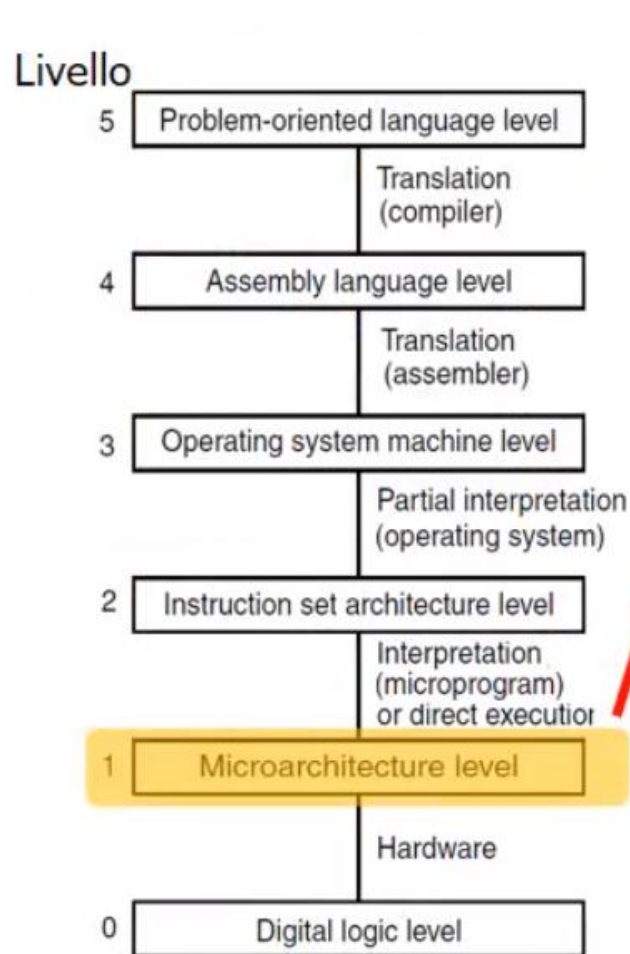


**IL LIVELLO DI
MICROARCHITETTURA
E
MACROARCHITETTURA
(Prima Parte)**

Argomenti

- IL LIVELLO DI MICROARCHITETTURA
 - Un esempio di microarchitettura
 - modello di esecuzione
 - data path (o percorso dati)
 - formato delle microistruzioni
 - 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

IL LIVELLO DI MICROARCHITETTURA



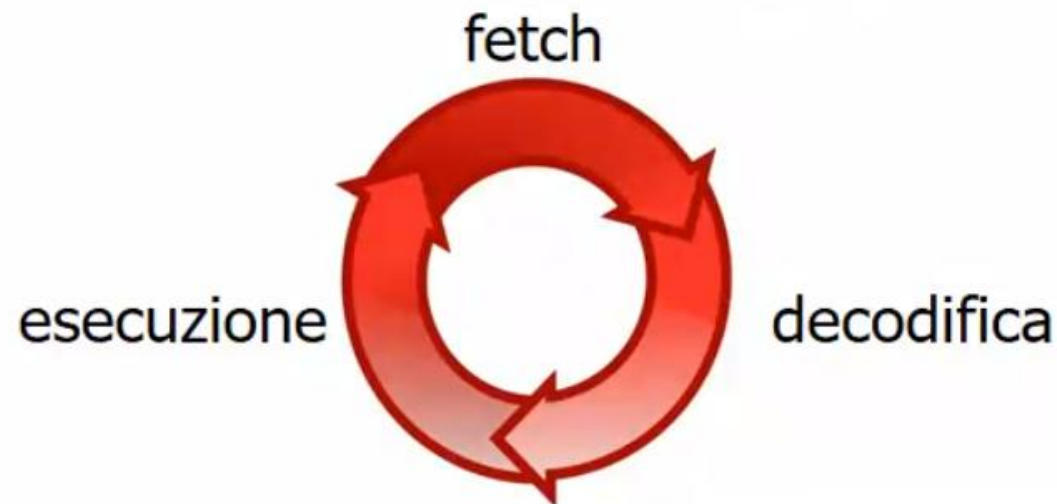
- Il livello di Microarchitettura fornisce una visione astratta per il livello ISA (Instruction Set Architecture).
- Dipende non solo dall'ISA ma anche dagli obiettivi di costo e di prestazioni della macchina che si intende costruire.

Esempio di microarchitettura

- 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 si compone di:
 - **Codice operativo** (Opcode) che identifica il tipo di istruzione.
 - Operando/i su cui si applicherà l'istruzione.

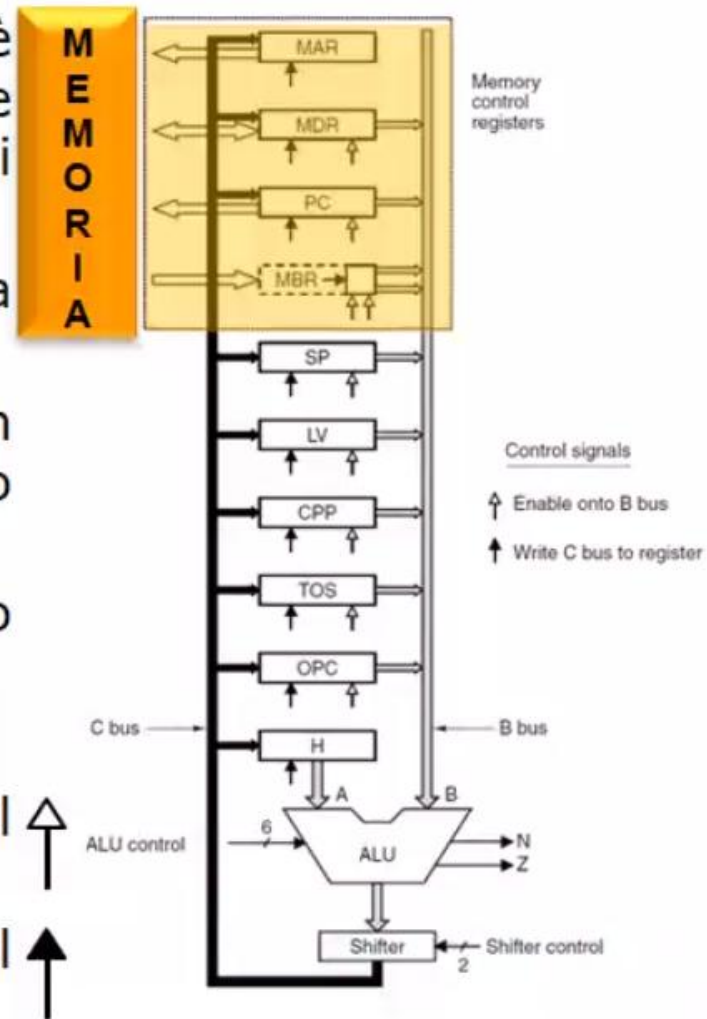
Il modello di esecuzione

- 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.



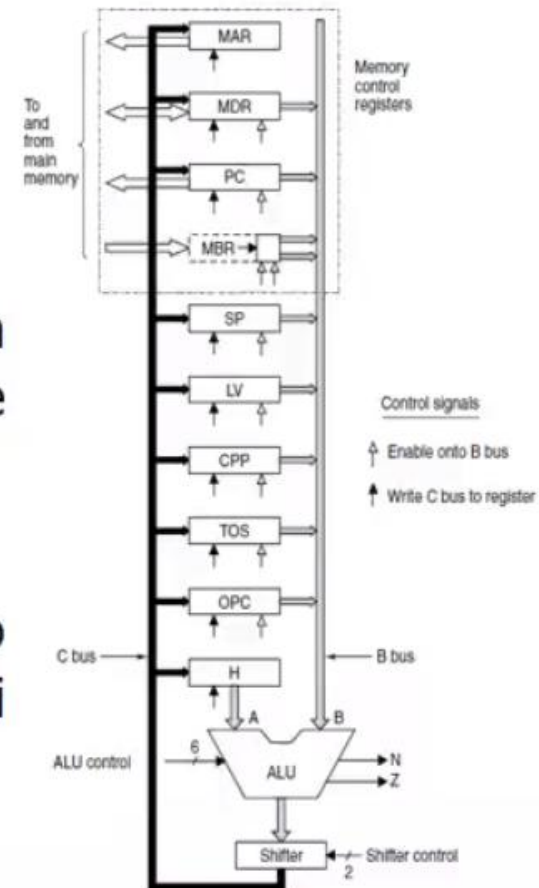
Il data path (o percorso dati)

- 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.



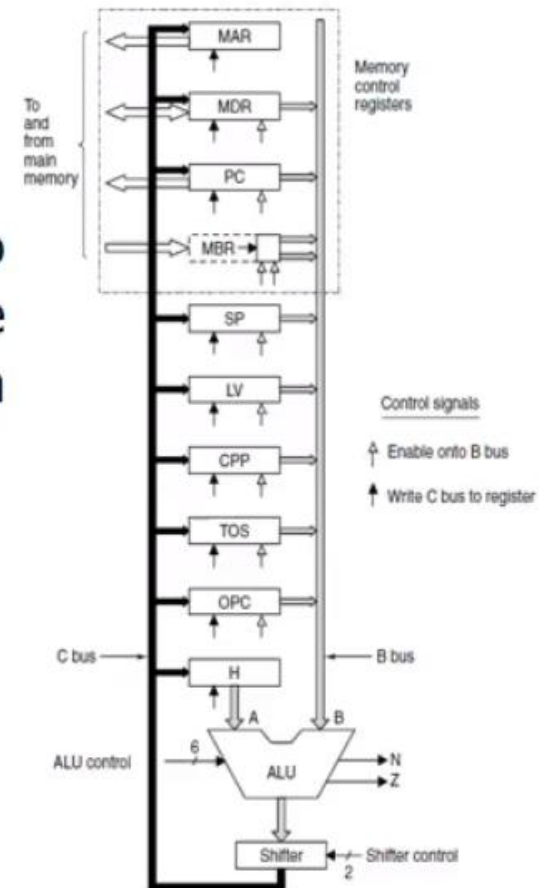
Il data path (o percorso dati)

- 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 Variable (**LV**), il riferimento nello stack alla base delle variabili locali.



Il data path (o percorso dati)

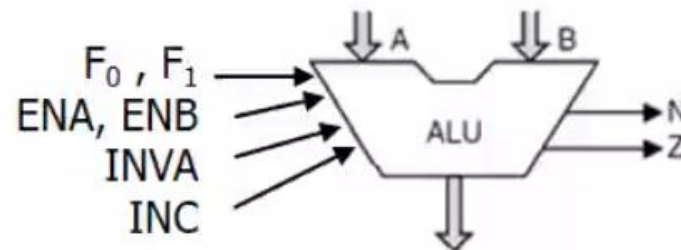
- Gli altri registri sono:
 - Constant Pool (**CPP**).
 - Top word On the Stack (**TOS**).
 - Op Code register (**OPC**): registro temporaneo, può contenere l'ultima istruzione eseguita prima di un salto.
 - Holding (**H**).



Arithmetic Logic Unit

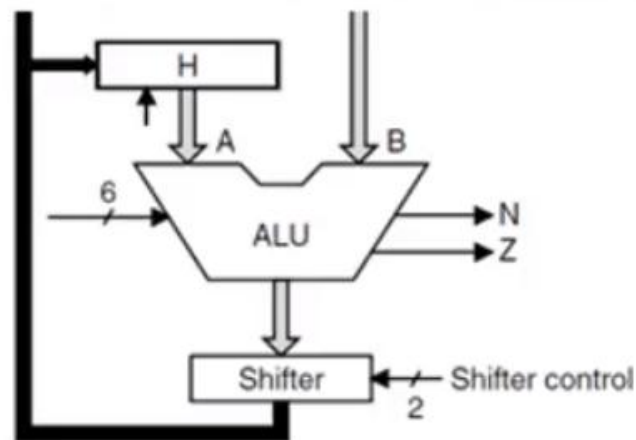
- La ALU ha sei linee di controllo:
 - F_0 e F_1 selezionano il tipo di funzione.
 - EN_x , abilita il valore della variabile x altrimenti lo annulla.
 - $INVA$, esegui una sottrazione della variabile A .
 - INC , incrementa.

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 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



Arithmetic Logic Unit

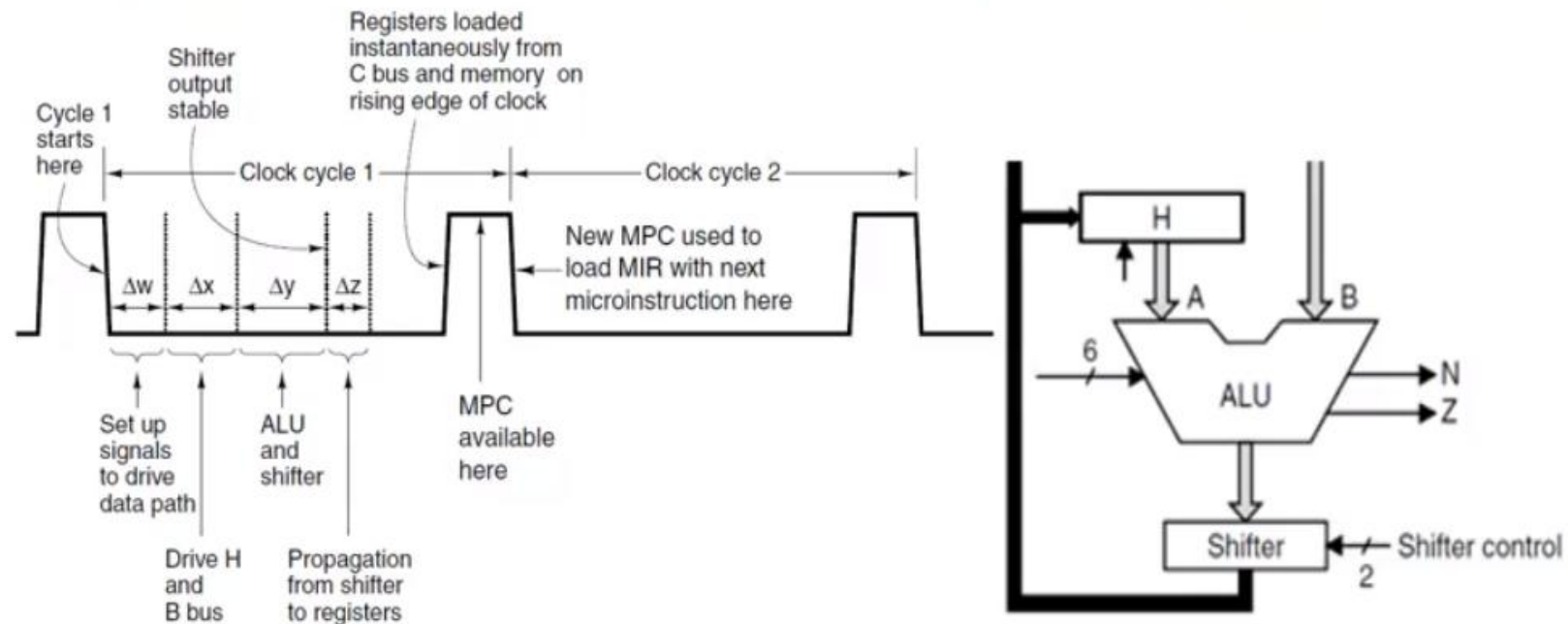
- La ALU agisce su due operandi: uno viene dal registro **H** e l'altro dal bus **B**.
- È possibile "spostare" l'operando da B ad A, applicando la funzione che restituisce l'operando B e successivamente memorizzando il risultato del bus **C** in **H**.



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

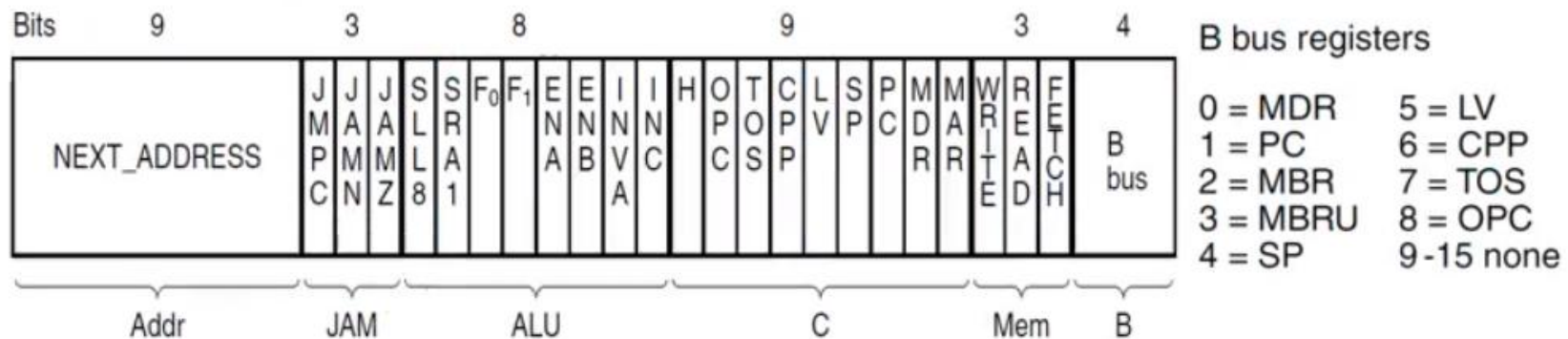
Incremento di un registro (un ciclo di clock)

- passo 1: si pone il valore del registro sul bus **B**
- passo 2: si disabilita l'operando A e si incrementa B
- passo 3: non si fa alcun scorrimento
- passo 4: si riscrive il risultato nel registro originario



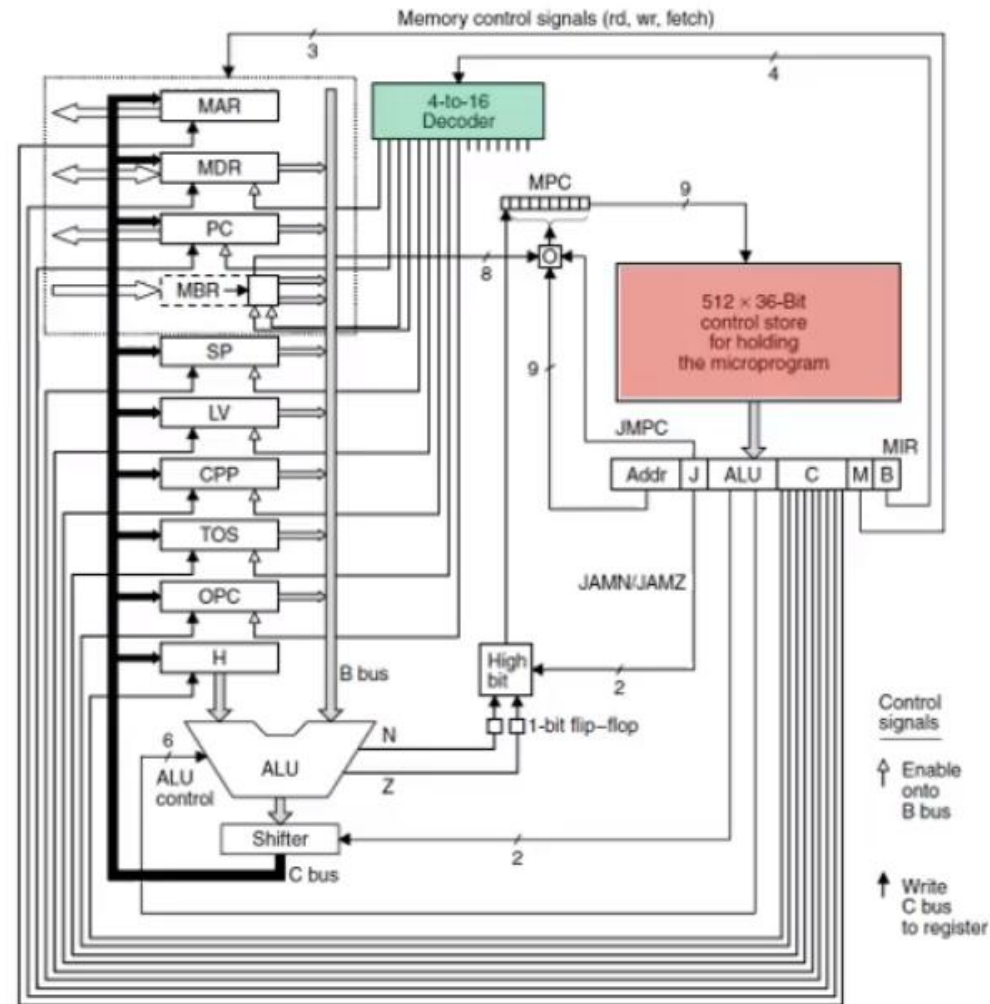
Il formato delle microistruzioni

- **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.



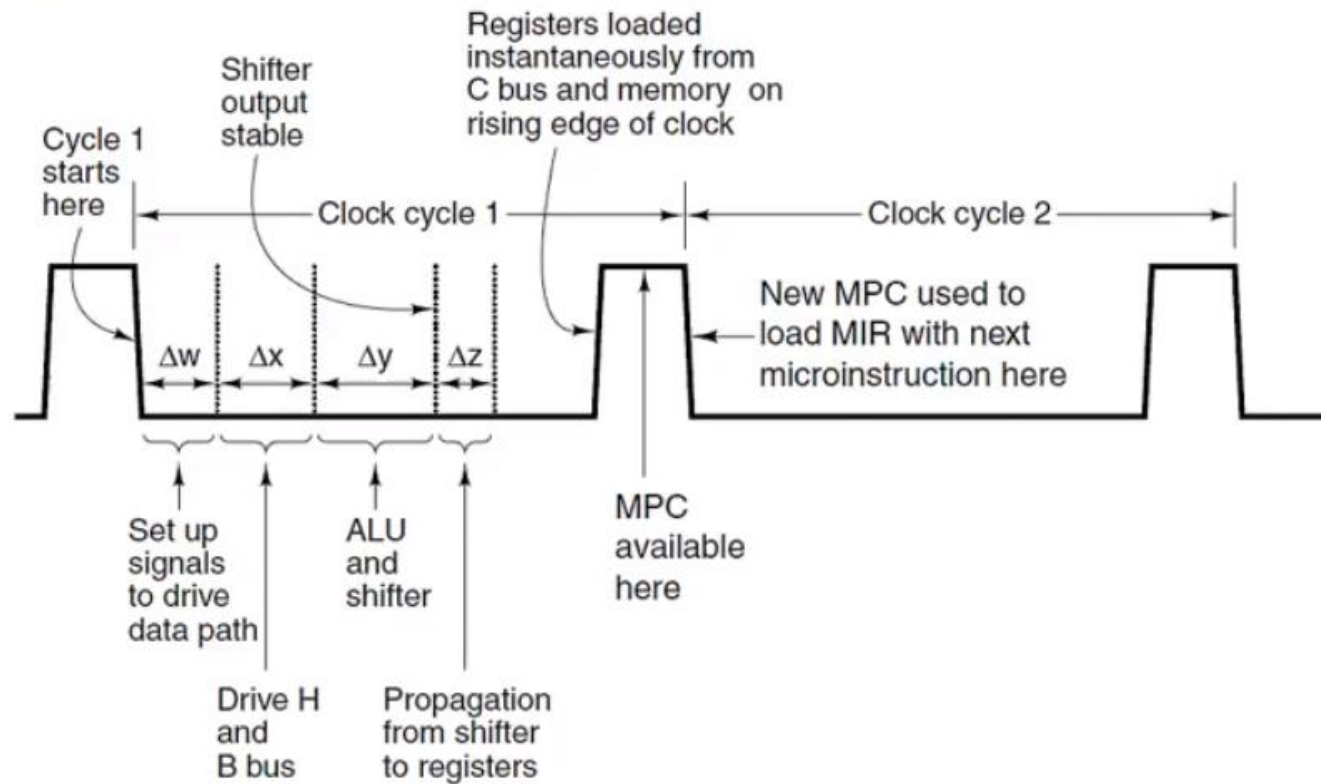
Microarchitettura Mic-1

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



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

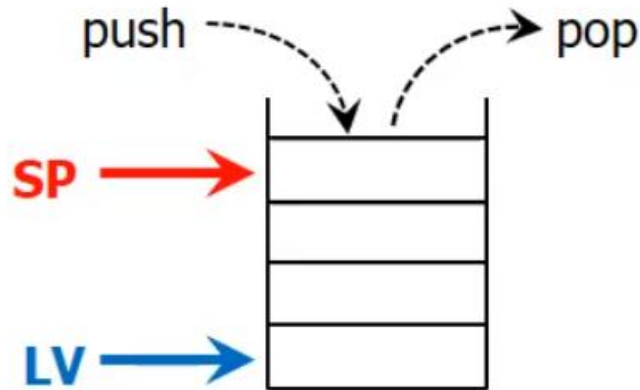


Un esempio di ISA: IJVM

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

Lo stack

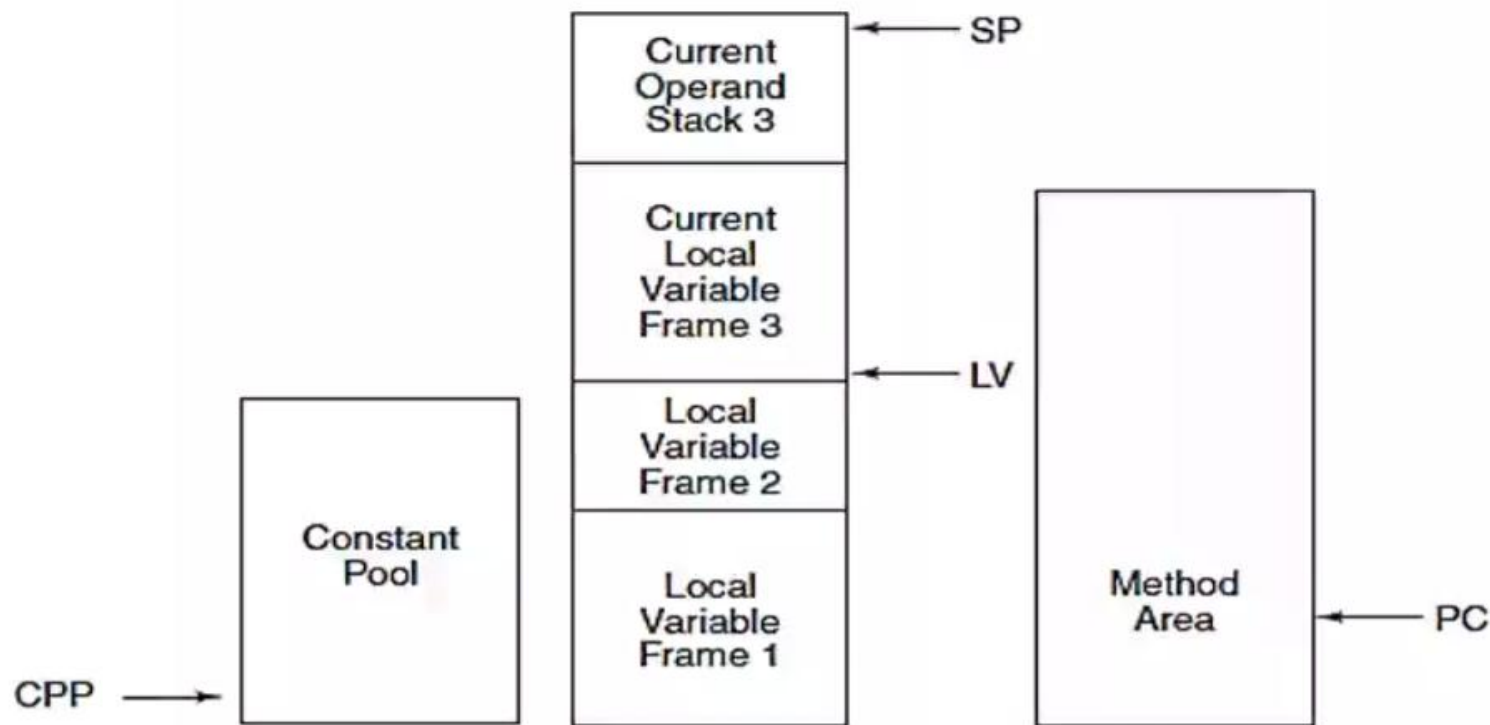
- È una struttura dati utilizzata per memorizzare lo stato di una procedura che segue la filosofia LIFO (Last-In-First-Out).



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

Il modello della memoria JVM

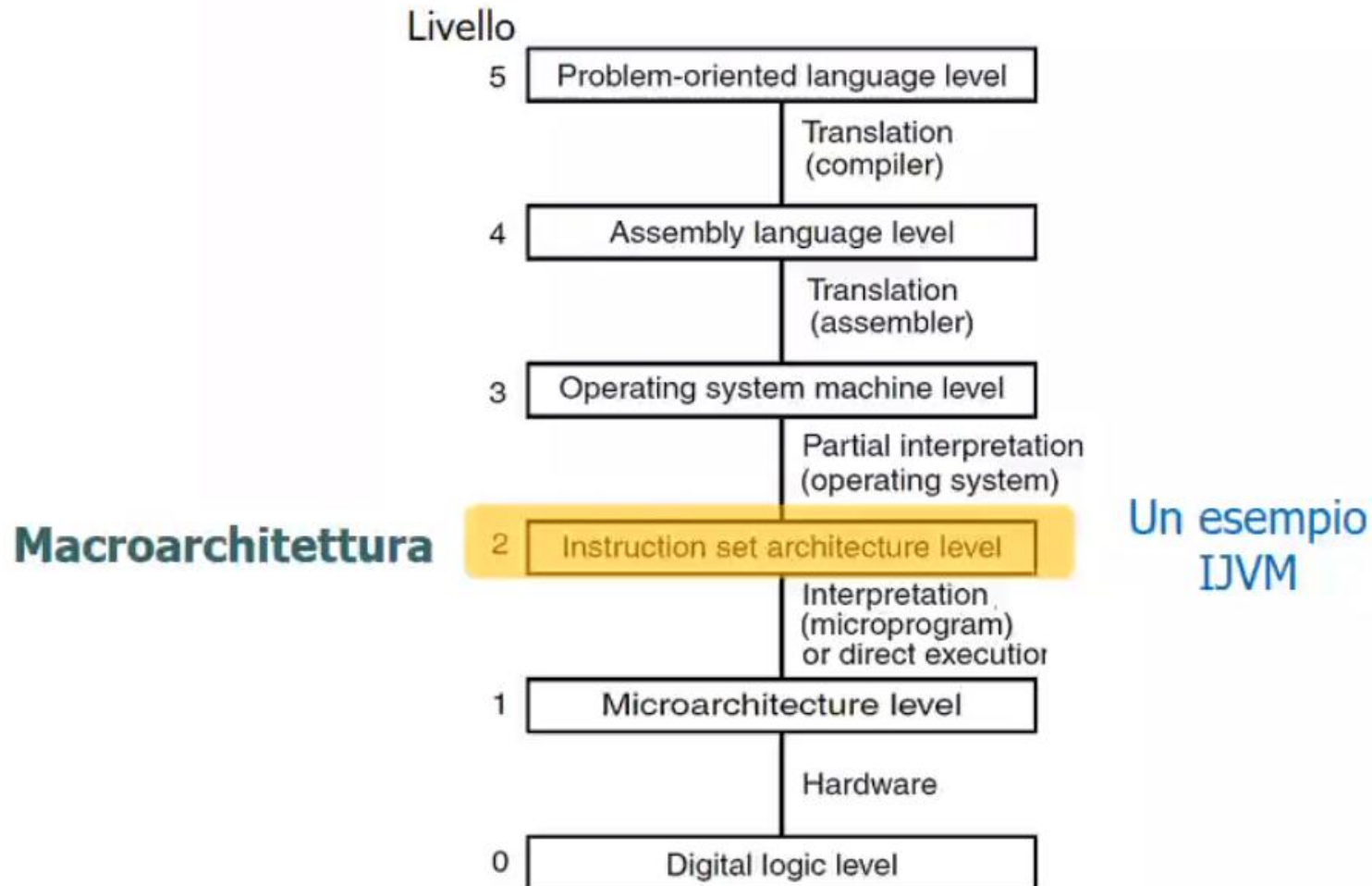
- La memoria può essere vista come un vettore di:
 - 4.294.967.296 Byte (4 GB)
 - 1.073.741.824 parole di 4 Byte



Insieme delle istruzioni della JVM

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

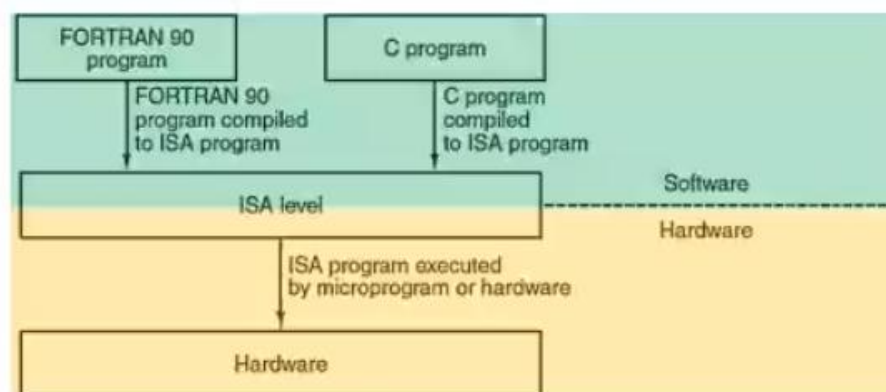


Overview del livello ISA

- 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

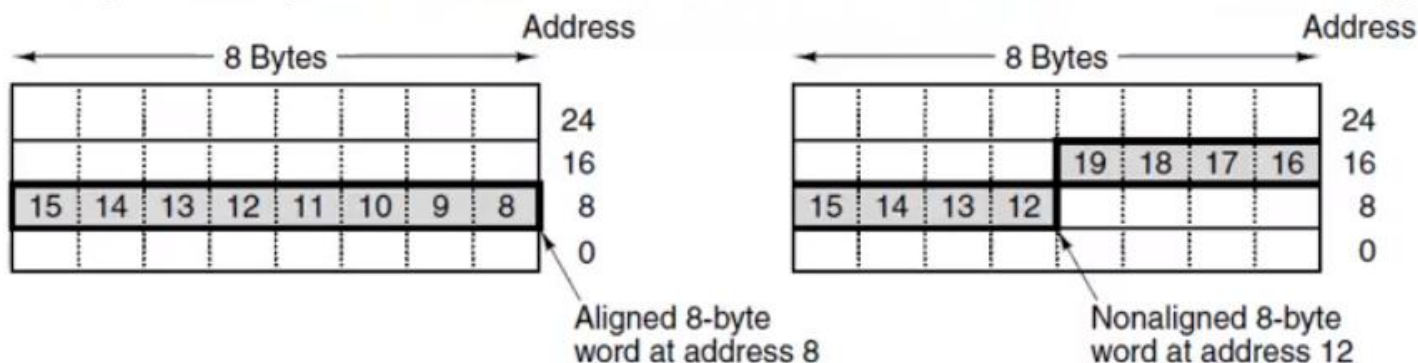
- È l'interfaccia tra i compilatori e l'hardware quindi il linguaggio che entrambi possono comprendere.



- 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

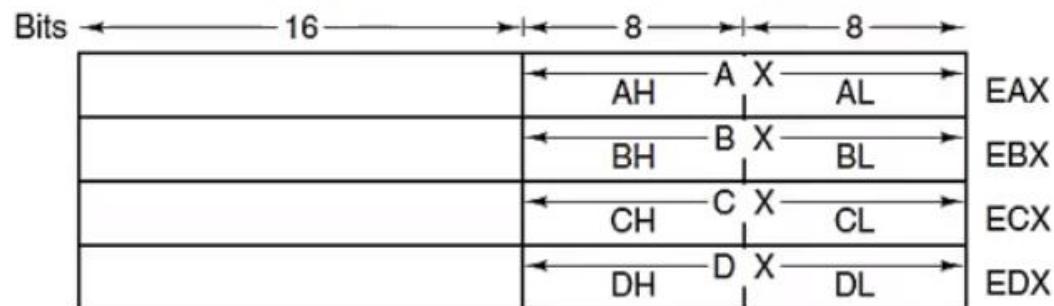
- Mantiene la compatibilità fino al 8086 e 8088 (anni '70), a loro volta basati sul 4004!
- Fino al 80286 il bus indirizzi era a 16 bit e si potevano indirizzare 16.384 segmenti da 64KB (invece che una memoria lineare da 2^{30} Byte).
- Dal x386 nasce una nuova architettura denominata **IA-32** (Intel 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

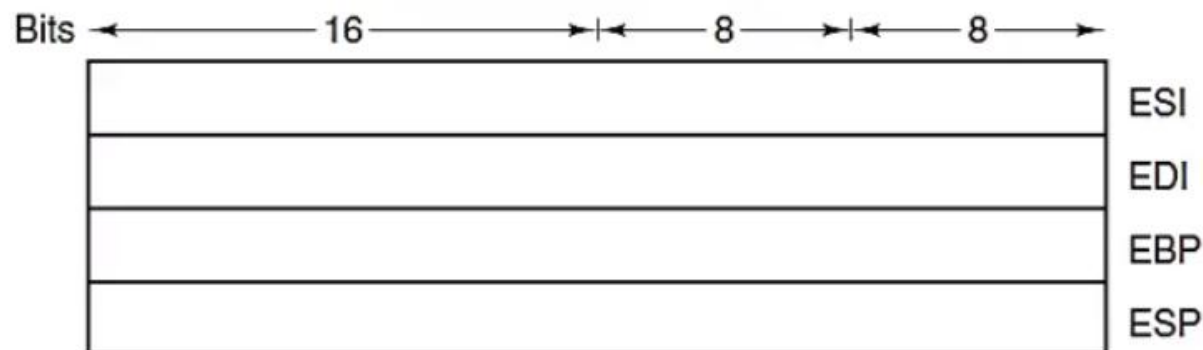
- Il core i7 ha 4 registri di uso generale (tutti a 32 bit):
 - **EAX**, per le operazioni aritmetiche.
 - **EBX**, usato come puntatore a indirizzi di memoria.
 - **ECX**, utilizzato come contatore nei cicli.
 - **EDX**, utilizzato nelle moltiplicazioni/divisioni durante le quali contiene con **EAX**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

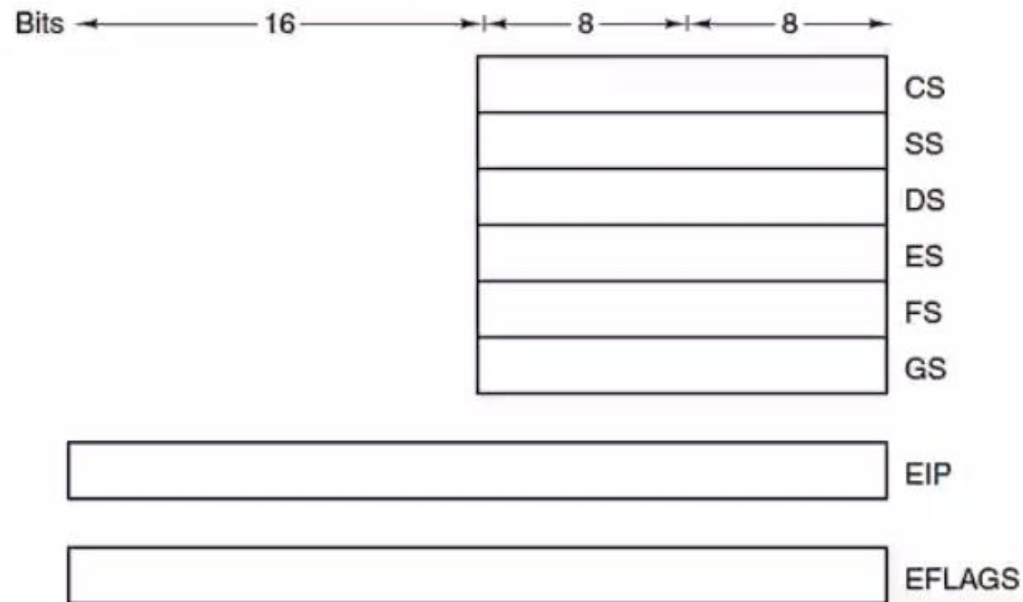
Registri di Intel Core i7

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



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 $\left\{ \begin{array}{ll} -2^{n-1} \div 2^{n-1} & \text{con segno} \\ 0 \div 2^n & \text{senza segno} \end{array} \right.$
 - Reali, si usa la rappresentazione in virgola mobile.
 - Boolean, si usa la rappresentazione numerica $\left\{ \begin{array}{ll} 0 & \text{false} \\ \neq 0 & \text{true} \end{array} \right.$
 - **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:

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

$$15_{10} = 00001111_2 \quad \longrightarrow \quad 11110000 + 1 =$$

$$11110001$$

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

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

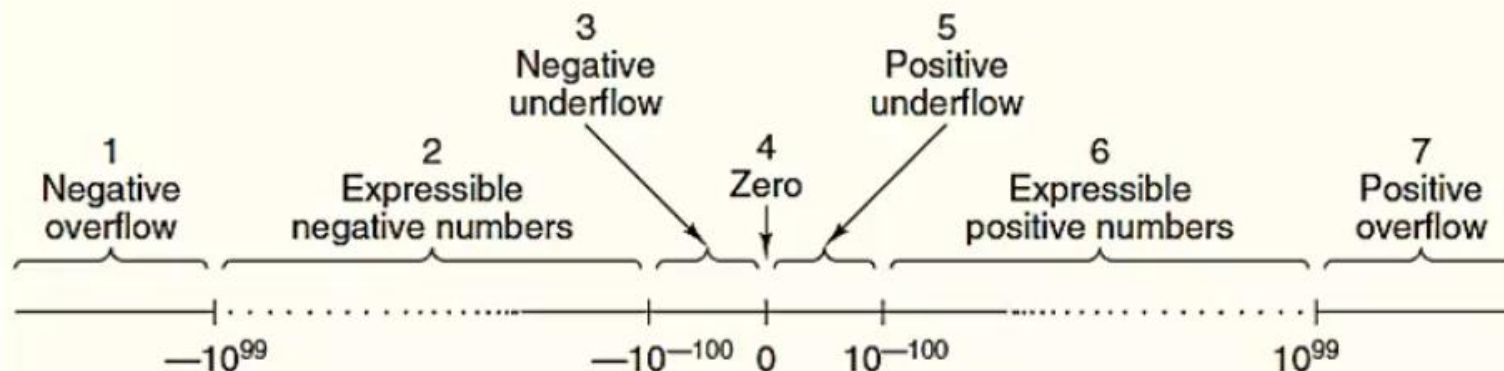
Rappresentazione del tipo reale

- Si utilizza la notazione scientifica:

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

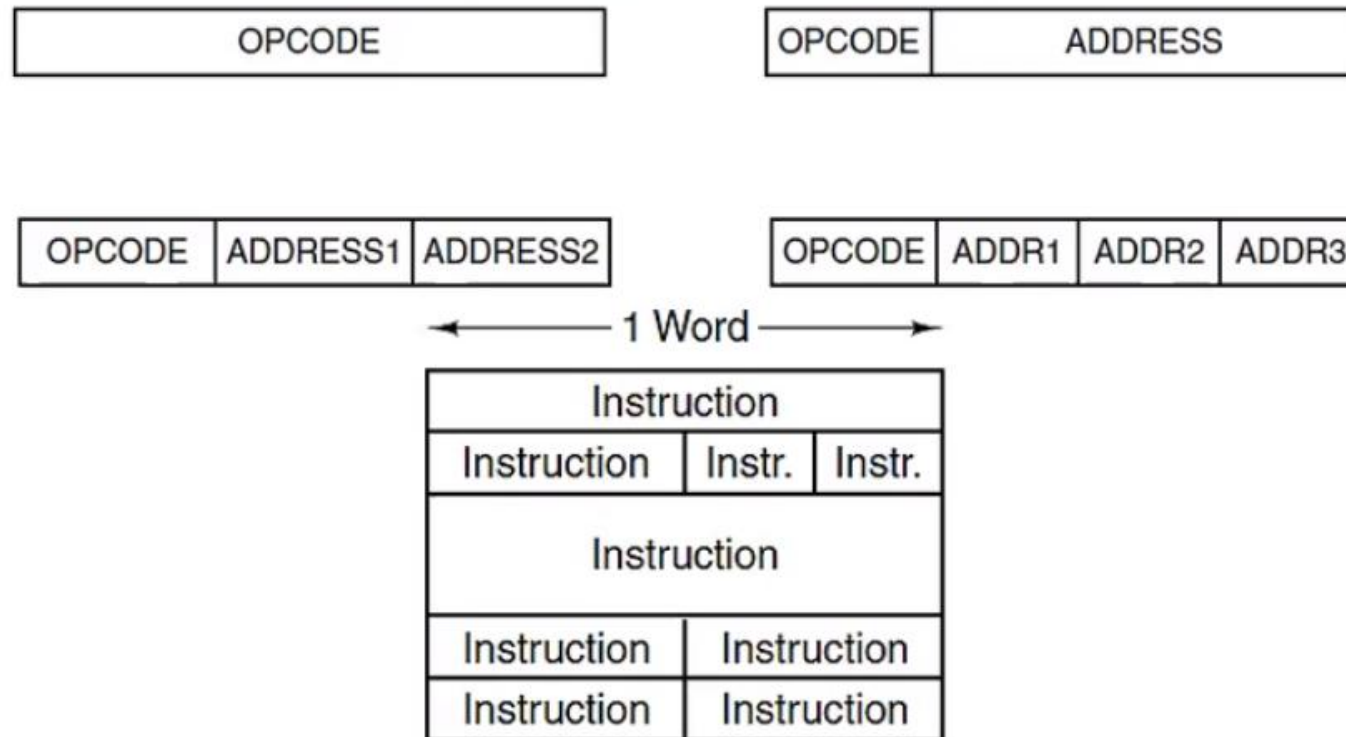
- **f** è chiamata mantissa o frazione
- **e** è chiamato esponente

- 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)



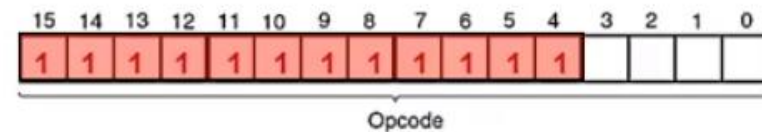
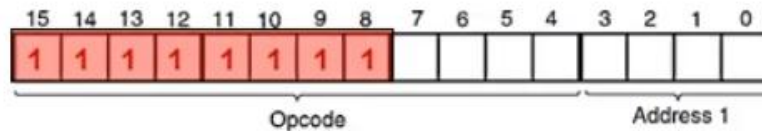
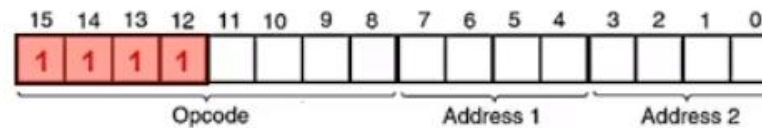
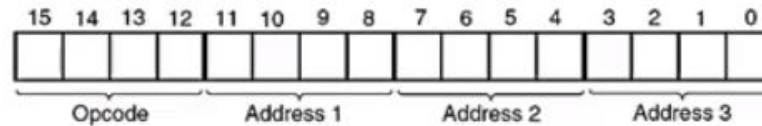
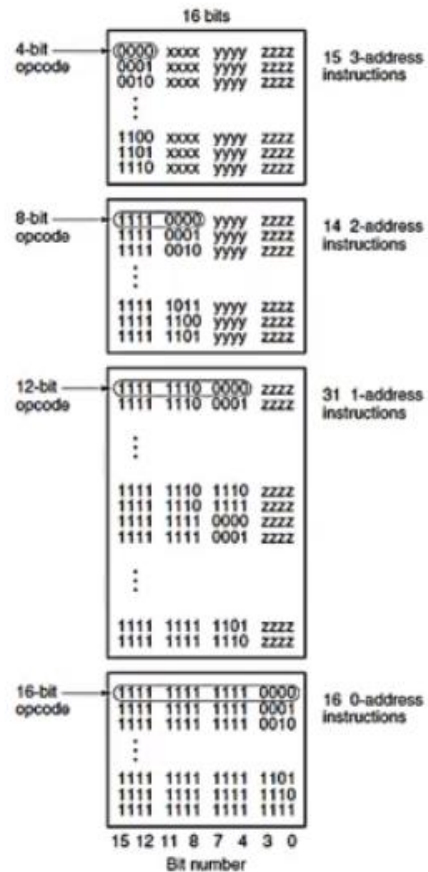
Criteri progettuali dei formati di istruzioni

- 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

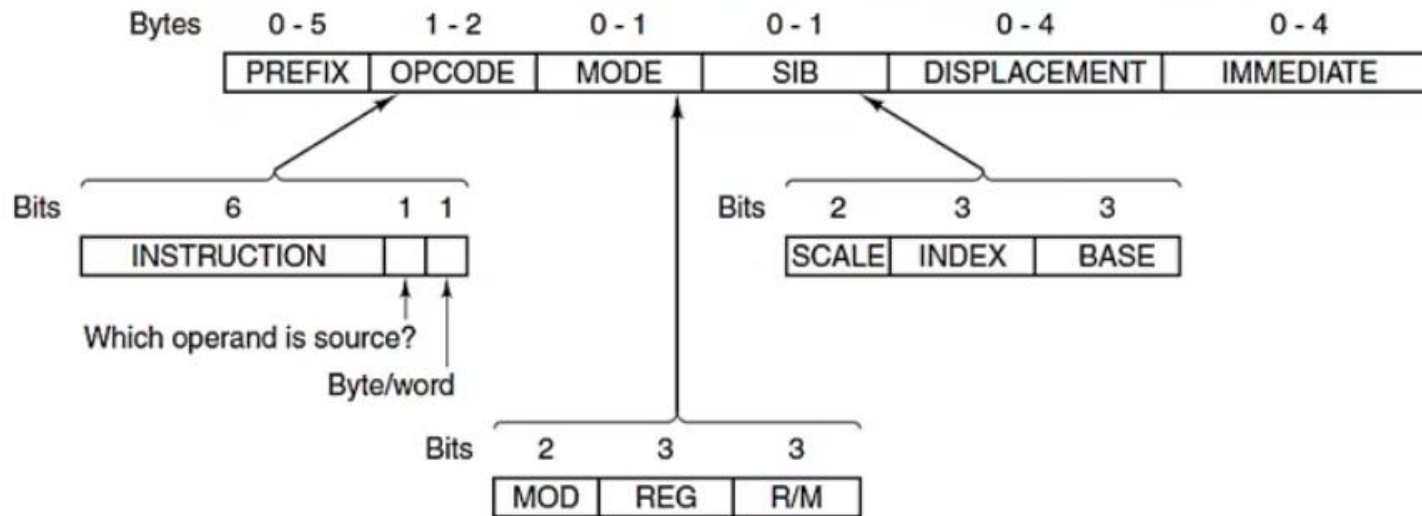
- In base al sistema di codifica delle istruzioni, prevedere uno spazio sufficiente ad esprimere tutte le operazioni desiderate.
- Il numero di bit da utilizzare nell'indirizzamento della memoria, uno spazio di indirizzamento ampio conduce ad istruzioni lunghe.

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.



Modalità di Indirizzamento

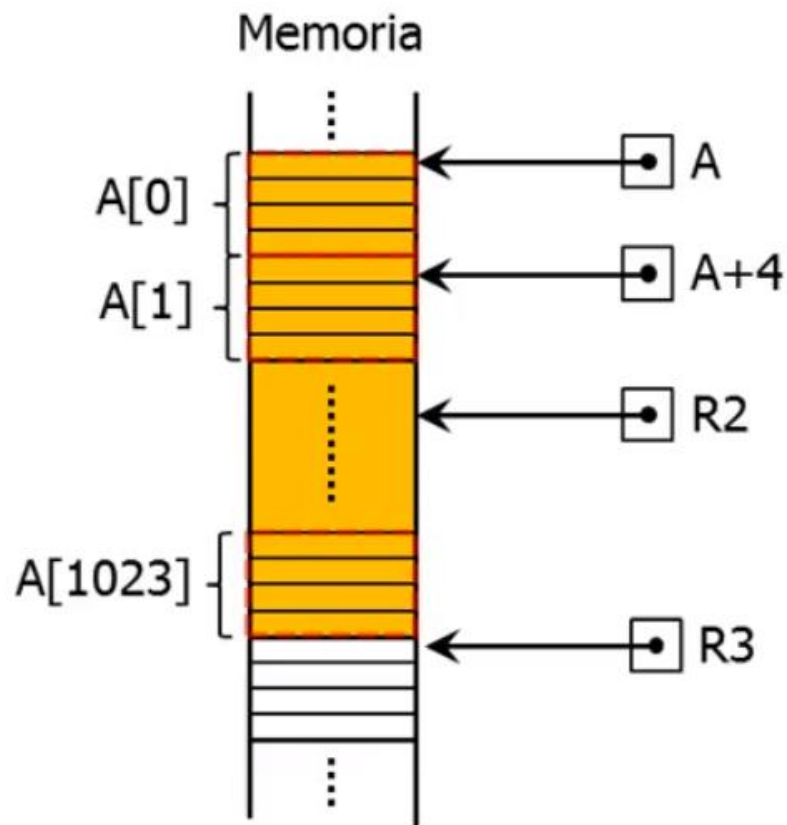
- **Immediato:** il valore dell'operando è nell'istruzione.
- **Diretto:** l'istruzione contiene l'indirizzo di memoria completo dell'operando (variabili globali).
- **Indiretto:** l'indirizzo di memoria fornito contiene l'indirizzo dell'operando.
- **A registro:** si specifica un registro che contiene l'operando.

Modalità di Indirizzamento

- **Indiretto a registro:** il registro specificato contiene l'indirizzo dell'operando.
- **Indicizzato:** l'indirizzo è dato da una costante più il contenuto di un registro.
- **A registro base:** viene sommato a tutti gli indirizzi il contenuto di un registro.
- **A stack:** l'operando è sulla cima dello stack (o ci deve andare).

Esempio

- Programma che calcola la somma degli elementi di un array (1024 interi) che iniziano all'indirizzo A.



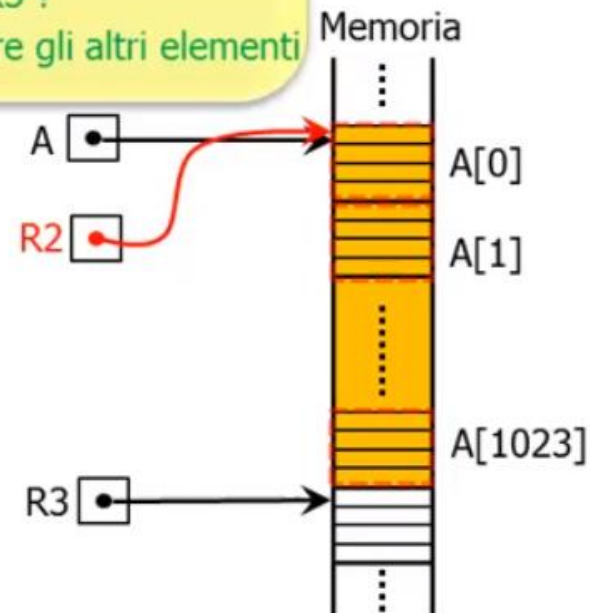
- un intero occupa 4 byte.
- la somma viene accumulata in R1.
- R2 punta all'elemento corrente.
- R3 è la prima posizione esterna all'array.

R1

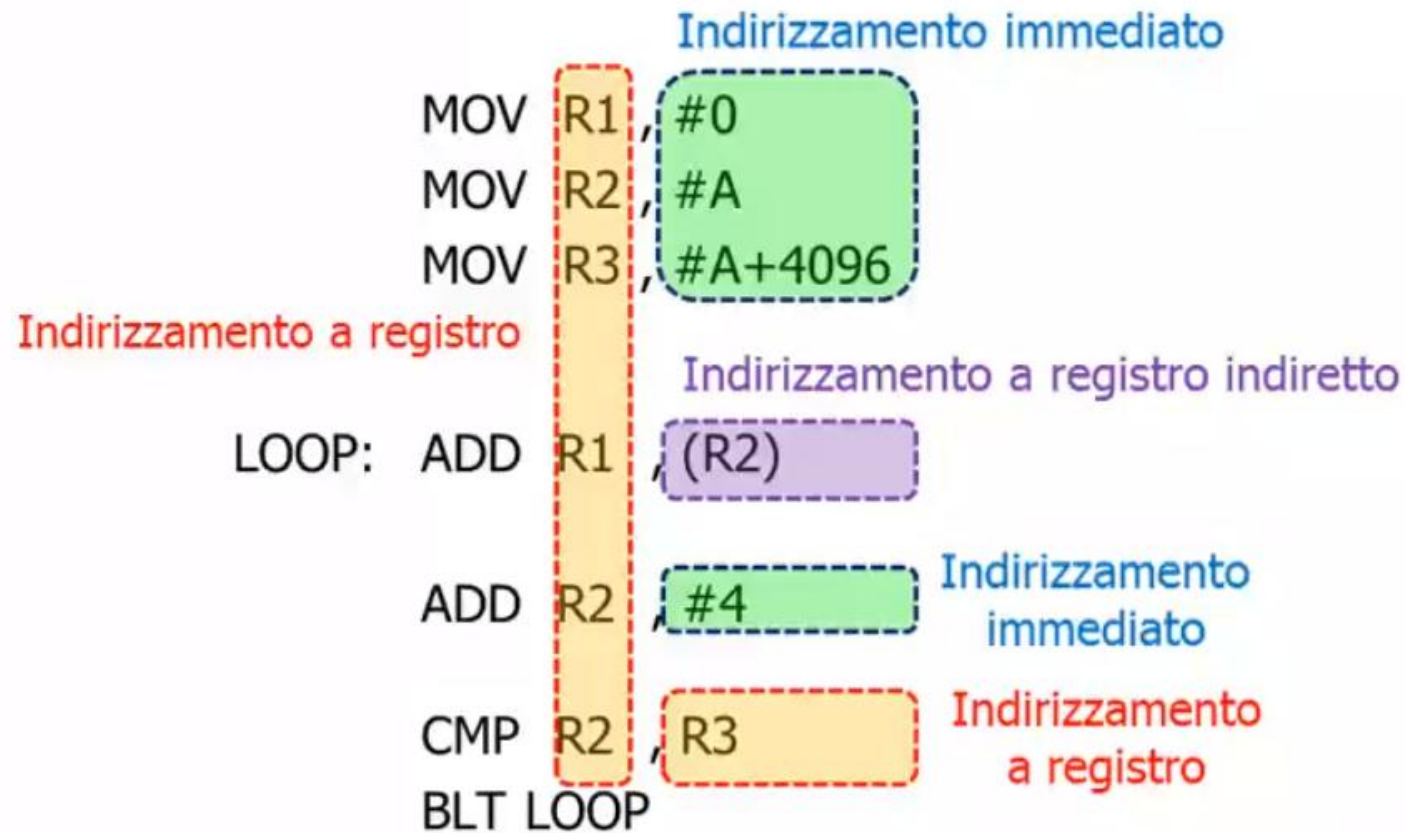
0

Esempio di indirizzamento a registro indiretto

```
MOV R1,#0      ; R1 registra la somma parziale, inizialmente 0
MOV R2,#A      ; in R2 c'è l'indirizzo dell'array A
MOV R3,#A+4096 ; in R3 c'è l'indirizzo oltre l'array A
LOOP: ADD R1,(R2) ; R1=R1+ (il valore puntato da R2)
      ADD R2,#4   ; incrementa R2 di una parola (4 byte)
      CMP R2,R3   ; R2 ha raggiunto l'indirizzo R3 ?
      BLT LOOP    ; se R2 < R3 occorre sommare gli altri elementi
```



Istruzioni al microscopio



Esempio di codice auto-modificante

- Questo programma utilizza una istruzione che si modifica durante l'esecuzione:

```
MOV R1,#0          ; accumula la somma in R1, inizialmente a 0
MOV R2,#4           ; R2 = 4
MOV R3,#A+4096      ; R3 = indirizzo prima parola dopo array A
LOOP: ADD R1, A+R2    ; somma ad A un offset crescente
      ADD R2,#4       ; incrementa R2 di una parola (4 byte)
      CMP R2,R3       ; R2==R3 ?
      BLT LOOP ; se R2 < R3 si devono sommare altri elementi
```

- L'idea fu di Von Neumann quando non esisteva l'indirizzamento indiretto a registro

Indirizzamento indicizzato

- L'esempio calcola l'AND bit a bit di due array A e B e pone in OR tutti i risultati parziali:

$(A[0] \text{ AND } B[0]) \text{ OR } (A[1] \text{ AND } B[1]) \text{ OR } (A[2] \text{ AND } B[2])$
OR ...

- R1 accumula l'OR degli AND.
- R2 indica la posizione corrente sugli array.

Indirizzamento indicizzato

- R3 contiene la costante 4096, per controllare la fine del loop.
- R4 è utilizzato per calcolare i singoli AND.

```

                                MOV R1,#0          ; accumula in R1 l'OR, inizialmente 0
                                MOV R2,#0          ; R2 = posizione corrente nei due array A e B
                                MOV R3,#4096 ; R3 = first index value not to use
LOOP:                          MOV R4,A(R2)       ; R4 = A[R2]
                                AND R4,B(R2)       ; R4 = A[R2] AND B[R2]
                                OR R1,R4           ; R1 = R1 OR R4
                                ADD R2,#4          ; R2 = R2 + 4
                                CMP R2,R3         ; R2==R3 ?
                                BLT LOOP          ; se R2 < R3 occorre considerare le altre celle
```

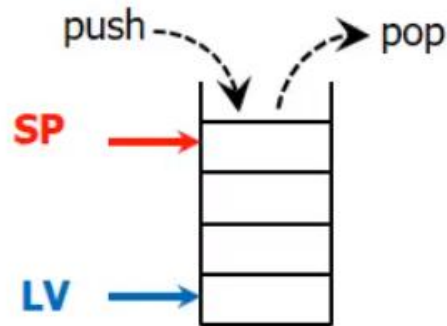
Indirizzamento indicizzato esteso

- L'indirizzo di memoria è calcolato sommando il contenuto di due registri:
 - Un registro memorizza la base.
 - Un registro memorizza l'indice.
- Ad esempio inizializzando R5 con A e R6 con B:

```
...  
LOOP:      MOV R4, (R2+R5)  
           AND R4, (R2+R6)  
...
```

Indirizzamento a stack

- Lo **Stack** è utilizzato per:
 - Gestire le chiamate di procedura.
 - Calcolare espressioni aritmetiche.
 - Salvare risultati intermedi.
- Lo Stack Pointer **SP** punta all'elemento affiorante della struttura.
- Le operazioni principali:
 - **push**: aggiunge un elemento alla cima dello stack.
 - **pop**: preleva un elemento dalla cima dello stack.
 - Operazioni aritmetiche sugli elementi affioranti (l'elemento al top è l'operando di destra mentre l'altro quello di sinistra).



Notazione Polacca Postfissa (o Inversa)

- La notazione polacca inversa (**RPN**) fu chiamata così in analogia alla notazione polacca (inventata da Łukasiewicz).
- È un modo per esprimere le espressioni matematiche in modo implicito senza l'utilizzo delle parentesi:

$(A+B) \times C$



$AB+C \times$

Forma infissa	Forma postfissa
$A + B \times C$	$A B C \times +$
$A \times B + C$	$A B \times C +$
$A \times B + C \times D$	$A B \times C D \times +$
$(A + B) / (C - D)$	$A B + C D - /$
$A \times B / C$	$A B \times C /$
$((A + B) \times C + D) / (E + F + G)$	$A B + C \times D + E F + G + /$

RPN ed esecuzione

- Supponiamo che la IJVM debba valutare l'espressione:

$$(8 + 2 \times 5) / (1 + 3 \times 2 - 4)$$

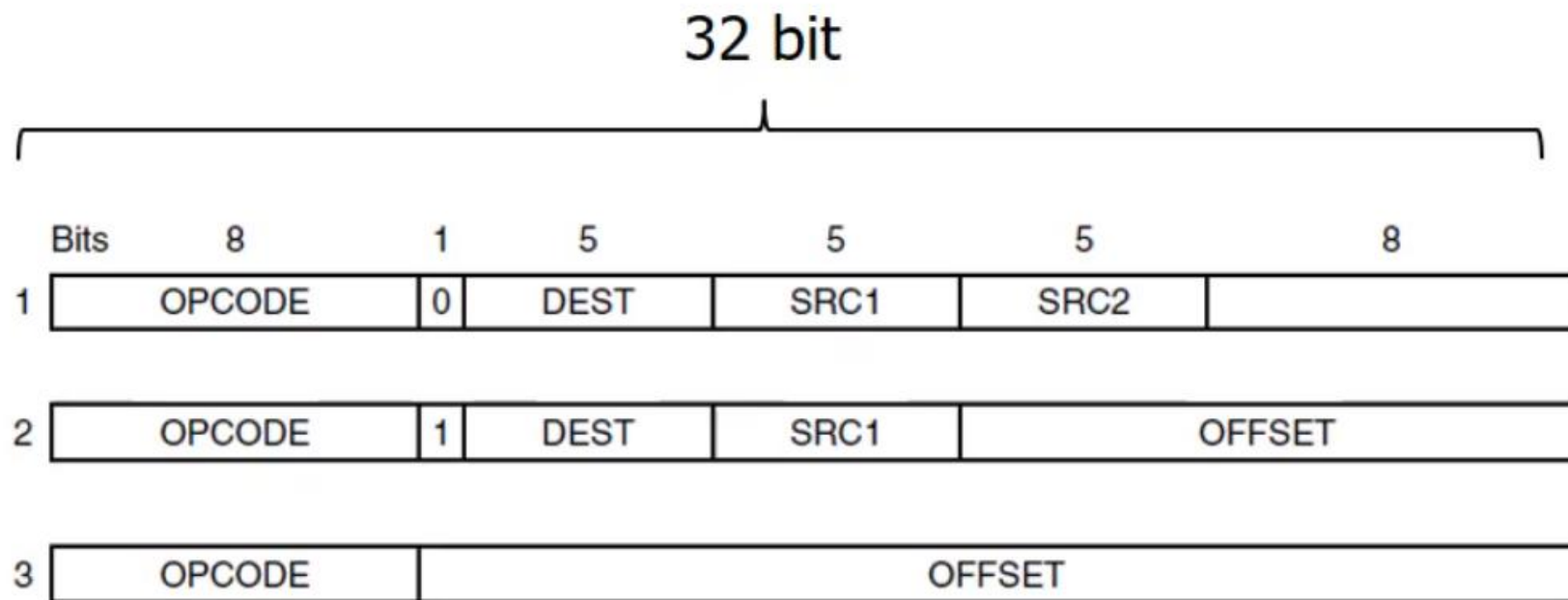
- che scritta in RPN risulta: $8\ 2\ 5\ \times\ +\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$

Step	Remaining string	Instruction	Stack
1	8 2 5 × + 1 3 2 × + 4 - /	BIPUSH 8	8
2	2 5 × + 1 3 2 × + 4 - /	BIPUSH 2	8, 2
3	5 × + 1 3 2 × + 4 - /	BIPUSH 5	8, 2, 5
4	× + 1 3 2 × + 4 - /	IMUL	8, 10
5	+ 1 3 2 × + 4 - /	IADD	18
6	1 3 2 × + 4 - /	BIPUSH 1	18, 1
7	3 2 × + 4 - /	BIPUSH 3	18, 1, 3
8	2 × + 4 - /	BIPUSH 2	18, 1, 3, 2
9	× + 4 - /	IMUL	18, 1, 6
10	+ 4 - /	IADD	18, 7
11	4 - /	BIPUSH 4	18, 7, 4
12	- /	ISUB	18, 3
13	/	IDIV	6

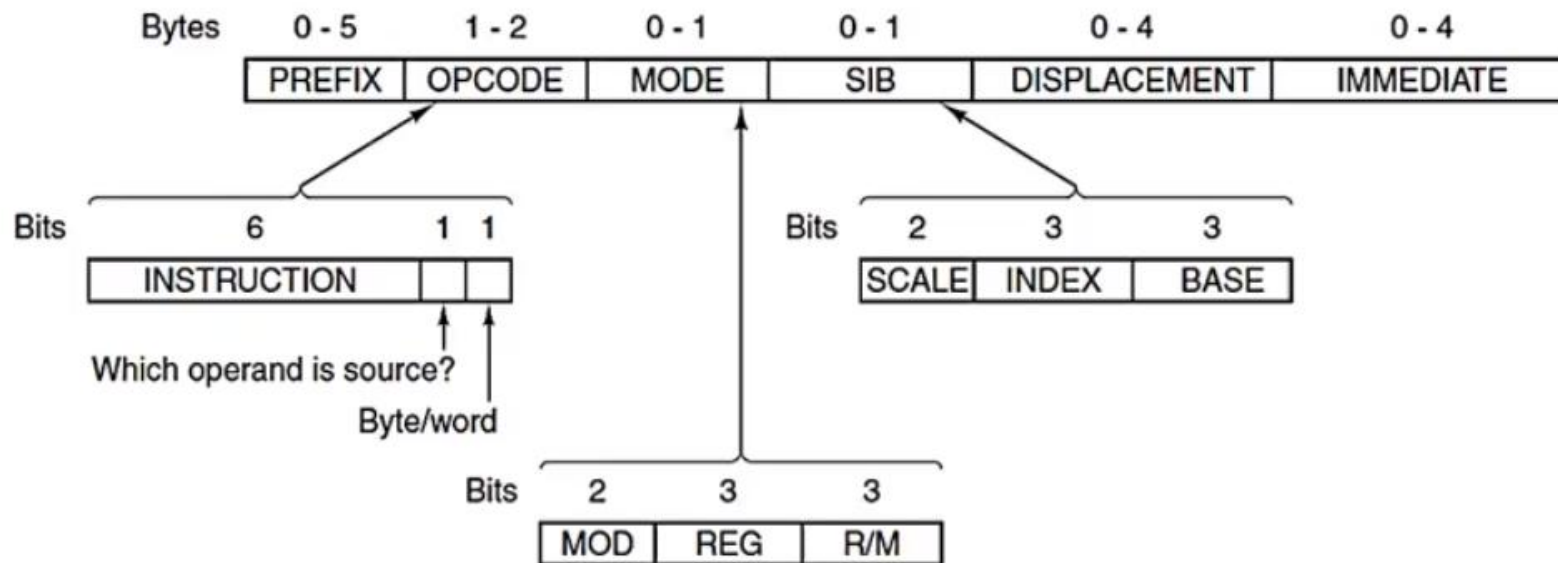
Ortogonalità

- Un set di istruzioni è caratterizzato da:
 - Codici operativi.
 - Modalità di indirizzamento.
- Quando tutte modalità di indirizzamento sono utilizzabili con tutti i codici operativi si dice che i codici e le modalità di indirizzamento sono tra loro **ortogonali**.
- Sebbene l'ortogonalità sia una caratteristica desiderabile perché semplifica la generazione del codice (es. SPARC) le architetture Intel non hanno questa caratteristica.

Instruction Set ortogonale



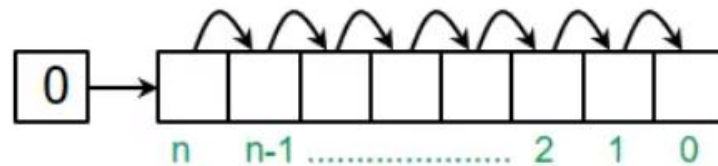
Instruction Set non ortogonale



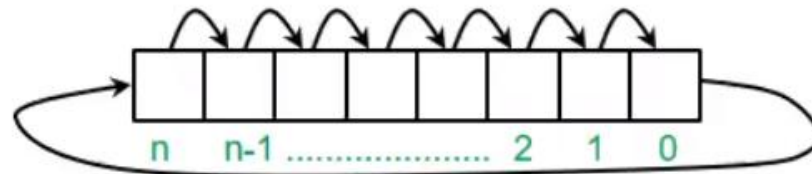
Tipi di istruzioni

- **Unarie**
 - Aritmetiche: complemento, opposto, incremento, radice,...
 - Bit a bit: not, shift e rotation.
- **Binarie**
 - Aritmetiche e logiche.
- **Trasferimento dati**
 - Da registro/memoria a registro/memoria (4 casi).
- **Selezione e confronto**
- **Iterazione**
- **Chiamata di una procedura**
- **Input/Output**
 - Polling, interrupt, DMA.

Istruzioni unarie sui bit



Shift a destra

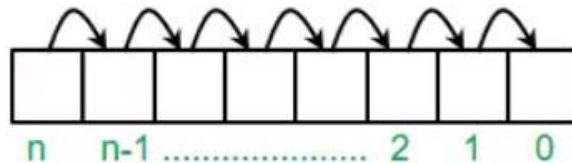


Rotazione a destra

00000000 00000000 00000000 01110011 A

00000000 00000000 00000000 00011100 Doppio **shift** a destra di A

11000000 00000000 00000000 00011100 Doppia **rotazione** a destra di A



Shift a destra con estensione del segno

11111111 11111111 11111111 11110000 A



00111111 11111111 11111111 11111100 **Shift** di A senza estensione del segno

11111111 11111111 11111111 11111100 **Shift** di A con estensione del segno

Applicazione dello shift

- Scorrere di k bit a sinistra/destra un **numero intero** X significa moltiplicarlo/dividerlo per 2^k (a meno di overflow/underflow).
- In base alla proprietà distributiva, si può estendere il ragionamento e calcolare rapidamente operazioni apparentemente complesse:

$$18 \cdot n = 2^4 \cdot n + 2^1 \cdot n$$

- In questo caso invece che eseguire una moltiplicazione (necessaria poiché non abbiamo una potenza di due) si può eseguire una somma e 5 scorrimenti:
 - $2^4 \cdot n$  corrisponde a scorrere n di 4 bit a sinistra
 - $2^1 \cdot n$  corrisponde a scorrere n di 1 bit a sinistra
 - Si calcola la somma dei due risultati

Attenzione alle istruzioni di shift

- In generale lo shift a destra può introdurre degli errori a causa del troncamento delle cifre che compongono il risultato
- Invece nel caso dei numeri negativi lo shift a sinistra non produce la moltiplicazione per 2 (contrariamente allo shift a destra che, troncamenti a parte, funziona correttamente)