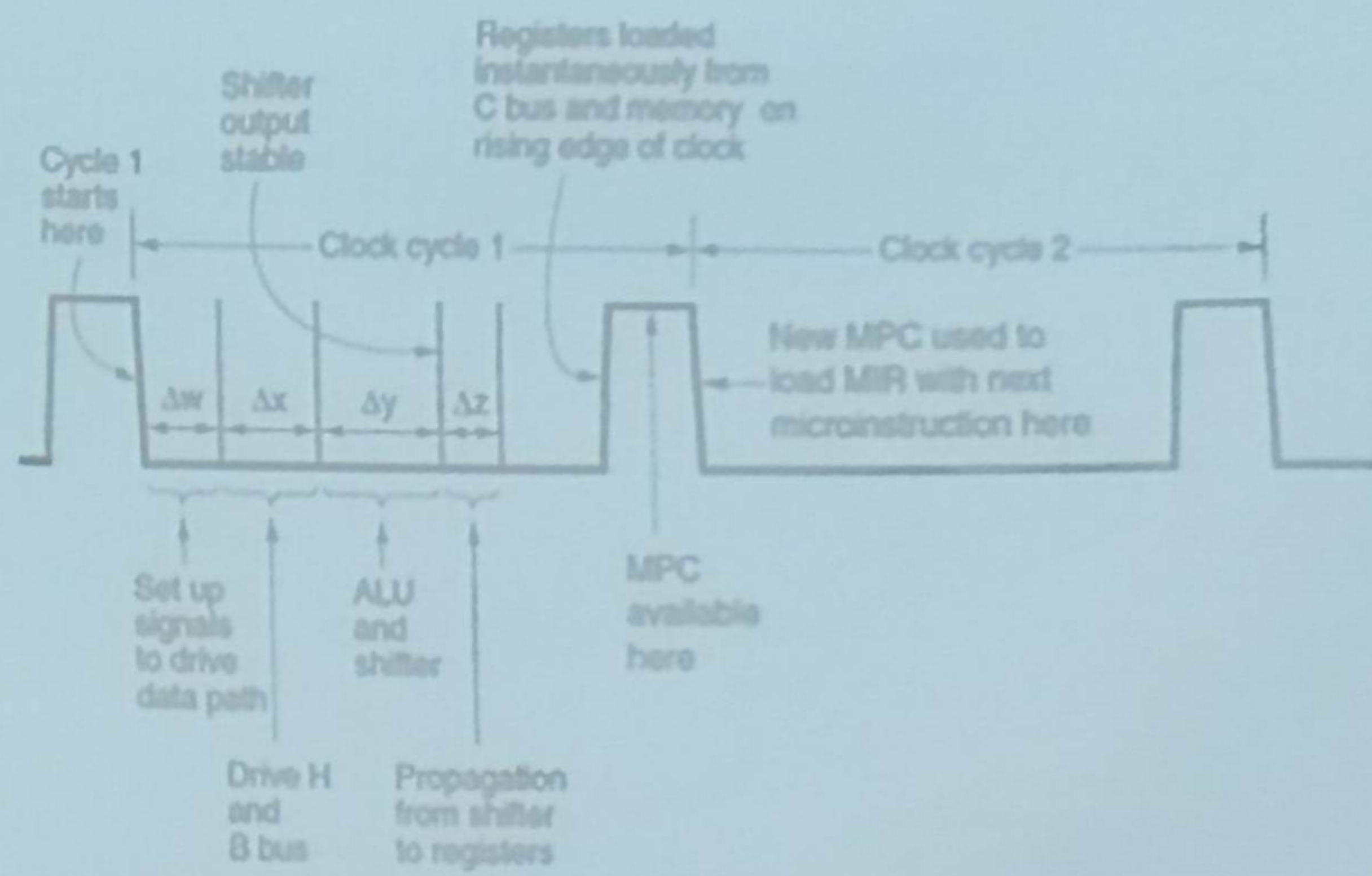
Il funzionamento di Mic-1

- All'inizio del fronte in discesa del clock, la parola puntata da MPC contenuta nella memoria di controllo passa in MIR



Il funzionamento di Mic-1

- I segnali si propagano nel data path ed un registro viene inserito nel bus B e la ALU sa quale operazione compiere.
- Una volta che gli input della ALU sono stabili, esegue il calcolo ed i segnali si propagano in uscita.
- Quando i segnali sono stabili, il bus C viene memorizzato all'interno dei registri candidati in corrispondenza del fronte in salita del clock (e termina così un ciclo).

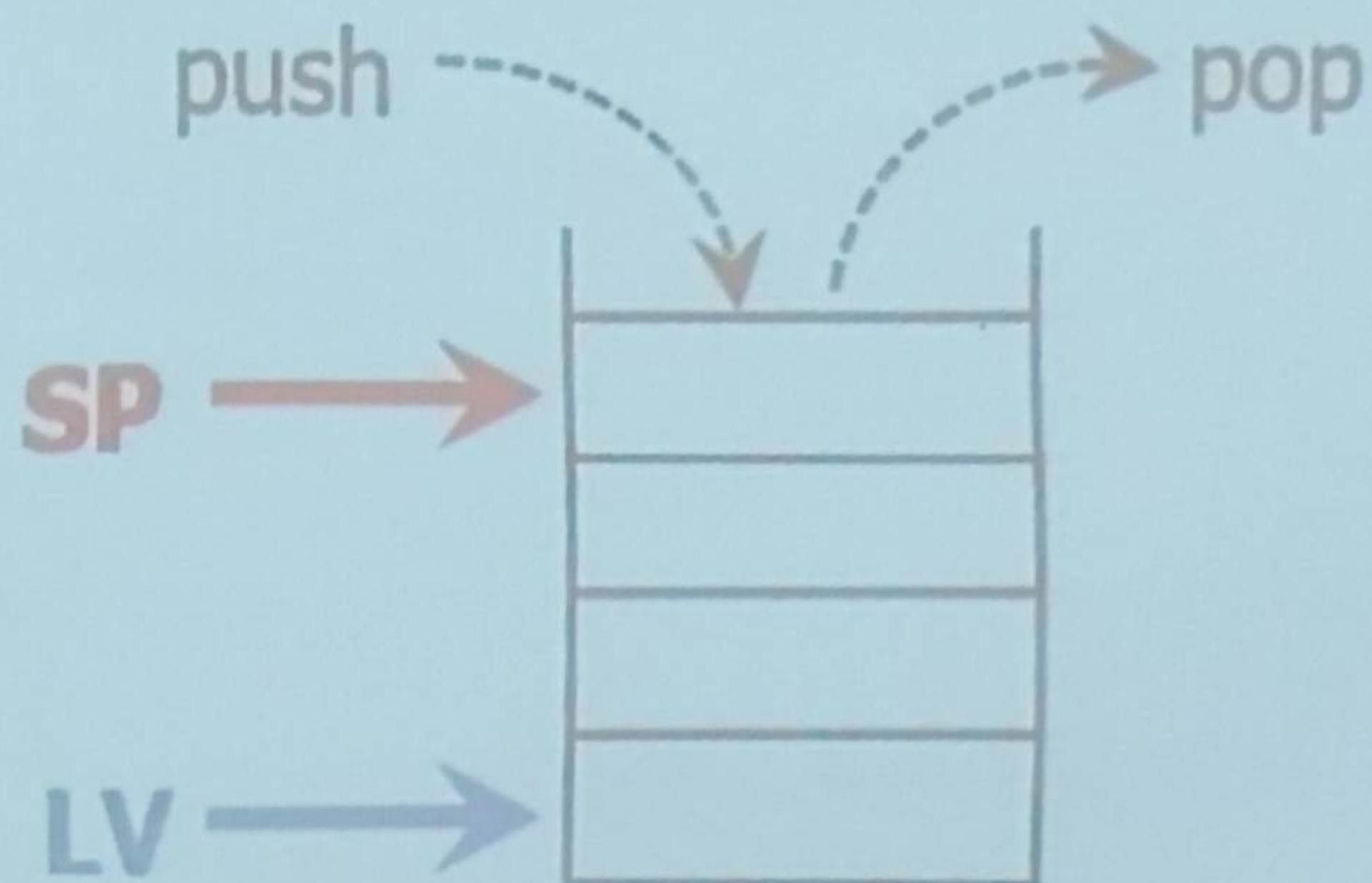


Un esempio di ISA: IJVM

- Lo stack.
- Il modello della memoria.
- L'insieme delle istruzioni.

Lo stack

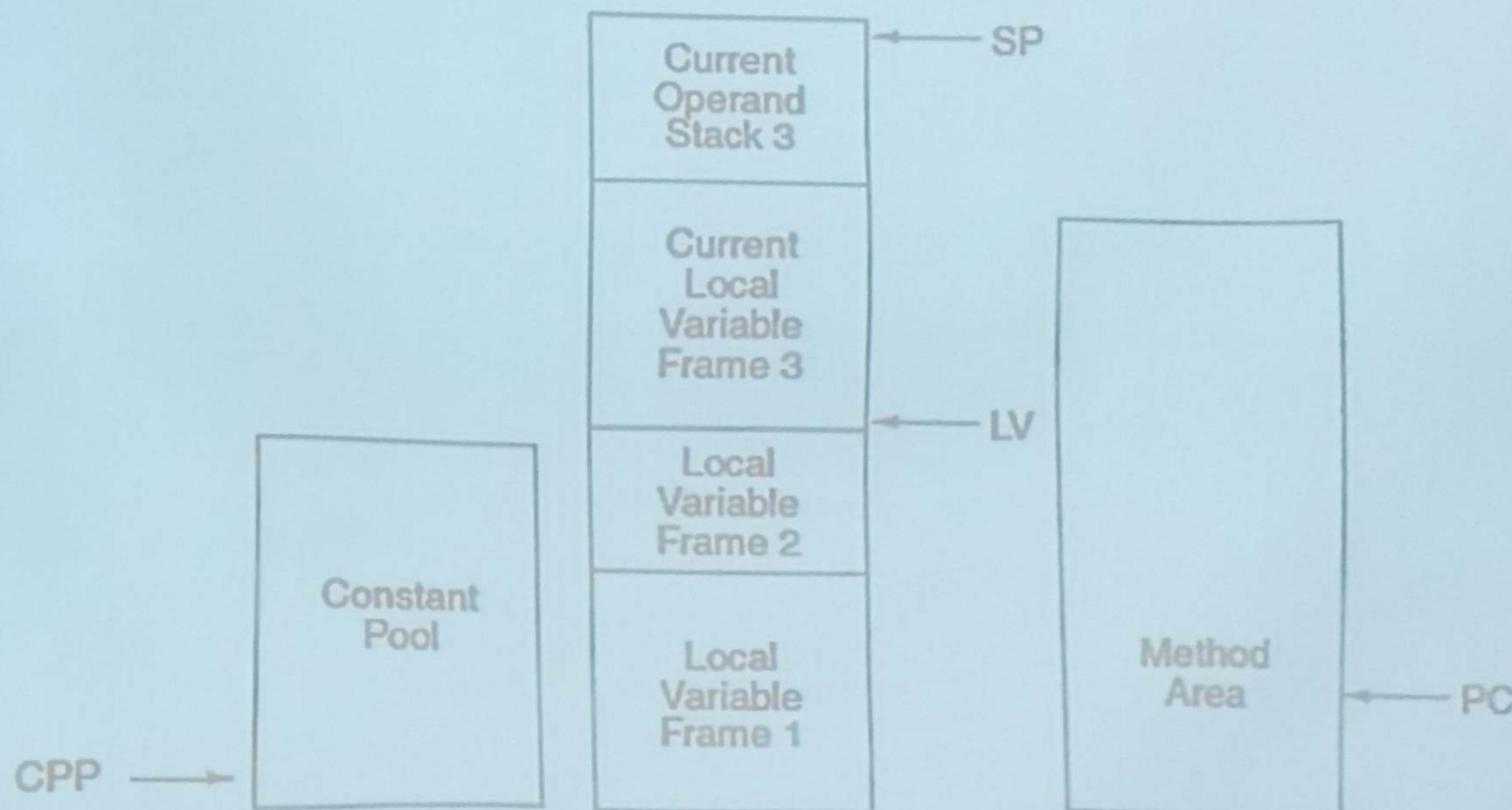
- È una struttura dati utilizzata per memorizzare lo stato di una procedura che segue la filosofia LIFO (Last-In-First-Out).



- Questa organizzazione dei dati permette di gestire anche situazioni di chiamate ricorsive (procedure che richiamano se stesse).

Il modello della memoria IJVM

- La memoria può essere vista come un vettore di:
 - 4.294.967.296 Byte (4 GB)
 - 1.073.741.824 parole di 4 Byte



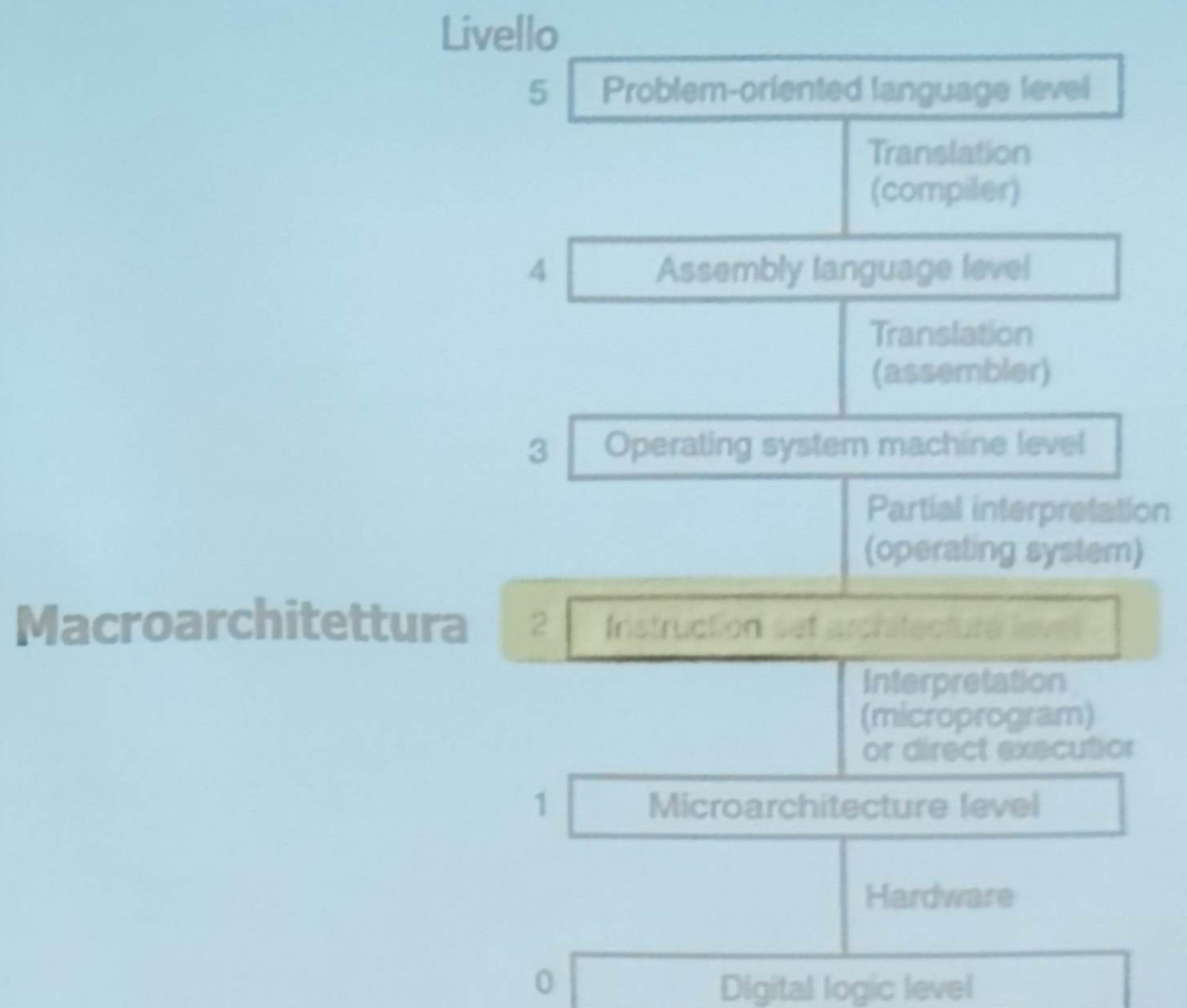
Il modello della memoria IJVM

- Le aree della memoria utilizzabili sono:
 - **Porzione costante della memoria**, caricata con il programma e referenziata dal registro CPP.
 - **Blocco delle variabili locali**, creato per ogni metodo che contiene i parametri di attivazione e le variabili locali.
 - **Stack degli operandi**, allocato sopra le variabili locali.
 - **Area dei metodi**, destinato ad accogliere il programma.

Insieme delle istruzioni della 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

IL LIVELLO DI MACROARCHITETTURA



Macroarchitettura

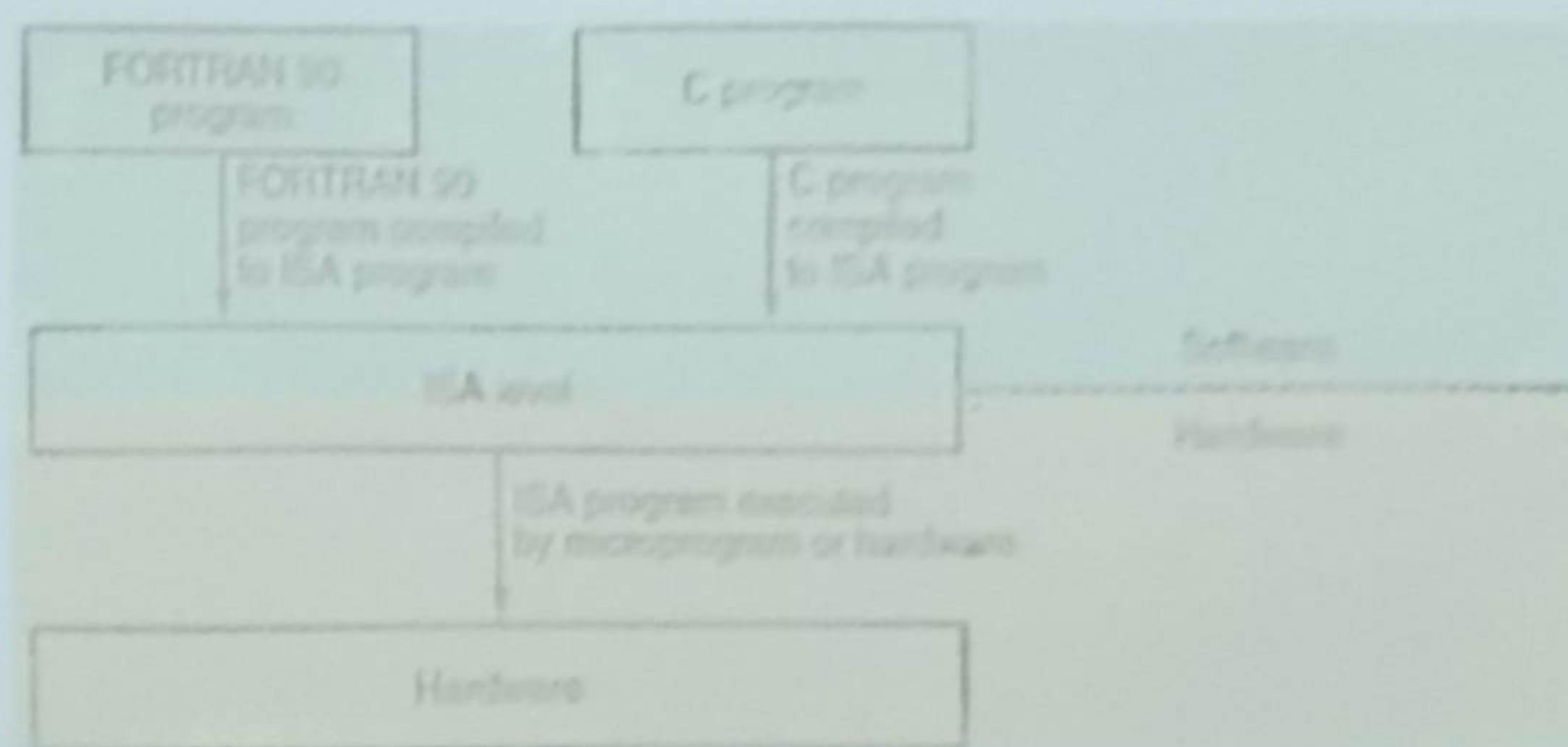
Un esempio
JVM

Overview del livello ISA

- Il livello di architettura dell'insieme delle istruzioni (ISA) è la macchina dal punto di vista del programmatore in linguaggio macchina.
- L'ISA si compone:
 - Del modello di memoria.
 - Dell'insieme dei registri.
 - Dei tipi di dati possibili.
 - Dell'insieme delle istruzioni.
- Dal livello ISA non è possibile conoscere il funzionamento della microarchitettura (parallelismo, pipeline,...).
- Normalmente il livello ISA è descritto attraverso un documento formale di definizione del produttore.

Il livello ISA

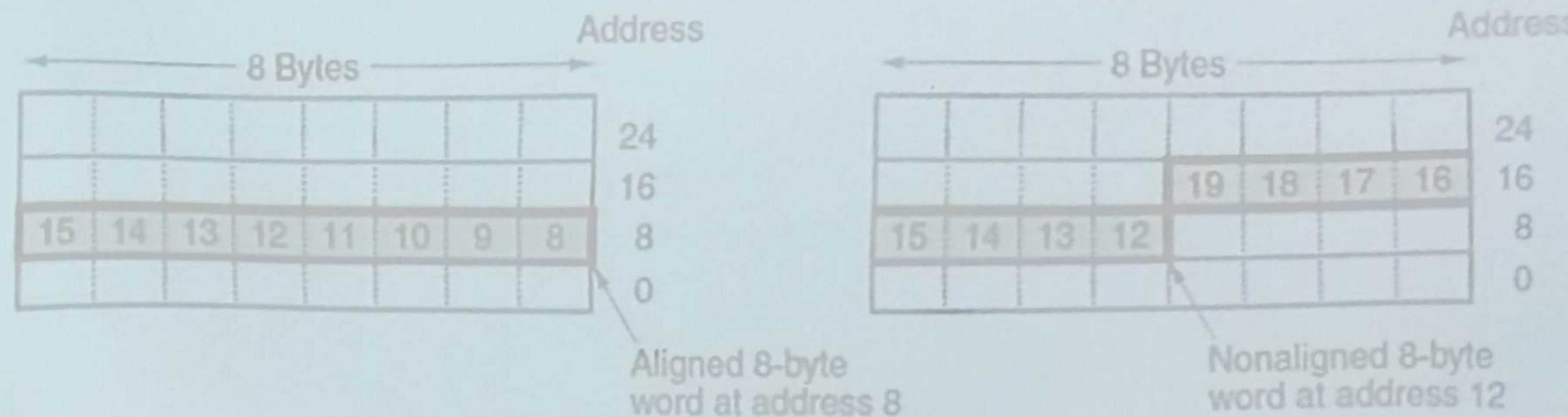
- È l'interfaccia tra i compilatori e l'hardware quindi il linguaggio che entrambi possono comprendere.



- Deve essere **retrocompatibile**: cioè essere in grado di far girare vecchi programmi.
- Normalmente ha due modalità operative:
 - Modalità kernel, per eseguire il SO e tutte le istruzioni.
 - Modalità utente, per eseguire i programmi utenti e non operazioni "sensibili" (come quelle che accedono alla cache).

I modelli di memoria

- Tutti i computer suddividono la memoria in celle adiacenti di un byte che sono a loro volta raggruppati in gruppi di 4 (32 bit) o 8 (64 bit).
- Le parole possono essere allineate all'indirizzo base (per



- I processori a livello ISA normalmente dispongono di uno spazio di memoria lineare (2^{32} o 2^{64}), talvolta alcuni hanno una suddivisione tra dati e istruzioni (questo rende più difficili gli attacchi di malware).

I registri

- Tutti i computer hanno dei registri visibili a livello ISA.
- Alcuni registri del livello sottostante non sono visibili (es. TOS e MAR).
- I registri ISA si possono suddividere in due categorie:
 - **Specializzati:** Program Counter, Stack Pointer e quelli visibili solo in modalità kernel (controllo della cache, la memoria, i dispositivi di I/O e altre funzionalità hardware).
 - **di uso generale:** sono utilizzati per memorizzare risultati temporanei delle variabili locali.

I registri

- Il **registro dei Flag (Program Status Word)** è un registro ibrido poiché è tra la modalità kernel e quella utente:
 - **N**, asserito quando il risultato è negativo.
 - **Z**, asserito quando il risultato è zero.
 - **V**, asserito quando il risultato causa un overflow.
 - **C**, asserito quando il risultato causa un riporto sul bit più significativo.
 - **A**, asserito quando c'è un riporto oltre il terzo bit (riporto ausiliario).
 - **P**, asserito quando il risultato è pari.

Overview del livello ISA del Core i7

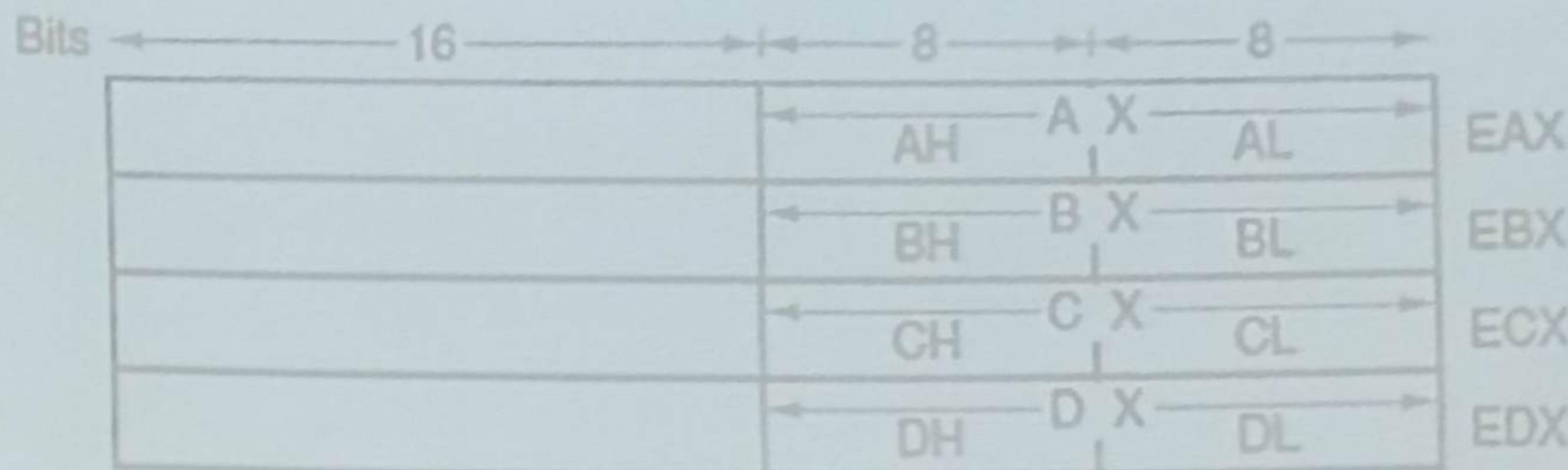
- Mantiene la compatibilità fino al 8086 e 8088 (anni '70), a loro volta basati sul 4004!
- Fino al 80286 il bus indirizzi era a 16 bit e si potevano indirizzare 16.384 segmenti da 64KB (invece che una memoria lineare da 2^{30} Byte).
- Dal x386 nasce una nuova architettura denominata **IA-32** (Intel Architecture-32 bit) su cui si fondano tutti gli attuali processori intel (x486, Pentium, Celeron, Xeon, Core duo e Core i7).
- I cambiamenti introdotti dal x386 sono le istruzioni MMX, SSE e SSE2 nate per applicazioni multimediali.
- Altra evoluzione importante è l'ampliamento del bus a 64 bit (x86-64).

Modalità operative di Intel Core i7

- Il core i7 ha tre diverse modalità operative:
 - **Modalità reale**, si comporta esattamente come un 8088 ma nel caso vengano eseguite istruzioni errate la macchina va in blocco.
 - **Modalità virtuale**, che permette di eseguire programmi 8088 in modo protetto e controllato da un vero SO.
 - **Modalità protetta**, si comporta come un core i7 con **4 privilegi** controllati dai bit del PSW:
 - Livello 0 o **modalità kernel**, usato dal SO
 - Livello 1 e 2, usati raramente
 - Livello 3, usati dai programmi utenti in modo protetto
 - La memoria è divisa in 16.384 segmenti da 64KB (dall'indirizzo 0 a $2^{32} - 1$), anche se la maggior parte dei SO supporta un solo segmento.

Registri di Intel Core i7

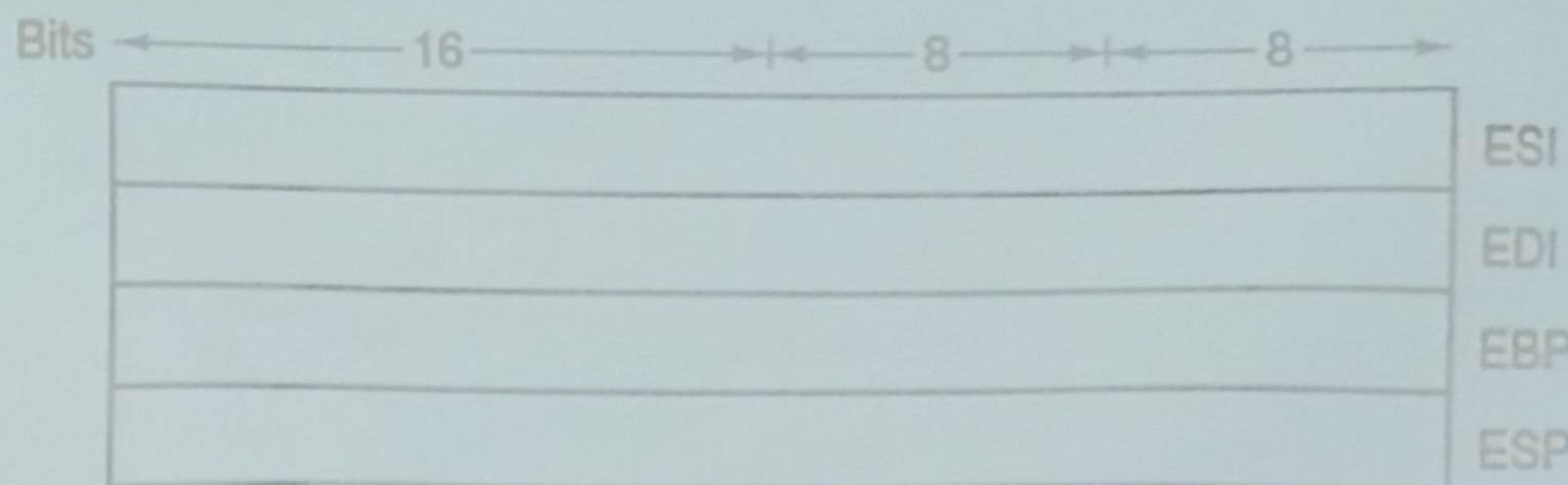
- Il core i7 ha 4 registri di uso generale (tutti a 32 bit):
 - EAX**, per le operazioni aritmetiche.
 - EBX**, usato come puntatore a indirizzi di memoria.
 - ECX**, utilizzato come contatore nei cicli.
 - EDX**, utilizzato nelle moltiplicazioni/divisioni durante le quali contiene con **EAX** prodotti/dividendi a 64 bit.



- Tutti questi registri possono essere utilizzati a 16 o 8 bit (i registri a 16, prefisso E per Esteso, furono introdotti dal x386).

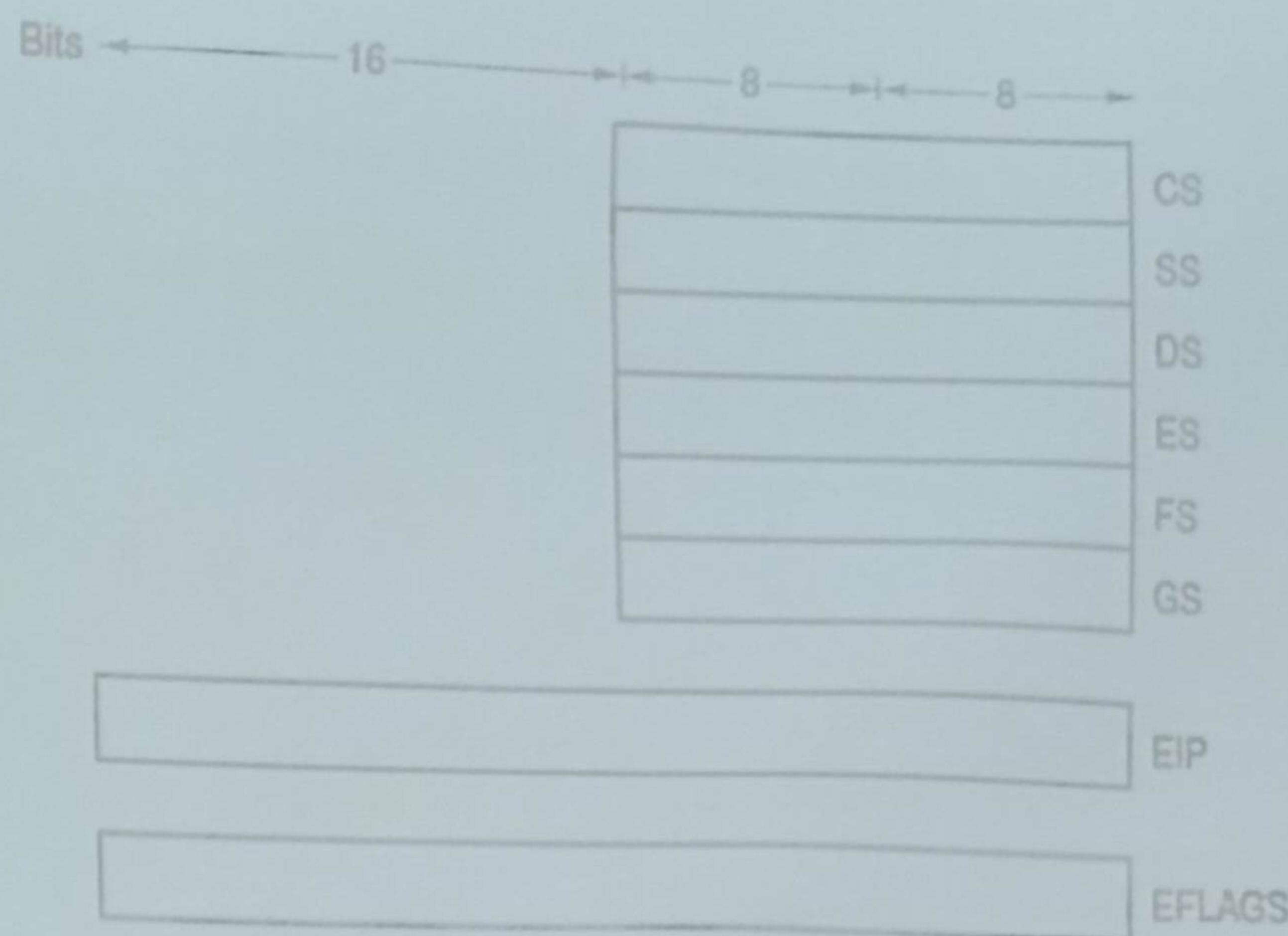
Registri di Intel Core i7

- Ulteriori quattro registri a 32 bit con caratteristiche specifiche:
 - **ESI**, puntatore in memoria alla stringa sorgente.
 - **EDI**, puntatore in memoria alla stringa destinazione.
 - **EBP**, referenzia l'indirizzo base del record di attivazione corrente (in analogia con il registro LV della LJVM).
 - **ESP**, puntatore allo stack (come SP di LJVM).



Registri di Intel Core i7

- I registri segmento (tutti a 16 bit) che derivano dalla compatibilità dell'indirizzamento a 16 bit dell'8088: **CS, SS, DS, ES, FS e GS**.
- Il program counter **EIP** (Extended Instruction Pointer)
- L'insieme dei bit dei flag della program status word, **EFLAGS**.



I tipi di dato

- I tipi di dato sono raggruppabili in:
 - **Numerici**
 - Interi
 - $2^{n-1} - 1 + 2^{n-1}$ con segno
 - $0 \div 2^n$ senza segno
 - Reali, si usa la rappresentazione in virgola mobile.
 - Boolean, si usa la rappresentazione numerica
 - 0 false
 - $\neq 0$ true
 - **Non numerici**
 - Caratteri (ASCII e UNICODE).
 - BitMap.
 - Puntatore (es. SP, PC,LV, CPP della Mic-1).

Rappresentazione del tipo intero con segno

1) con bit di segno:

b_2	b_1	b_0	
0	0	0	4
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	0
1	0	1	-1
1	1	0	-2
1	1	1	-3

2) con complemento alla base: si inverte il numero
(complemento a uno) e si somma uno:

$$\text{Es/ } -15_{10} = ?_2$$

$$15_{10} = 00001111_2 \quad \rightarrow \quad \begin{array}{r} 11110000 \\ + 1 \\ \hline 11110001 \end{array}$$

$$-15_{10} = 11110001_2$$

b_2	b_1	b_0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

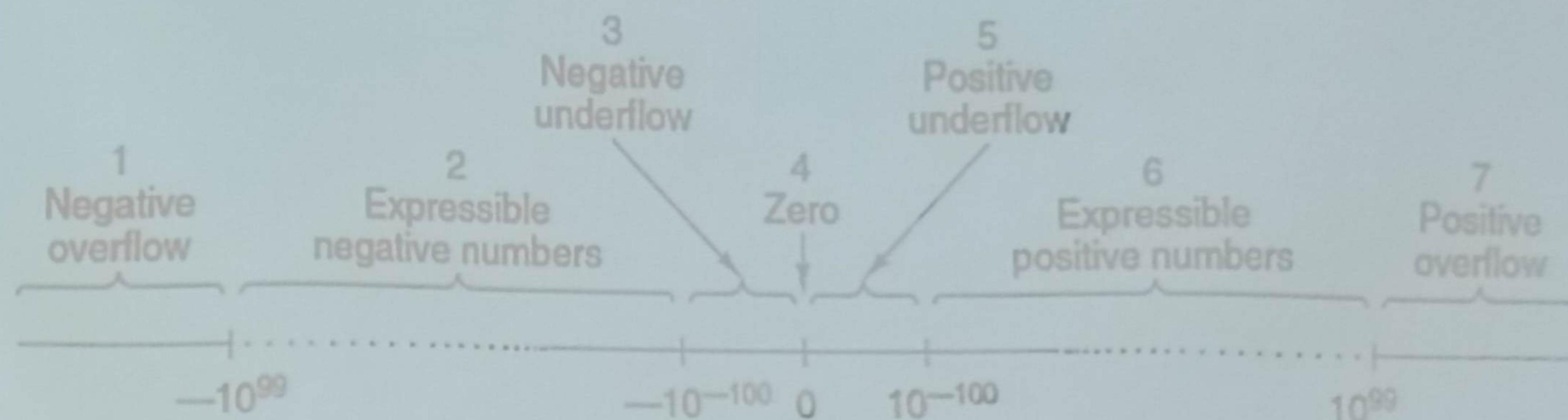
Rappresentazione del tipo reale

- Si utilizza la notazione scientifica:

$$N = f \times 10^e$$

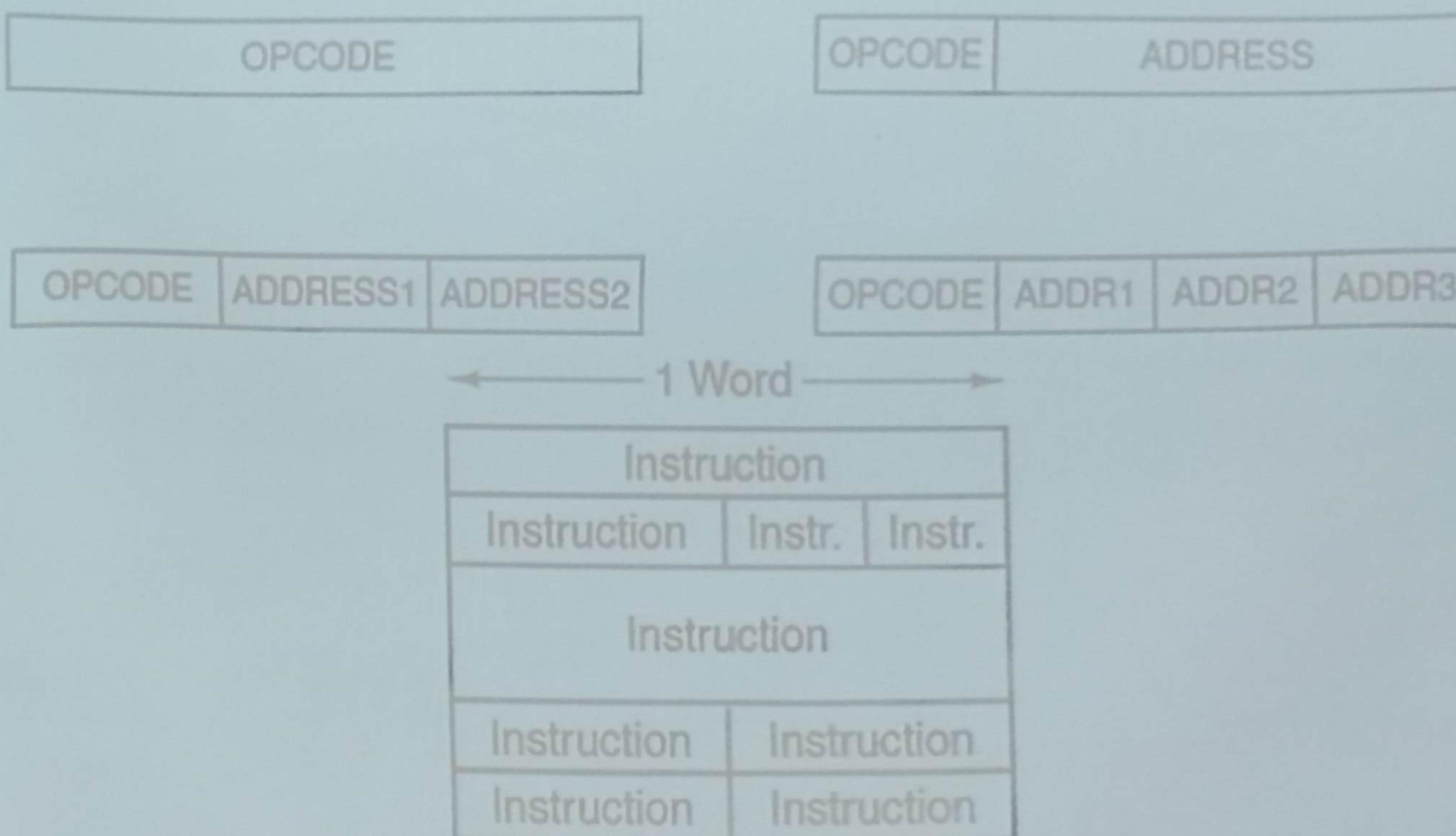
- **f** è chiamata mantissa o frazione
- **e** è chiamato esponente

- Il range di valori dipende dal numero di bit utilizzati per codificare la frazione ed esponente



Formati di istruzioni

- Una istruzione si compone di:
 - codice operativo dell'istruzione (**Opcode**)
 - indirizzi di riferimento degli operandi (opzionali)



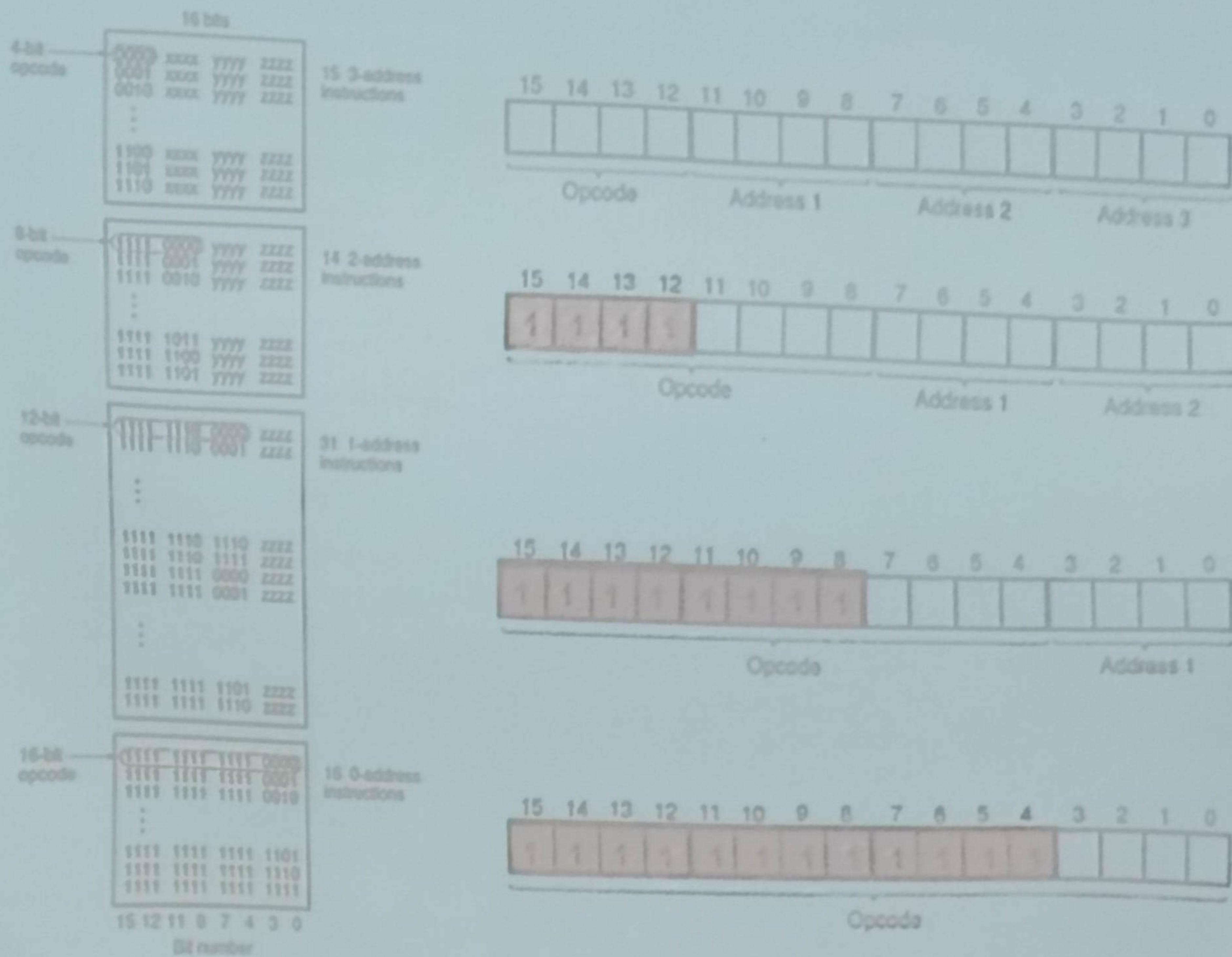
Criteri progettuali dei formati di istruzioni

- Istruzioni corte sono preferibili alle lunghe per vari ordini di ragioni:
 - Banalmente, permette di avere programma più piccoli a parità di istruzioni.
 - Permette di memorizzare maggiori quantità di istruzioni nelle cache dei processori che, notoriamente, hanno una larghezza di banda (bit/sec) limitata rispetto alla capacità di calcolo dei processori.
- Minimizzare troppo la dimensione può causare la difficoltà nella decodifica.

Criteri progettuali dei formati di istruzioni

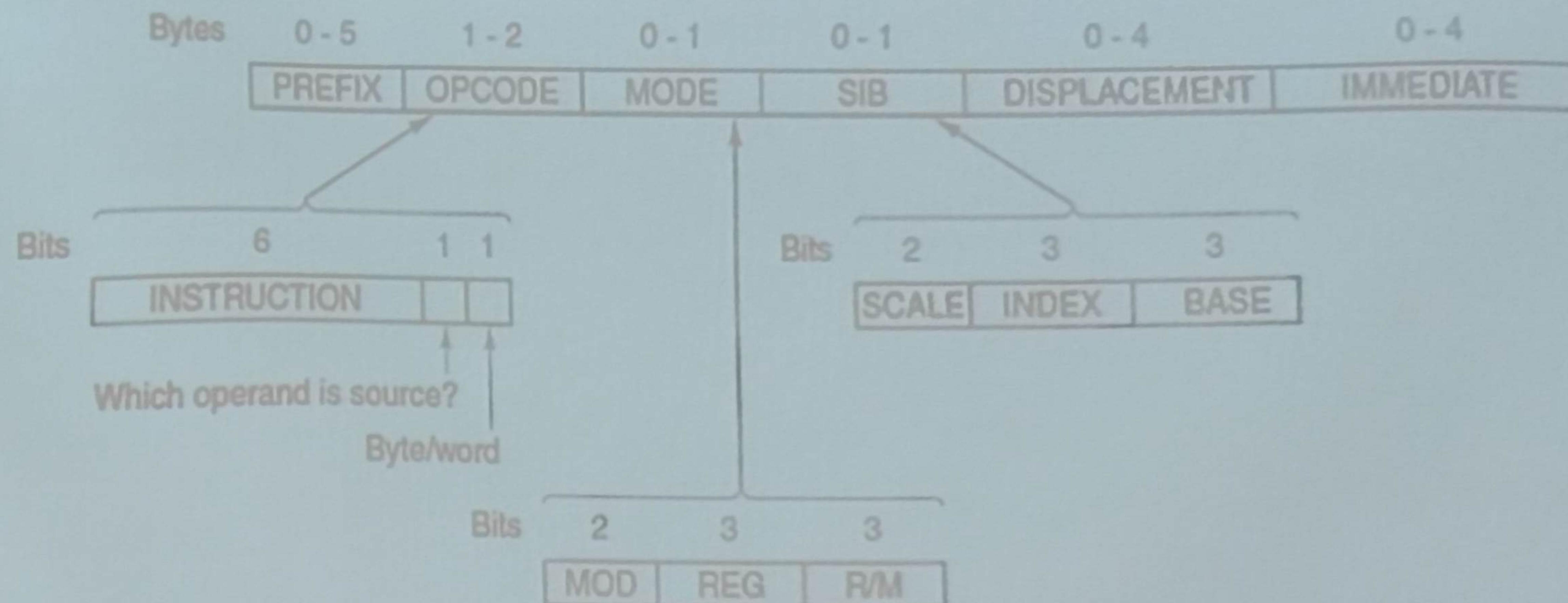
- In base al sistema di codifica delle istruzioni, prevedere uno spazio sufficiente ad esprimere tutte le operazioni desiderate.
- Il numero di bit da utilizzare nell'indirizzamento della memoria, uno spazio di indirizzamento ampio conduce ad istruzioni lunghe.

Codice operativo espandibile



Formati delle istruzioni del Core i7

- Esistono 6 campi (di cui 5 opzionali) di lunghezza variabile.
- Uno dei due operandi è sempre un registro, l'altro un registro o in memoria.
- MODE stabilisce la modalità di indirizzamento.
- l'indirizzo di memoria è l'offset di un segmento.
- **SIB** (Scale, Index e Base) è un bit supplementare.



Conclusioni

- Introdotto il livello di microarchitettura ed analizzato il funzionamento interno.
- Approfondite le problematiche sui tipi di dato.
- Analizzato funzionamento dell'Intel Core i7.
- Approfondito i criteri progettuali del modello ISA.