Architettura degli elaboratori

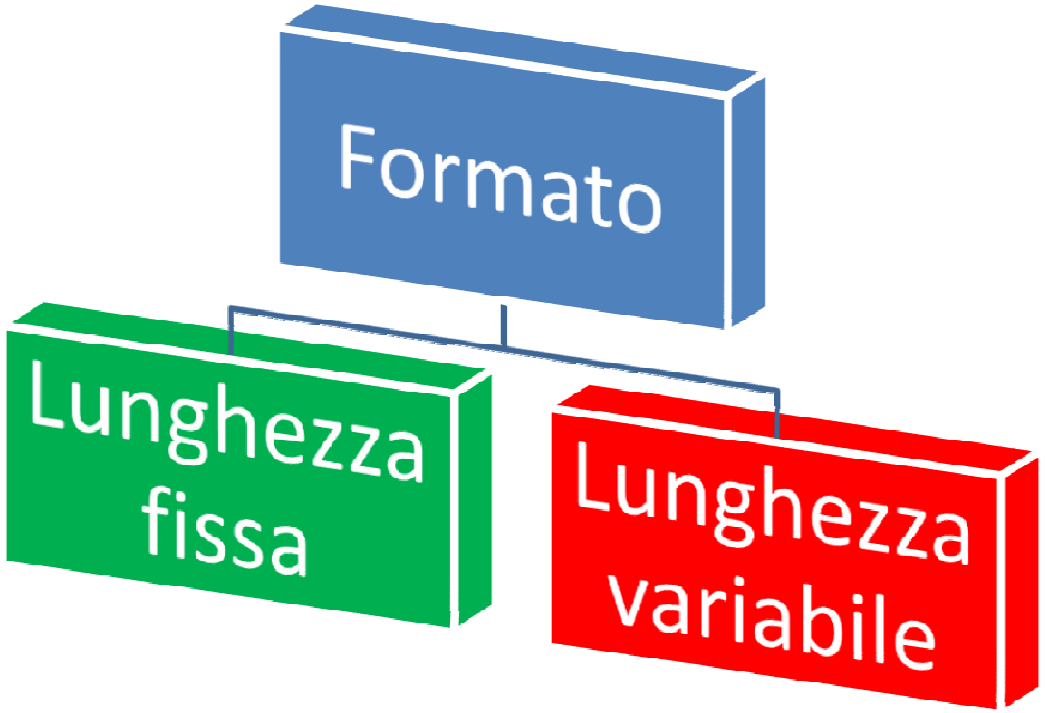
Istruzioni

Dott. Franco Liberati

ISTRUZIONI

Generalità

- Le istruzioni sono stringhe binarie che indicano al calcolatore elettronico operazioni da svolgere
- I bit di una istruzione sono suddivisi in sottostringhe denominate **campi**
- La suddivisione in campi individua il **formato dell'istruzione**



ISTRUZIONI

Generalità: campi

- I campi principali sono:
  - il **Codice Operativo** (o OPCODE), che specifica il tipo di operazione da eseguire (addizione, trasferimento dati, salto,…)
  - Il **Modo di Indirizzamento** (o ADDRESSING MODE), che indica il dato (operando o indirizzo) su cui devono essere effettuate le operazioni indicate dal codice operativo

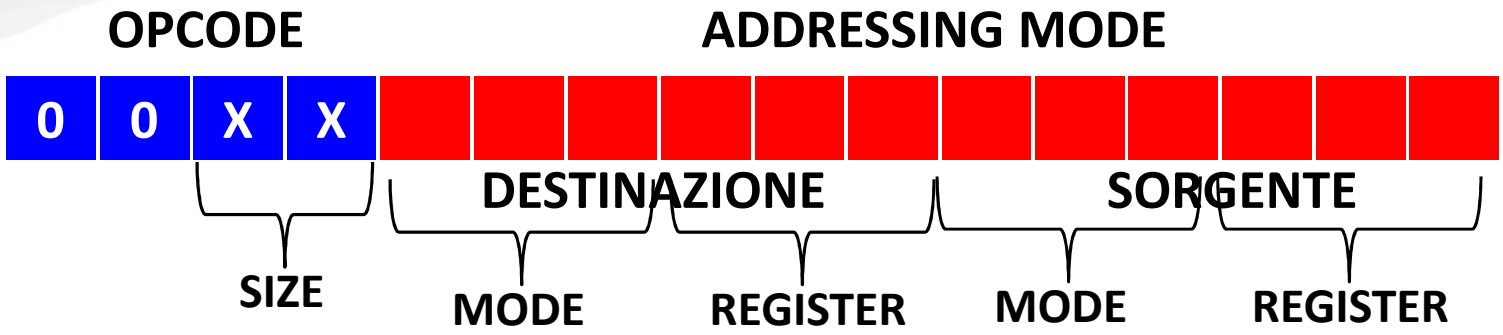
Codice operativo (OPCODE)	Modo di indirizzamento (ADDRESSING MODE)

```
LI          $t0,133
#Mette in $t0 il valore 133
ADD         $t0,$t1,$t2
#Somma gli operandi in $t1 e $t2 e pone il risultato in $t0
MOVE        $t0,$t1
#Sposta il contenuto di $t1 in $t0 (sovrascrive $t0)
LW          $t0,($a0)
#Sposta in $t0 il contenuto memorizzato nell'indirizzo
#riportato in $a0
LH          $t0,4-($a2)
#Sposta in $t0 i primi sedici bit del contenuto memorizzato
 #nell'indirizzo riportato in $a2 decrementato di 4
J           pippo
#Salta all'indirizzo rappresentato dall'etichetta pippo
```

ISTRUZIONI

Esempio: istruzione spostamento Motorola 68000

MOVE <ea> <ea> #Sposta un dato da una sorgente ad una destinazione



SIZE

01: operazione svolta interessando 8bit

11: operazione svolta interessando 16bit

10: operazione svolta interessando 32bit

MODE

000: registro Dn

010: locazione di memoria il cui indirizzo è nel registro indirizzi An

…

MOVE.L D1,D0

0010 000 001 000 000

MOVE.W D3,(A2)

0011 000 011 010 010

ISTRUZIONI

Generalità: indirizzo effettivo

- Il modo di indirizzamento può fare riferimento ad un operando contenuto nella stessa istruzione (come avviene nell'indirizzamento immediato) o, come spesso accade, si ha un **indirizzo effettivo**
- Un indirizzo effettivo è una locazione di memoria oppure è una etichetta che specifica un registro
- La locazione di memoria può fare riferimento ad una parte del programma (ad esempio con le istruzioni di salto) oppure ad un'area in cui è presente un dato (impiegato da una istruzione di trasferimento, che sposta un operando dalla memoria ad un registro)

Indirizzo	Etichetta	Istruzione
000		lb $t0,n
004		lb $t1,k
008		li $t2,0
012		li $t3,1
016		li $t4,1
020	ciclo:	bgt $t4,$t0,fine_ciclo
024		mul $t3,$t3,$t1
028		add $t2,$t2,$t3
032		addi $t4,$t4, 1
036		j ciclo
040	fine_ciclo:	
044		sw $t2,somma
300		n: byte 23
304		k:byte 12
308		somma: .word 0

Indirizzo effettivo (locazione memoria dati)

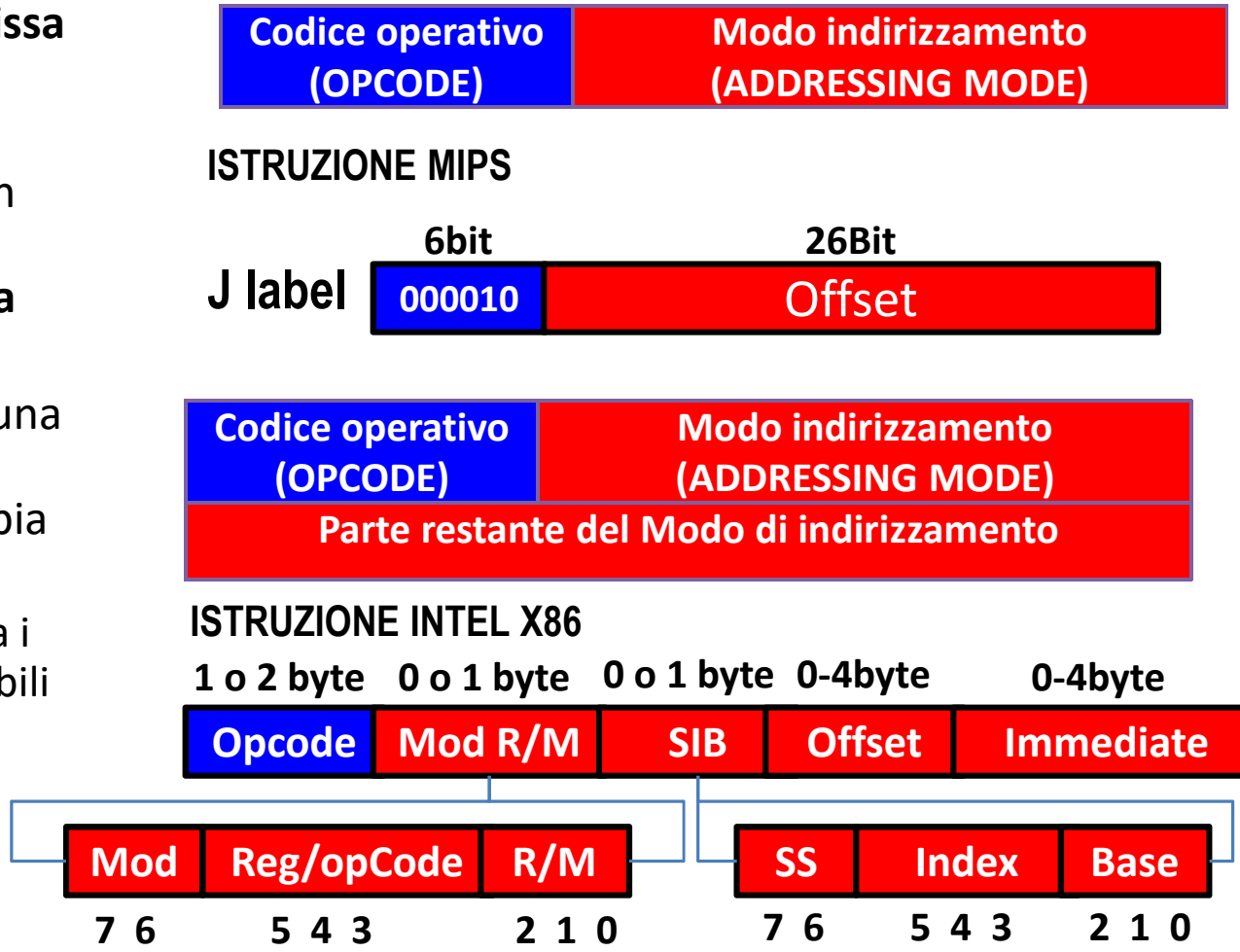
Indirizzo effettivo (locazione memoria istruzioni)

Indirizzo effettivo (registro nella Control Unit)

ISTRUZIONI

Generalità: lunghezza dell'istruzione

- Le istruzioni possono avere **lunghezza fissa** o **lunghezza variabile**
- Il **formato a lunghezza fissa** prevede un insieme di istruzioni (*instruction set*) con una dimensione predefinita (una sottoclasse di questa sono le **istruzioni a referenziamento implicito**)
- In alternativa, una istruzioni può avere una **lunghezza variabile**. In base al tipo di istruzione e agli operandi coinvolti cambia la dimensione
  - Un istruzione a lunghezza variabile ha i **bit in eccesso** - cioè non rappresentabili nella parola - ospitati nella parola successiva (richiede più accessi in memoria)



ISTRUZIONI

Generalità: lunghezza dell'istruzione (fissa – MIPS)

- Il MIPS ha un **formato a lunghezza fissa a 32 bit** (anche detto **ISA ortogonale**) in cui l'OPCODE è costituito da 6bit e l'ADDRESSING MODE da 24bit

In questa architettura, qualora si faccia riferimento a un indirizzo o ad un operando che richieda più di 16bit, l'istruzione (che in realtà è una pseudo istruzione) è suddivisa in due istruzioni elementari che consentono il riempimento dell'operando/indirizzo in un registro

li $t0,3000000000

10110010 11010000 01011110 00000000

B2 50 5E 00

Diventa

lui $at, B250

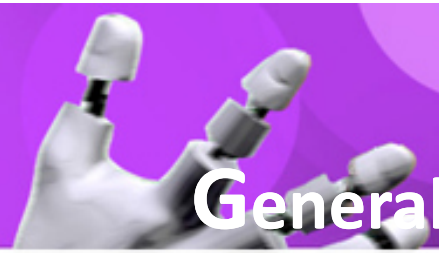
In $at

10110010 11010000 00000000 00000000

ori $t0,$at,5E00

In $t0

10110010 11010000 01011110 00000000



ISTRUZIONI

Generalità: lunghezza dell'istruzione (variabile– x86)

I **Processori intel x86** hanno un formato a **lunghezza variabile da 8bit a 64bit** in cui l'OPCODE è costitutito da 8bit o 16bit e l'ADDRESSING MODE varia da 0bit a 48bit

I primi 6-14bit dell'OPCODE discriminano il tipo di istruzione. L'ultimo bit, **s**, dell'OPCODE indica la grandezza degli operandi (se si tratta di un registro, o di un indirizzo a 16 o 32bit) mentre il penultimo bit, **d**, specifica se il risultato va messo in un registro o in una locazione di memoria

OPCODE ADD

000000ds

Somma con scrittura del risultato in una locazione di memoria il cu indirizzo **al** è a 32bit

0000000s

add [ebx], al

Somma con scrittura del risultato in un registro ad uso generale

0000001s

add al, [ebx]

Somma tra registri con scrittura del risultato in una locazione di memoria il cui indiizzo è a 32bit

00000000

add [ebx], al *cioè* al← ebx+ ebx

Somma tra un registro e un operando a 32bit con scrittura del risultato in un registro ad uso generale

00000011

add <const32>, [ebx] *cioè* ebx← ebx+ const32

ISTRUZIONI

Linguaggio macchina

- Le istruzioni sono eseguite quando sono scritte in **linguaggio macchina** (nei primi elaboratori esisteva solo questo tipo di linguaggio)

Linguaggio assemblativo:	J ciclo
Assemblaggio:	J 68786
Linguaggio Macchina (MIPS):	000010 00000000010000110010100000

Architettura degli elaboratori

Linguaggio assemblativo

LINGUAGGIO ASSEMBLATIVO

Sintassi

- Il programmatore ricorre ad una rappresentazione simbolica delle istruzioni, utilizzando codici mnemonici che possono essere interpretati in maniera più comoda rispetto alle sequenze binarie: **istruzioni assembly**
- La **sintassi** di una istruzione assembly è costituita da:
  - Un **indirizzo**, dove risiede l'istruzione in memoria (spesso omesso, perché impostato dall'assemblatore)
  - Una **etichetta** (opzionale): utile per individuare l'indirizzo dell'istruzione a cui bisogna saltare
  - Una **direttiva** (opzionale): informazioni utili all'assemblatore (riservare locazioni di memoria dove stipare i dati, inizio del programma, definizione di MACRO,…)
  - Una **istruzione**:
    - un **codice mnemonico**, che descrive l'istruzione con pochi, ma significativi, caratteri
    - Il **modo di indirizzamento**, cioè i dati su cui operare o il luogo dove essi risiedono
  - i **commenti**,indispensabili per la comprensione del codice

etichetta: direttiva/istruzione[opcode,addressing mode] # commento

64	CICLO:	ADD	$t0,$t1$t2	#Esegue $t0=t1 + t2

LINGUAGGIO ASSEMBLATIVO

Set delle istruzioni

❑ L'insieme delle istruzioni assembly (*instruction set assembly* ISA) definiscono un **linguaggio assemblativo** (*assembly* o *assembly language*)

ISA MIPS (alcune istruzioni)

Add	add $d,$s,$t	$d = $s + $t
Add unsigned	addu $d,$s,$t	$d = $s + $t
Subtract	sub $d,$s,$t	$d = $s - $t
Subtract unsigned	subu $d,$s,$t	$d = $s - $t
Add immediate	addi $t,$s,C	$t = $s + C (signed)
Add immediate unsigned	addiu $t,$s,C	$t = $s + C (unsigned)
Multiply	mult $x,$y	LO = (($x * $y) << 32) >> 32; HI = ($x * $y) >> 32;
Divide	div $x, $y	LO = $x / $y HI = $x % $y
Divide unsigned	divu $x, $y	LO = $x / $y HI = $x % $y

Trasferimento dati	Load double word	ld $x,C($y)
	Load word	lw $x,C($y)
	Load halfword	lh $x,C($y)
	Load halfword unsigned	lhu $x,CONST($y)
	Load byte	lb $x,C($y)
	Load byte unsigned	lbu $x,C($y)
	Store double word	sd $x,C($y)
	Store word	sw $x,C($y)
	Store halfword	sh $x,C($y)
	Store byte	sb $x,C($y)
	Load upper immediate	lui $x,C
	Move from high	mfhi $x
	Move from low	mflo $x
	Move from Coprocessor Z	mfcZ $x, $y
	Move to Coprocessor Z	mtcZ $x, $y

Logiche	And	and $d,$s,$t
	And immediate	andi $t,$s,C
	Or	or $x,$y,$z
	Or immediate	ori $x,$y,C
	Exclusive or	xor $x,$y,$3
	Nor	nor $x,$y,$z
	Set on less than	slt $x,$y,$z
	Set on less than immediate	slti $x,$y,C
Bitwise Shift	Shift left logical	sll $x,$y,C
	Shift right logical	srl $x,$y,C
	Shift right arithmetic	sra $x,$y,C
Salti condizionati	Branch on equal	beq $s,$t,C
	Branch on not equal	bne $x,$y,C
Salti incondizionati	Jump	j C
	Jump register	jr $x
	Jump and link	jal C

LINGUAGGUO ASSEMBLATIVO

Pseudoistruzioni

- Il legame che intercorre tra una istruzione macchina e una istruzione assembly è di **uno a uno**, nel senso che ad ogni istruzione macchina corrisponde una ed una sola istruzione assembly
- Per comodità molti linguaggi assembly utilizzano delle **pseudoistruzioni** ovvero delle istruzioni che sono composte da una o più istruzione assembly elementare

ESEMPIO DI PSEUDO ISTRUZIONE IN MIPS

```
    LW $t0,x
    LW $t1,y
    BGT $t0,$t1, SALTO
    #Se è vero che $t0>$t1 vai all'etichetta SALTO
    SW $t0,z
SALTO:
    …

    SLT   $1,$9,$8
    #set del registro $1 ($at) ad 1 se $t0>$t1
    BNE $1,$0,0x001002
    #se AT!=0 salta alla locazione di memoria 0x001002
```

LINGUAGGIO ASSEMBLATIVO

Definizione ed uso di Macro

- Un linguaggio assembly consente di definire delle **macro**: una macro sostituisce una serie di istruzioni
- La macro va prima definita (le si associa un identificatore) e poi va richiamata nel programma

ESEMPIO DI MACRO IN MIPS

```
.macro end
    li $v0,10
    syscall
.end_macro

.text
.globl main

main:
        …
end
```

In fase di pre-assemblamento si sostituisce end con le istruzioni predefinite

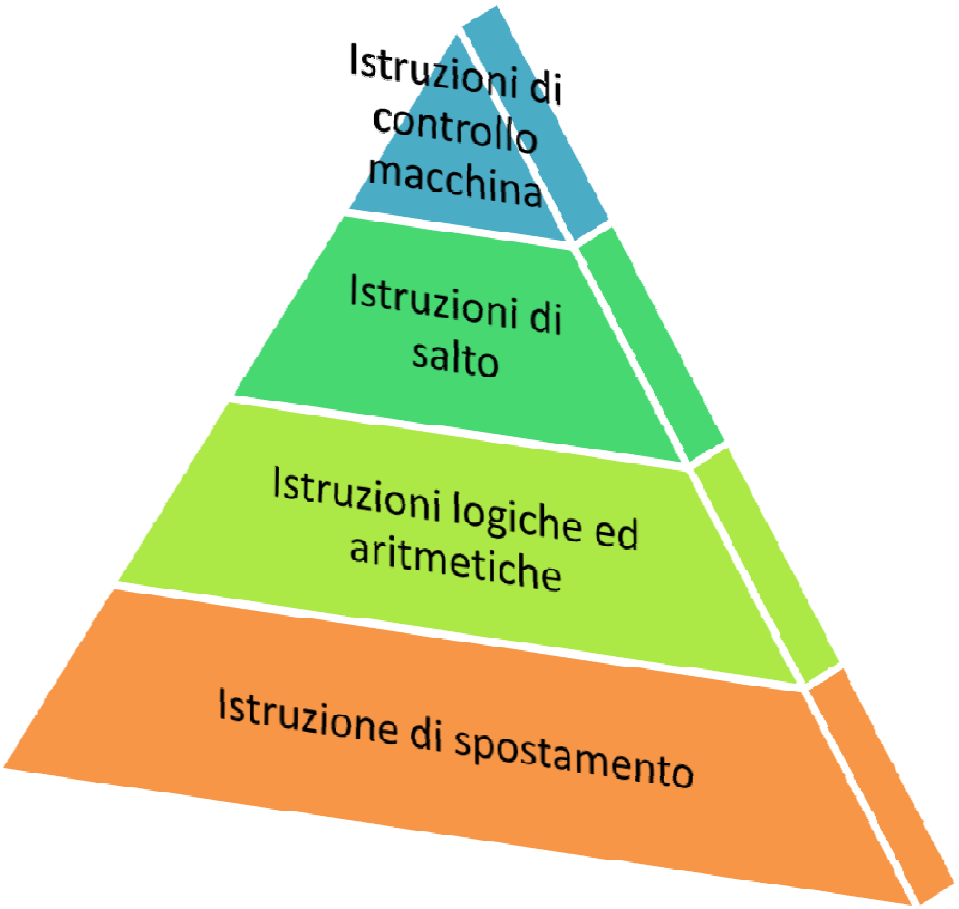
```
.text
.globl main

main:
        …
    li $v0,10
    syscall
```

Architettura degli elaboratori

Classi di istruzioni

ARGOMENTI DELLA LEZIONE

- Classi di Istruzione
  - Istruzione di spostamento dati
  - Istruzioni logiche ed aritmetiche
  - Istruzioni di salto:
    - condizionato
    - non condizionato
    - a funzione (o a subroutine)
    - trap
  - Istruzione di controllo della macchina



Architettura degli elaboratori

Istruzione di spostamento

ISTRUZIONI DI SPOSTAMENTO

- Le istruzioni per lo spostamento dei dati servono a trasferire (ovvero copiare) un dato da una sorgente ad una destinazione e cioè da:
  - memoria a registro
  - registro a memoria
  - registro a registro
  - memoria a memoria

Codice Mnemonico	Sorgente	Destinazione

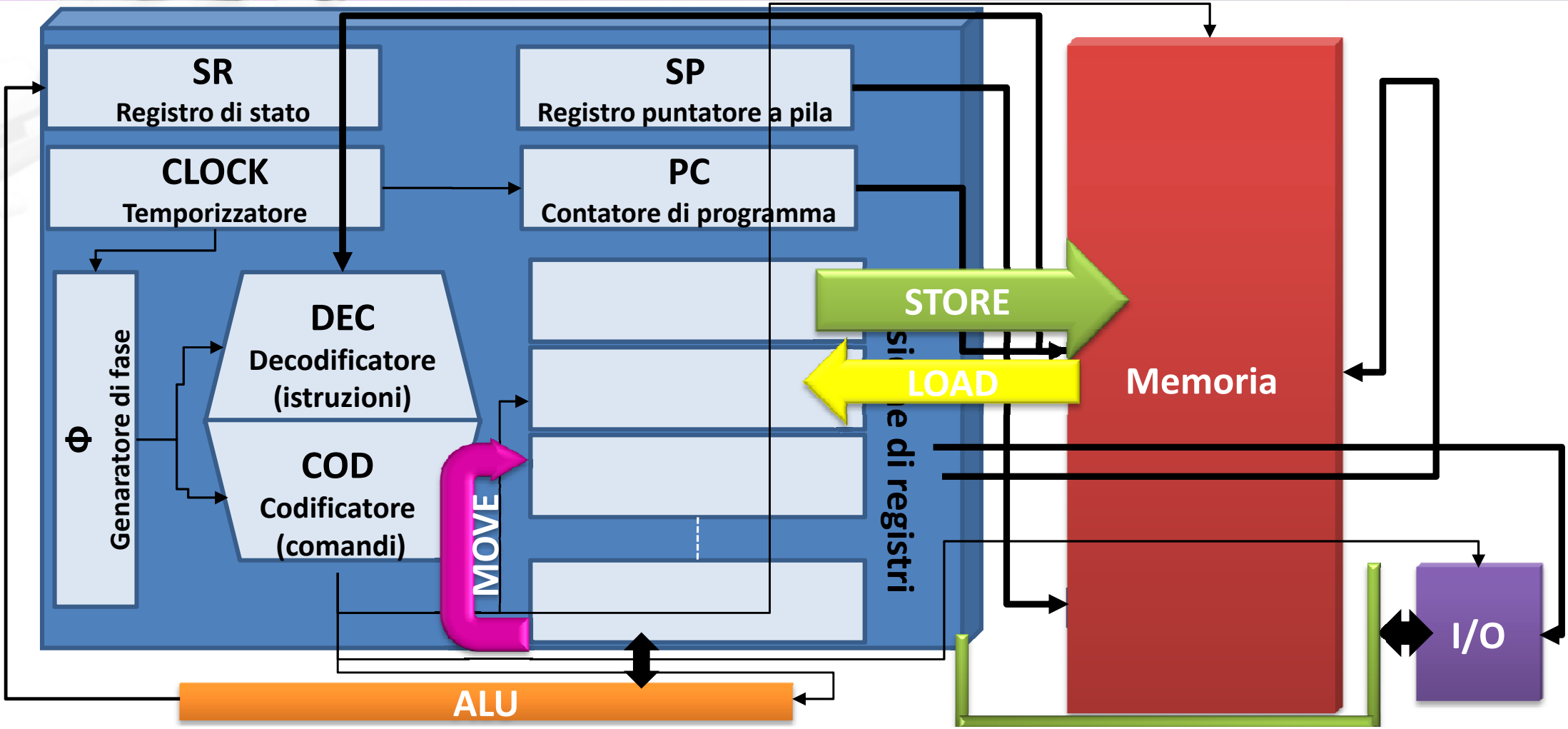
ISTRUZIONI DI SPOSTAMENTO

- Le istruzioni di spostamento possono interessare la CPU e la Memoria (LOAD, STORE, PUSH e POP) o solamente i registri nella CPU (MOVE)
- Il contenuto della destinazione è sovrascritto da quello della sorgente

CODICE	OPERANDI	Commento
LOAD	<Sorgente>,<Destinazione>	Legge l'operando dalla sorgente (una locazione di memoria) e lo copia nella destinazione (tipicamente un registro)
STORE	<Sorgente>,<Destinazione>	Legge l'operando dalla sorgente (tipicamente un registro) e lo copia nella destinazione (una locazione di memoria esplicitata)
MOVE	<Sorgente>,<Destinazione>	Sposta il contenuto di un registro Sorgente ad un registro Destinazione
PUSH	<Sorgente>	Sposta un operando da una Sorgente (un registro o una locazione in memoria) in cima allo stack/pila Equivale a STORE sorg,-($SP)
POP	<Destinazione>	Sposta un operando dalla cima dello stack/pila in una Destinazione (un registro o una locazione in memoria) Equivale a LOAD ($SP)+,dest

ISTRUZIONI SPOSTAMENTO

Load/Store/Move



ISTRUZIONI DI SPOSTAMENTO

Esempio: scambio di informazioni da due locazioni di memoria

	operando1	operando2	R0	R1	R2
.DATA					
operando1: WORD 56	56				
operando2: WORD 100	56	100			
.TEXT					
LOAD.W R0, operando1	56	100	56		
LOAD.W R1, operando2	56	100	56	100	
MOVE R2,R0	56	100	56	100	56
MOVE R0,R1	56	100	100	100	56
MOVE R1,R2	56	100	100	56	56
STORE.W R0, operando1	100	100	100	56	56
STORE.W R1, operando2	100	56	100	56	56
.END					

Architettura degli elaboratori

Istruzione logiche-aritmetiche

ISTRUZIONI LOGICHE-ARITMETICHE

Generalità

- Le **istruzioni aritmetiche** consentono di effettuare le operazioni su numeri interi binari rappresentati in complemento a due

- In alcuni casi le ALU possono svolgere operazioni anche con numeri in virgola mobile, spesso queste operazioni sono demandate ad una unità di calcolo – il coprocessore matematico - che è visto come un dispositivo di I/O

ISTRUZIONI LOGICHE-ARITMETICHE

Istruzioni aritmetiche

- Le **istruzioni aritmetiche** di base offerte dalla ALU sono il complemento, la comparazione e l'addizione; le funzioni come la moltiplicazione o divisione e la sottrazione possono essere ricavate sfruttando algoritmi che impiegano le operazioni elementari sopra citate
- Le istruzioni aritmetiche sono eseguite dall'ALU la quale produce due linee di uscita:
  - il risultato dell'operazione;
  - il vettore di bit *flags* (anche CC o *condition code*) che è implicitamente caricato nello Status Register

CODICE	OPERANDI	Commento
ADD	<Destinazione><Sorgente><Sorgente>	Legge gli operandi dalla sorgente (memoria/registri), effettua la somma ed il risultato è trasferito nella destinazione (tipicamente un registro).
CMP	<Destinazione><Sorgente><Sorgente>	Legge gli operandi dalla sorgente (memoria/registri), effettua la comparazione ed il risultato è trasferito nella destinazione (tipicamente un registro)
NEG	<Destinazione><Sorgente>	Legge l'operando dalla sorgente (memoria/registro), effettua la negazione ed il risultato è trasferito nella destinazione (tipicamente un registro)
MUL	Registro, <Sorgente>, <Sorgente>	Legge gli operandi (moltiplicando e moltiplicatore) dalla sorgente (memoria/registri) ed effettua la moltiplicazione riportando il risultato in un registro
DIV	Registro, <Sorgente>, <Sorgente>	Legge gli operandi (dividendo e divisore) dalla sorgente (memoria/registri) e restituisce il quoziente della divisione tra interi in un registro
REM	Registro, <Sorgente>, <Sorgente>	Legge gli operandi (dividendo e divisore) dalla sorgente (memoria/registri) e restituisce il resto della divisione tra interi in un registro

ISTRUZIONI LOGICHE-ARITMETICHE

Esempio: cubo di un numero

	operando	cubo	R0	R1	R2
.DATA					
operando: WORD 8	8				
cubo: WORD 0	8	0			
.TEXT					
LOAD.W R0, operando	8	0	8		
MUL R1,R0, R0	8	0	8	64	
MUL R2,R1,R0	8	0	8	64	512
STORE.W R2, cubo	8	512	8	64	512
.END					

ISTRUZIONI LOGICHE-ARITMETICHE

Istruzioni logiche

- Le **operazioni logiche** permettono l'esecuzione delle più importanti operazioni definite nell'algebra booleana su stringhe binarie. Come per le operazioni aritmetiche, anche in questo caso, le operazioni avvengono per tutti i bit nelle corrispondenti posizioni
- La sintassi è simile alle istruzioni aritmetiche e l'operando sorgente può essere in una locazione di memoria, in un registro, o un operando (residente nell'istruzione); mentre la destinazione è di solito un registro.
- Le istruzioni logiche permettono di modificare alcuni bit di un registro, di esaminare il loro valore o di settarli a 0 o 1 (sono usati per realizzare **maschere**)

CODICE	OPERANDI	Commento
AND	Registro, <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'AND riportando il risultato in un registro
OR	Registro, <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'OR riportando il risultato in un registro
XOR	Registro, <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'XOR riportando il risultato in un registro
NOT	Registro, <Sorgente>	Legge l'operando dalla sorgente (memoria/registri) ed effettua l'NOT riportando il risultato in un registro

ISTRUZIONI LOGICHE-ARITMETICHE

Esempio: maschera primi 24 bit

	operando	maschera	LS24bit	R0	R1	R2
.DATA						
operando: WORD 3271696668	11000011 00000010 00100001 00011100					
maschera: WORD 16777215	11000011 00000010 00100001 00011100	00000000 11111111 11111111 11111111				
.TEXT						
LOAD.W R0, operando	3271696668			3271696668		
LOAD.W R1,maschera	3271696668	16777215		3271696668	16777215	
AND R2,R1, R0	3271696668	16777215		11000011 00000010 00100001 00011100	0000000 11111111 11111111 11111111	00000000 00000010 00100001 00011100
STORE.W R2, LSB24	3271696668	16777215	00000000 00000010 00100001 00011100	11000011 00000010 00100001 00011100	0000000 11111111 11111111 11111111	00000000 00000010 00100001 00011100
.END						

ISTRUZIONI LOGICHE-ARITMETICHE

Istruzioni logico-aritmetiche

- Le istruzioni di **rotazione** (rotate) e **slittamento** (shift) operano su un solo dato posto in un registro. Queste istruzioni cambiano l'ordine dei bit nel registro ed hanno un significato:
  - **logico**: per effettuare lo scorrimento dei bit del registro nella direzione e nel numero di posizioni specificati. Il bit C (carry o trabocco) dello Status Register riceve l'ultimo bit che fuoriesce dal registro;
  - **aritmetico**: è opportuno ricordare che uno shift a destra equivale a dividere l'operando per 2^k (con k il numero di posizioni scorse), mentre uno scorrimento verso sinistra equivale a moltiplicare l'operando per 2^k (con k il numero di posizioni scorse)

CODICE	OPERANDI	Commento
SL	Registro, k	Slittamento a sinistra di k posti del registro
SR	Registro, k	Slittamento a destra di k posti del registro
ROL	Registro, k	Rotazione a sinistra di k posti del registro
ROR	Registro, k	Rotazione a destra di k posti del registro

ISTRUZIONI LOGICHE-ARITMETICHE

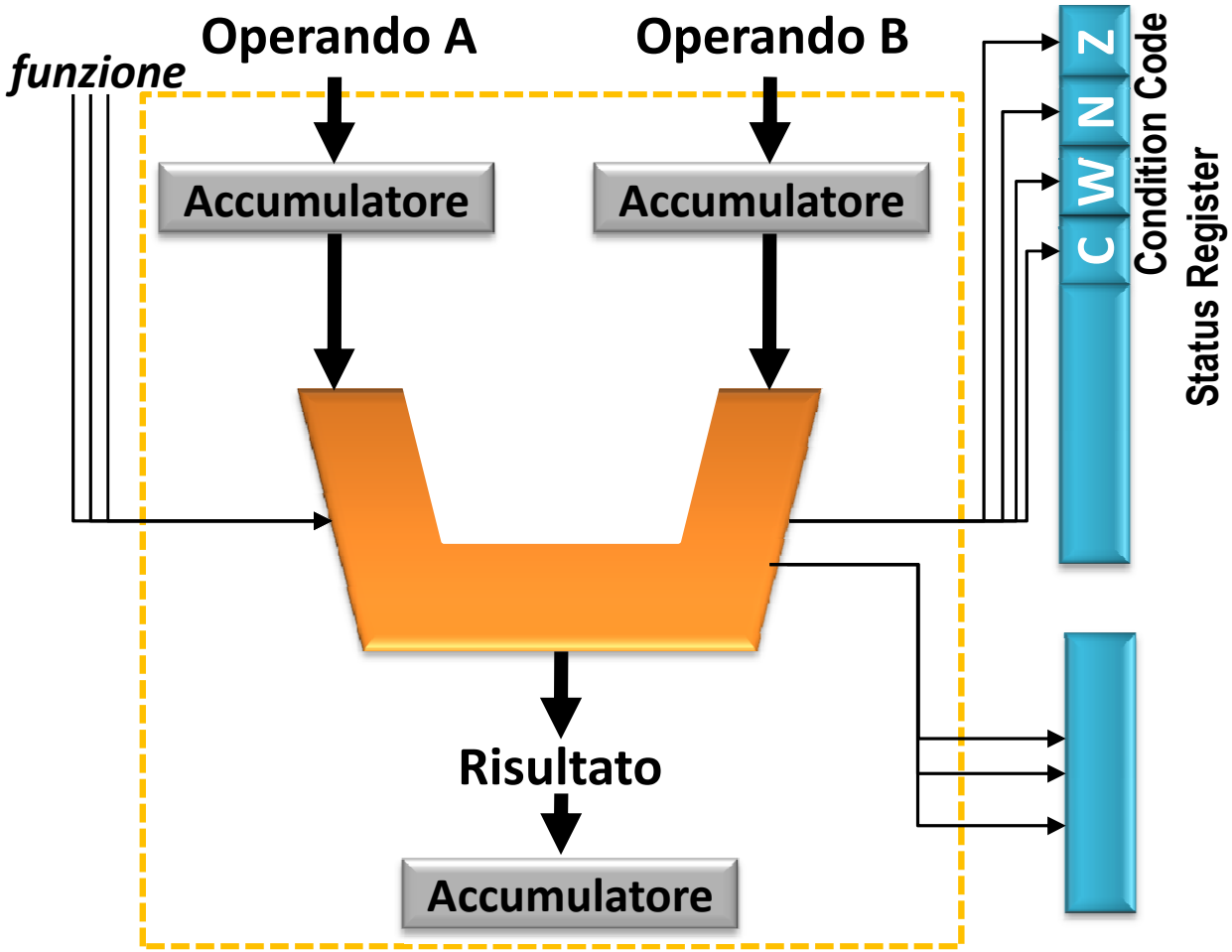
Esempio: analisi positività di un numero

	Operando	LS24bit	R0	R1	R2
.DATA					
operando: WORD 3271696668	11000011 00000010 00100001 00011100				
LBS24: WORD 0	3271696668	0			
.TEXT					
LOAD.W R0, operando	3271696668	0	11000011 00000010 00100001 00011100		
SR R1,R0,31	3271696668	0	11000011 00000010 00100001 00011100	00000000 00000000 00000000 00000001	
STORE.W R1, LSB24	3271696668	1			
.END					

ISTRUZIONI

Istruzioni logiche aritmetiche: condition Code

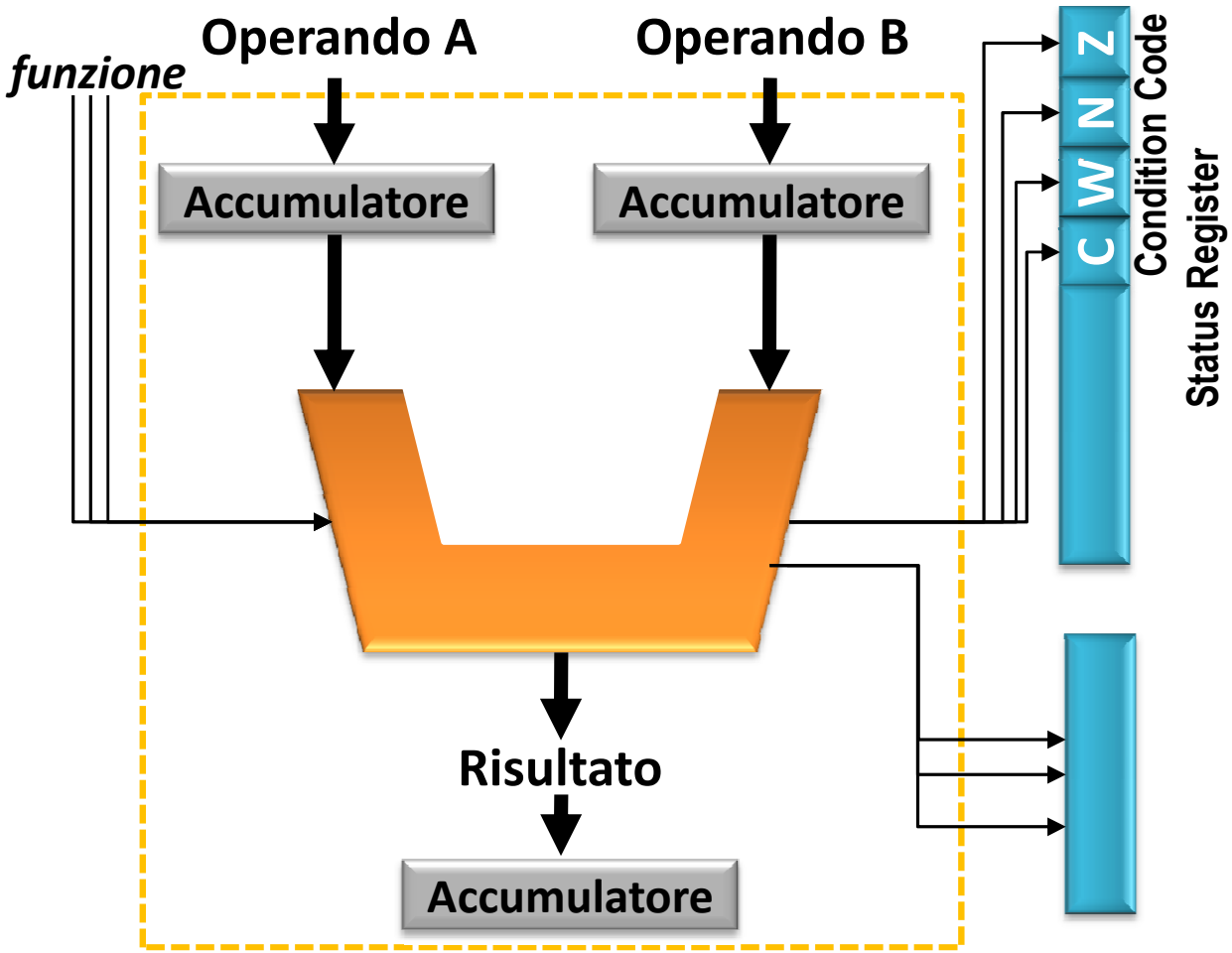
- Ogni **istruzione logico-aritmetica**, produce dei bit, definiti **flags**, che sono implicitamente memorizzati nel **registro di stato** (PSW, *processor status word*, o STATUS *register*) con il nome codici di condizione, o *condition code*
- I Condition Codes svolgono un **ruolo fondamentale per le istruzioni di salto condizionato**



ISTRUZIONI

Codici di condizione

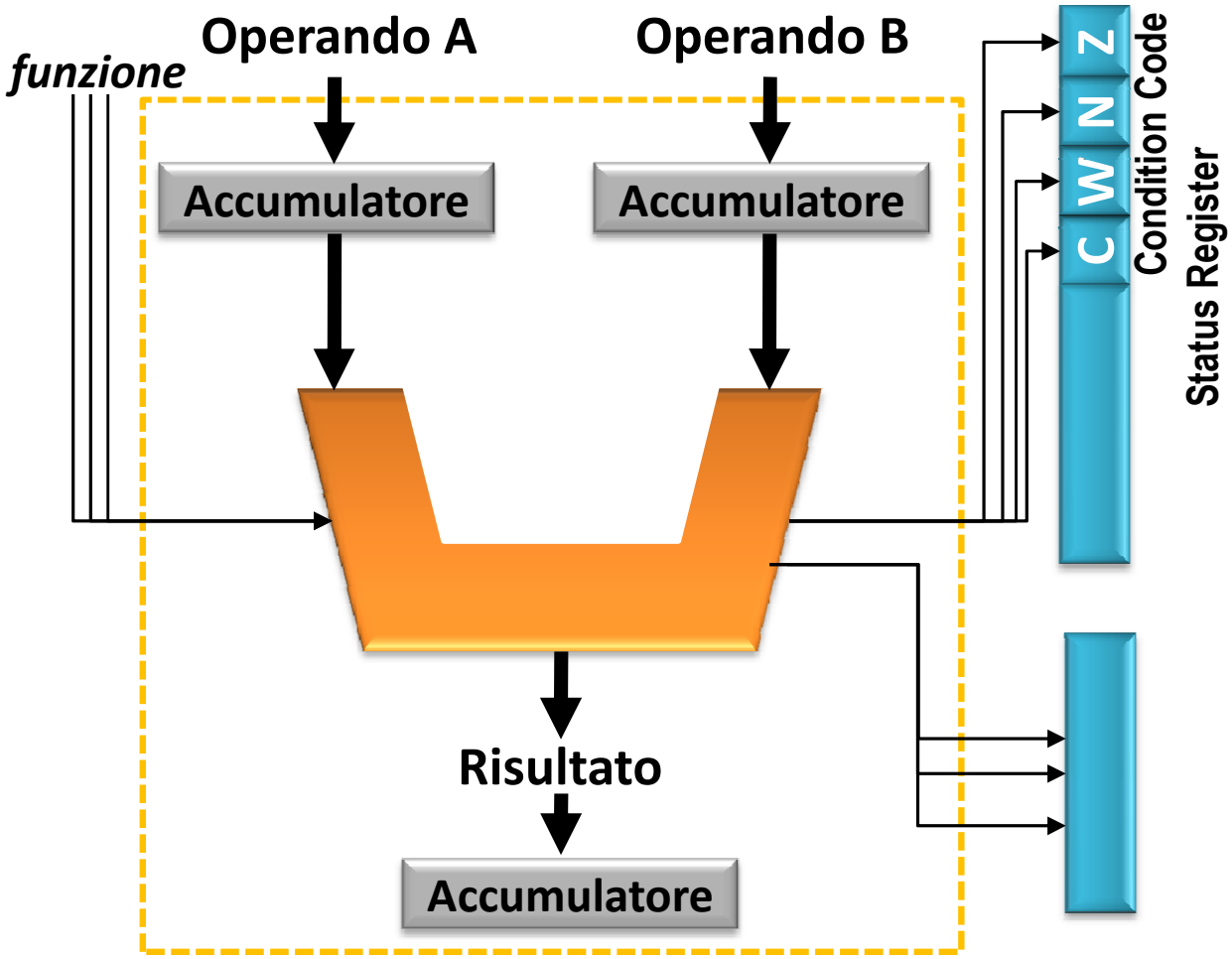
- I principali flags sono:
  - **C - Carry**: Individua il trabocco ed è impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un riporto (addizione) o un prestito (sottrazione) a sinistra del bit più significato del risultato, 0 altrimenti
  - **N - Negative**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un risultato negativo, 0 altrimenti. Ovvero Negative è una copia del bit più significativo del risultato
  - **Z - Zero**: impostato ad 1 se l'ultima operazione effettuata dall'ALU è nulla, 0 altrimenti.



ISTRUZIONI

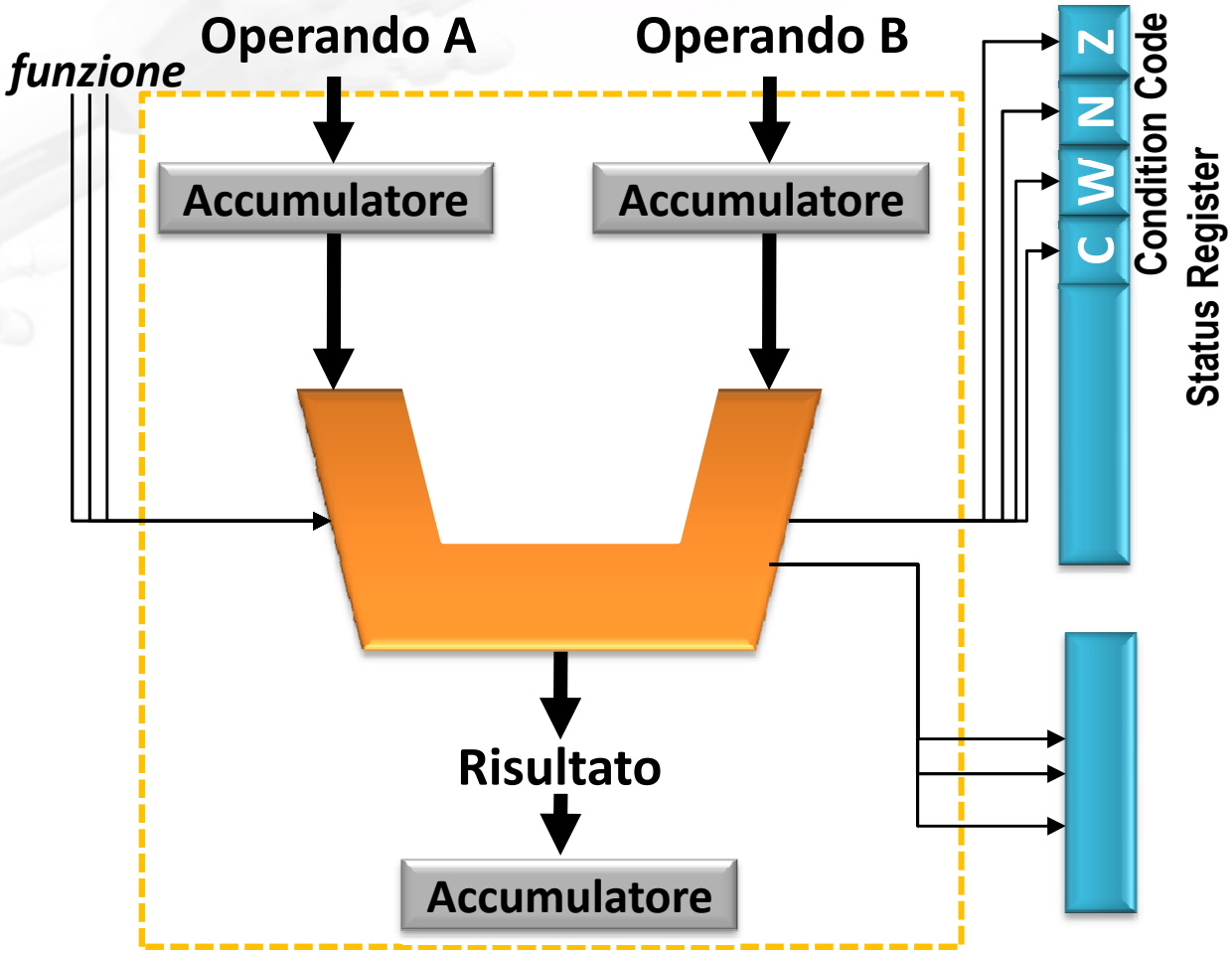
Codici di condizione

- I principali flags sono:
  - **W - Overflow**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha superato la capacità di rappresentazione data dalla lunghezza della parola, 0 altrimenti



ISTRUZIONI

Codici di condizione




```
li $t1, -1
li $t2, 1
add $t0, $t1, $t2
```

Z = 1
(perchè il risultato è 0)
C = 1
(perchè si verifica un trabocco)
N = 0
(perchè il risultato è 0, che è considerato positivo)
W = 0
(perchè non c'è overflow)

CONDITION CODE

Modifiche dei CC in relazione alle istruzione

Codice	C	N	Z	W
ADD	1/0	1/0	1/0	1/0
CMP	1/0	1/0	1/0	1/0
NEG	1/0	1/0	1/0	1/0
SUB	1/0	1/0	1/0	1/0
AND	0	1/0	1/0	0
OR	0	1/0	1/0	0
XOR	0	1/0	1/0	0
NOT	0	1/0	1/0	0
SL	1/0	1/0	1/0	1/0
SR	1/0	1/0	1/0	1/0
ROL	1/0	0	0	0
ROR	1/0	0	0	0

Architettura degli elaboratori

Istruzioni Implicite per il settaggio dei bit

ISTRUZIONI IMPLICITE

Settaggio dei bit

- Esistono istruzioni, con **modo di indirizzamento implicito** (cioè non bisogna specificare l'indirizzo effettivo perché già noto all'Unità di controllo), che consento di operare sui singoli bit del Registro di Stato

CODICE	Commeto	CODICE	Commeto
CLRC	Imposta a 0 il flag C	**SETC**	Imposta a 1 il flag C
CLRN	Imposta a 0 il flag N	**SETN**	Imposta a 1 il flag N
CLRZ	Imposta a 0 il flag Z	**SETZ**	Imposta a 1 il flag Z
CLRW	Imposta a 0 il flag W	**SETW**	Imposta a 1 il flag W

CONDITION CODE

Modifiche dei CC in relazione alle istruzione

Codice	C	N	Z	W
CLRC	0	-	-	-
CLRN	-	0	-	-
CLRZ	-	-	0	-
CLRW	-	-	-	0
SETC	1	-	-	-
SETN	-	1	-	-
SETZ	-	-	1	-
SETW	-	-	-	1

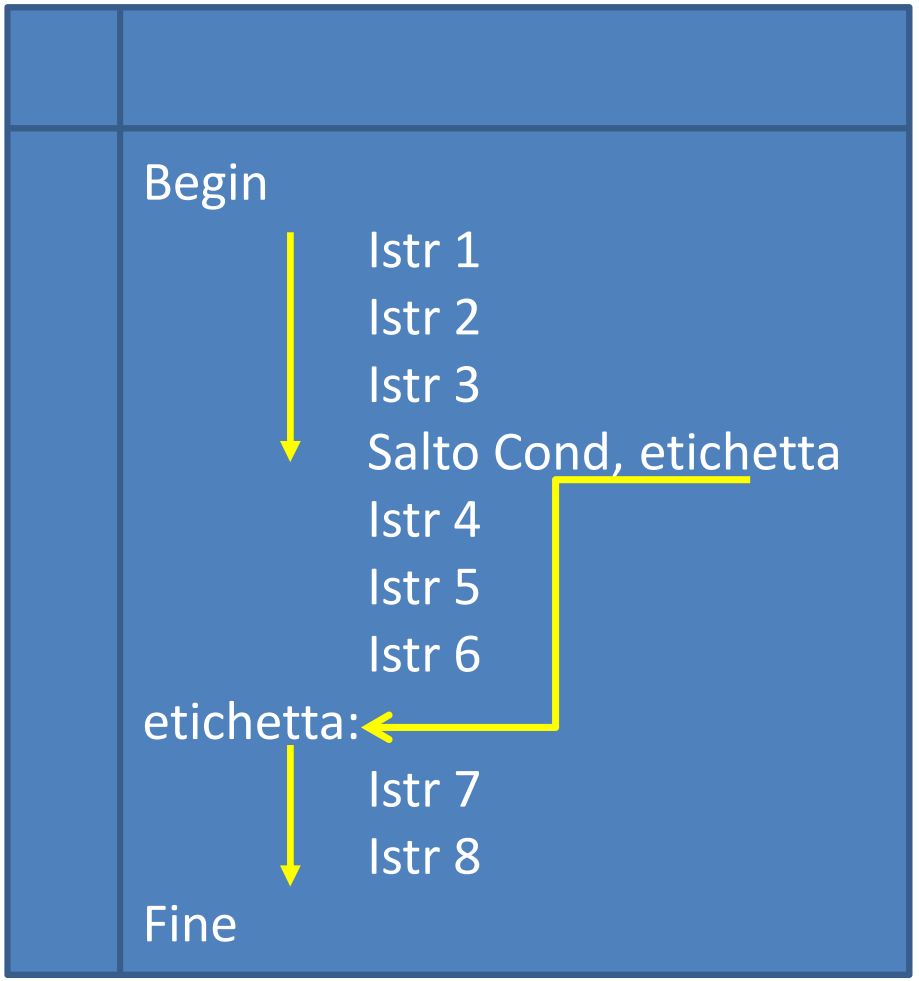
Architettura degli elaboratori

Istruzioni di salto

ISTRUZIONI DI SALTO

Generalità

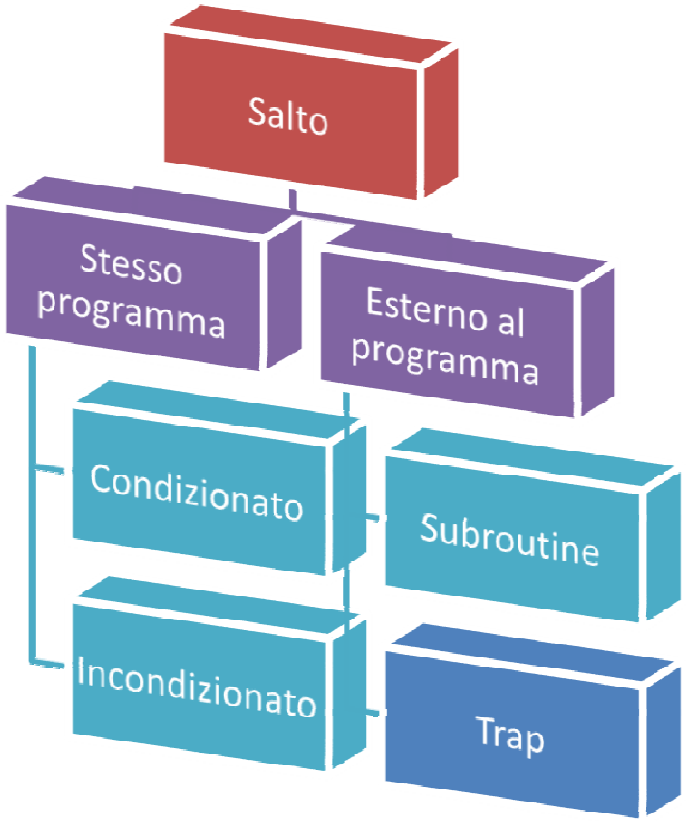
- Le **istruzioni di salto** individuano una classe particolare in quanto non agiscono direttamente sui dati, ma servono per modificare l'ordine sequenziale di esecuzione delle istruzioni del programma



ISTRUZIONI DI SALTO

Classificazione

- Le istruzioni di salto si dividono in:
  - salto all'interno dello stesso programma
    - **condizionato**: il salto è eseguito in base ad una certa condizione stabilita dal programmatore (Branch)
    - **incondizionato**: il salto è sempre eseguito (Jump), senza valutare alcuna condizione
  - salto ad un altro programma: **salto a subroutine** (salto a sottoprogramma)
  - ***trap*** (o interruzioni software)



ISTRUZIONI DI SALTO

Condizionato e incondizionato

ESEMPIO SALTO CONDIZIONATO

La decodifica ed esecuzione di una istruzione di *brach*,

BEQ $t0,$t1, 0x100,

può essere così descritta

Decodifica:

Unità di Controllo ← BEQZ

Esecuzione:

ACC ← $t0-$t1

Se Z=1 ⇒ PC← 0x100

Se Z=0 ⇒ *non fa nulla*

ESEMPIO SALTO INCONDIZIONATO

La decodifica ed esecuzione di una istruzione di *jump*,

J 0x100

può essere così descritta

Decodifica:

Unità di Controllo ← J

Esecuzione:

PC ← 0x100

ISTRUZIONI DI SALTO

Salto condizionato

- Le istruzioni di salto sono fondamentali perché rompono la sequenzialità offrendo la possibilità di **effettuare scelte**, cioè prendere decisioni e perché offrono l'**iterazione**, cioè consentono di eseguire più volte una parte di programma (es.: il ciclo while)

CODICE	OPERANDI	Commento
BEQZ	<Sorg1>, Indirizzo	Se l'operando contenuto in una sorgente (registro/memoria) è uguale a zero salta all'indirizzo specificato
BGT	<Sorg1>,<Sorg2>,Indirizzo	Legge gli operandi dalla sorgente (memoria/registri) e salta all'indirizzo se Sorg1 è maggiore della Sorg2
BLT	<Sorg1>,<Sorg2>,Indirizzo	Legge gli operandi dalla sorgente (memoria/registri) e salta all'indirizzo se Sorg1 è minore della Sorg2
J	Indirizzo	Salto incondizionato all'indirizzo specificato

ISTRUZIONI DI SALTO

Salto condizionato

Le istruzioni di salto condizionato, pertanto, richiedono l'analisi dei condition code

Menmonico	Significato	Flag
EQ	Uguale	Z=1
NEQ	Non uguale	Z=0
BGE	Maggiore o uguale (senza considerare il segno)	C=1
BGE	Maggiore o uguale (considerando il segno)	N=W
BGT	Maggiore (senza considerare il segno)	C= 1 and Z=0
BGT	Maggiore	Z=0 or N=W
BLE	Minore o uguale (senza considerare il segno)	C= 0 or Z=1
BLE	Minore o uguale (considerando il segno)	Z=1 or N!=W
BLT	Minore (senza considerare il segno)	C=0
BLT	Minore	N!=W

ISTRUZIONI DI SALTO

Esempio: calcolo del massimo

	Operando1	Operando2	Massimo	R0	R1	R2
.DATA						
operando1: WORD 327	327					
operando2: WORD 45968	327	45968				
Massimo: WORD 0	327	45968	0			
.TEXT						
LOAD.W R0, operando1	327	45968	0	327		
LOAD.W R1, operando2	327	45968	0	327	45968	
MOVE R2,R0	327	45968	0	327	45968	327
BGT R0,R1, SALTO	327	45968	0	327	45968	327
MOVE R2,R1	327	45968	0	45968	45968	45968
SALTO:						
STORE.W R2,Massimo	327	45968	45968	45968	45968	45968

ISTRUZIONI DI SALTO

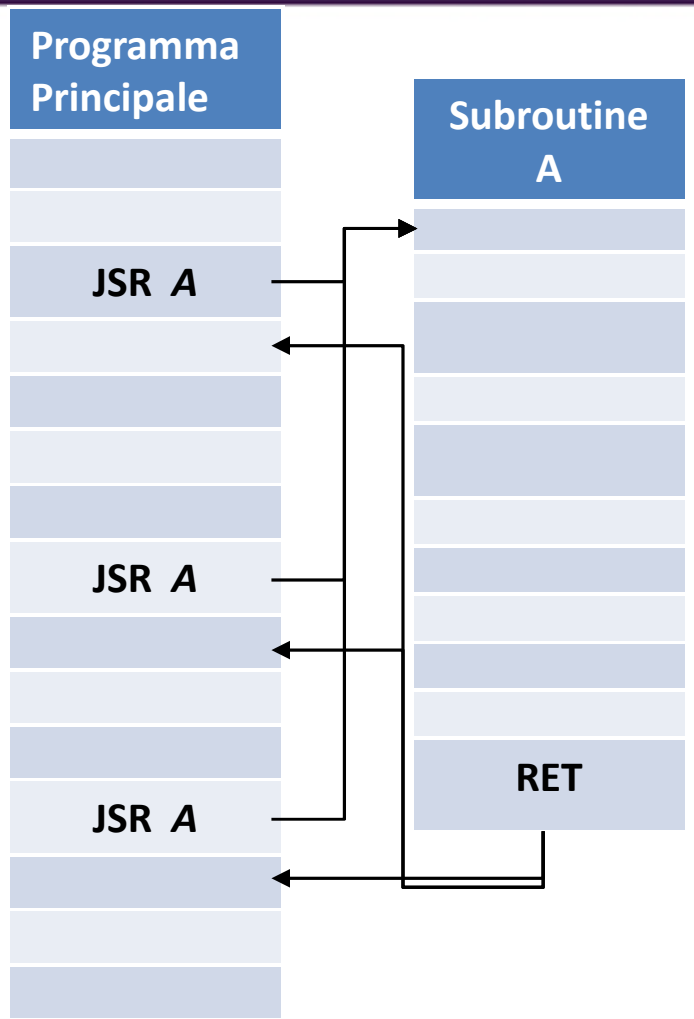
Esempio: calcolo del massimo (condizione non verificata)

	Operando1	Operando2	Massimo	R0	R1	R2
.DATA						
operando1: WORD 3270000	3270000					
operando2: WORD 45968	3270000	45968				
Massimo: WORD 0	3270000	45968	0			
.TEXT						
LOAD.W R0, operando1	3270000	45968	0	3270000		
LOAD.W R1, operando2	3270000	45968	0	3270000	45968	
MOVE R2,R0	3270000	45968	0	3270000	45968	3270000
BGT R0,R1, SALTO	3270000	45968	0	3270000	45968	3270000
MOVE R2,R1						
SALTO:						
STORE.W R2,Massimo	3270000	45968	3270000	3270000	45968	3270000

ISTRUZIONI DI SALTO

Salto a subroutine

- L'**istruzione di salto a subroutine** (o chiamata a funzione) permette di saltare da un programma (il programma principale) ad un sottoprogramma, di eseguirlo e di tornare alla istruzione immediatamente successiva a quella di chiamata
- L'utilizzo di subroutine è utile quando un determinato insieme di istruzioni deve essere eseguito più volte e per avere un codice più chiaro e compatto. Inoltre le subroutine possono essere realizzate da terzi, essere scambiate e modificate ai propri fini



ISTRUZIONI DI SALTO

Salto a subroutine

CODICE	OPERANDI	Commento
JSR	Indirizzo	Salva il valore del PC incrementato nello Stack e salta all'indirizzo specificato che individua l'inizio del sottoprogramma
RET		Ritorna al programma principale ripristinando il valore del PC recuperato nello stack

ISTRUZIONI DI SALTO

Salto a subroutine

SALTO A SUBROUTINE

La decodifica ed esecuzione di una istruzione di *salto a funzione*,

JSR etichetta_subroutine

può essere così descritta (se la sub routine è alla posizione 0x100) :

Decodifica:

Unità di Controllo ← JSR 0x100

Esecuzione:

(SP) ←PC+1 # istruzione successiva
SP ←SP-1 #spostamento stack
PC ←0x100 #salto

NB: equivalente ad una PUSH

RITORNO DA SUBROUTINE

La decodifica ed esecuzione di una istruzione di ritorno da subroutine

RET

può essere così descritta

Decodifica:

Unità di Controllo ← RET

Esecuzione:

SP← SP+1 #decremento stack
PC← (SP) #estrazione del PC conservato
#nello stack

NB: equivalente ad una POP

ISTRUZIONI DI SALTO

Esempio: calcolo del massimo (realizzato con subroutine)

Indirizzo	FUNZIONE PRINCIPALE		Indirizzo	SUBROUTINE	
0	.DATA				
4	operando1: WORD 327169				
8	operando2: WORD 45968				
12	Massimo: WORD 0				
16	.TEXT				
20	**LOAD.W** R0, operando1		400	***MASSIMO:***	
24	**LOAD.W** R1, operando2		404		**MOVE** R3,R0
28	**JSR *MASSIMO***		408		**BGT** R0,R1, SALTA
32	**STORE.W** R3,Massimo		412		**MOVE** R3,R1
36	END		416	*SALTA:*	
			420		**RET**

PC	SR
16	-
20	-
24	-
28	32
400	32
404	32
408	32
420	32
32	-
36	-

ISTRUZIONI DI SALTO

Salto a subroutine MIPS

SALTO A SUBROUTINE MIPS

In MIPS un salto a subroutine è ottenuto salvando il valore del PC in un registro speciale **$ra**

Così

JAL etichetta_subroutine

può essere così descritta (se la sub routine è alla posizione 0x1000) :

Decodifica:

Unità di Controllo ← JAL 0x1000

Esecuzione:

$RA ←PC+1 # istruzione successiva

PC ←0x1000 #salto

RITORNO DA SUBROUTINE

La decodifica ed esecuzione di una istruzione di ritorno da subroutine

JR $ra

può essere così descritta

Decodifica:

Unità di Controllo ← JR $ra

Esecuzione:

PC← ($ra) #aggiornamento PC con #indirizzo di ritorno

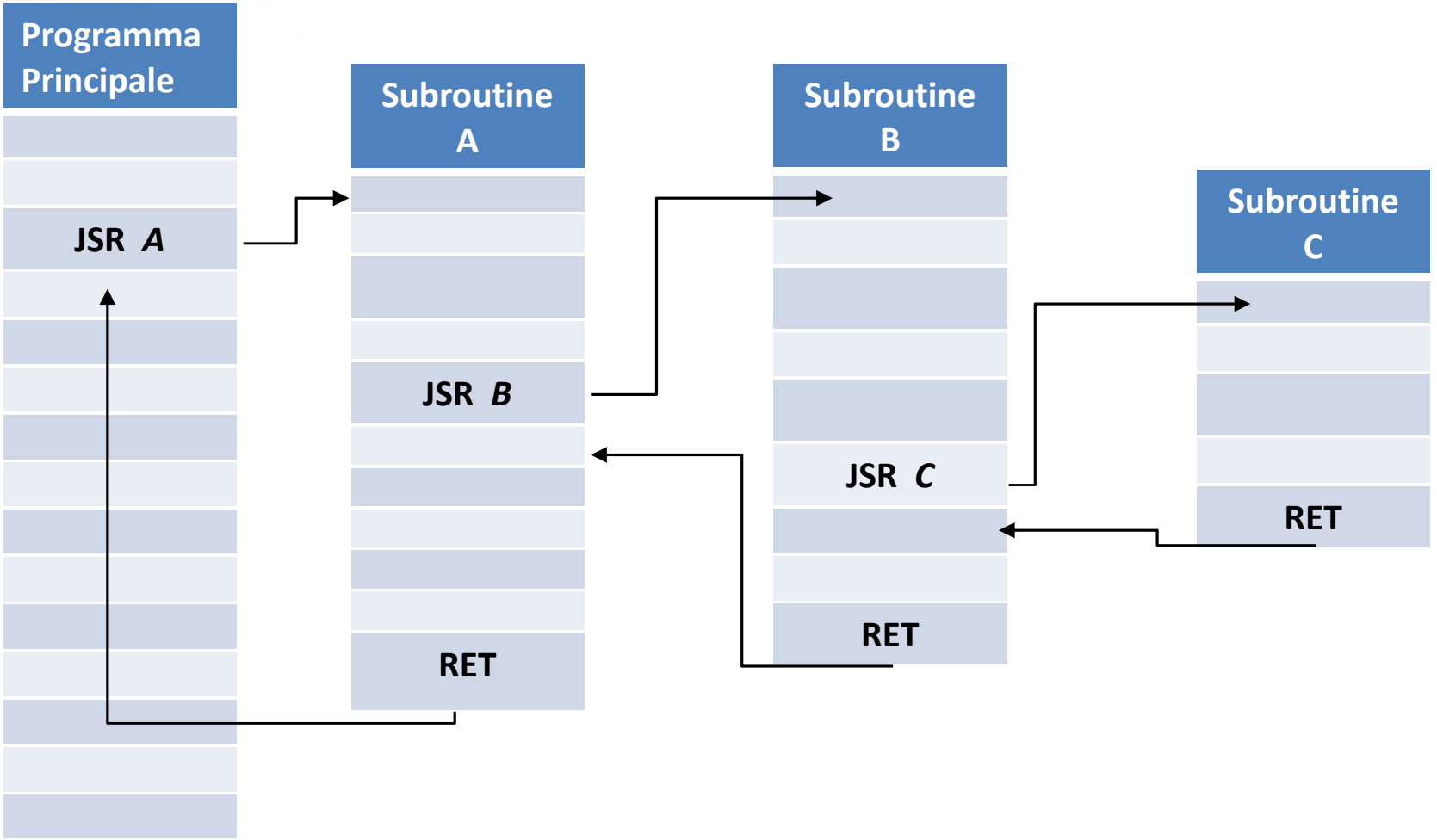
ISTRUZIONI DI SALTO

Annidamento di subroutine

- Molto spesso però i sottoprogrammi possono a loro volta chiamare altri programmi e così via. Può avverarsi cioè un **annidamento di subroutine** (*nested subroutine*)

ISTRUZIONI DI SALTO

Annidamento di subroutine



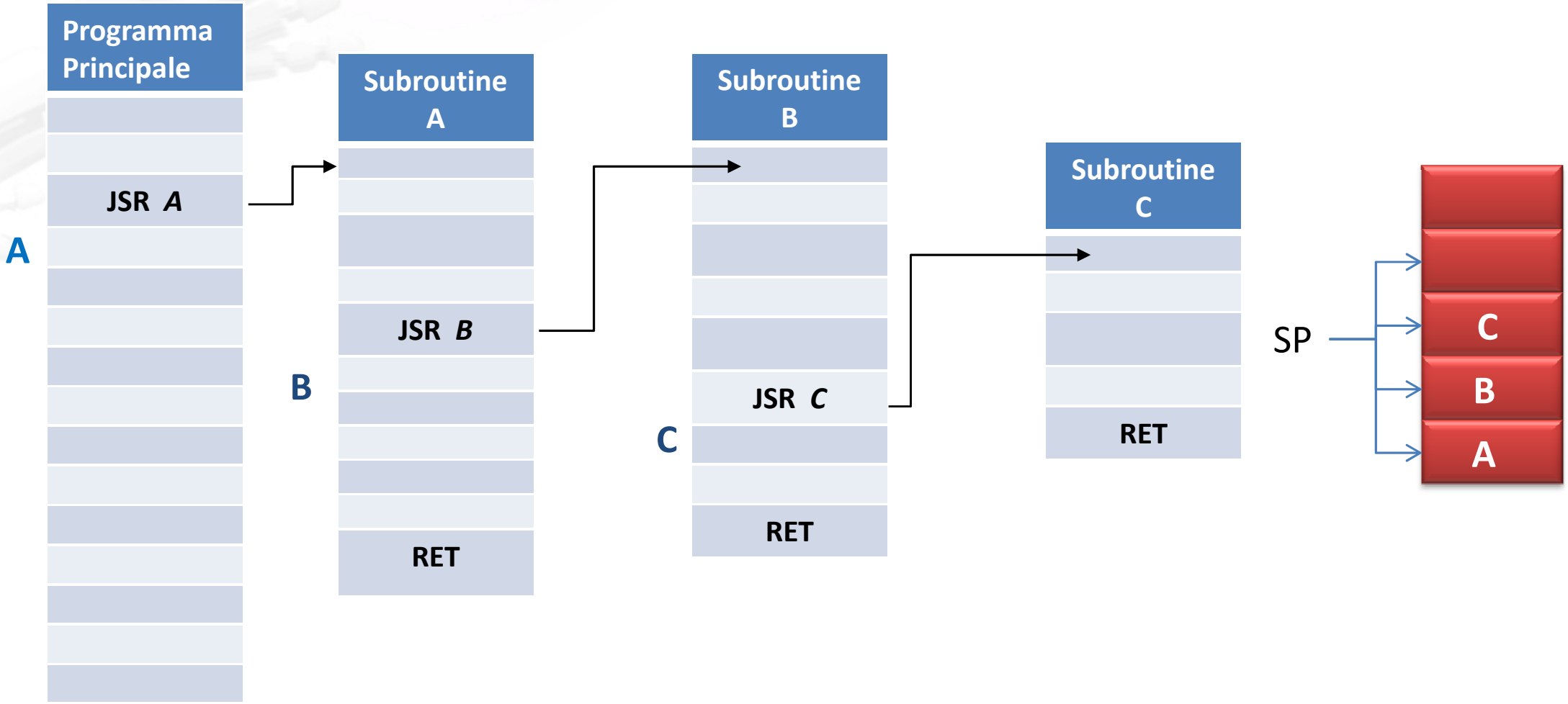
ISTRUZIONI DI SALTO

Annidamento di subroutine

- La gestione di funzioni ricorsive o l'annidamento di funzioni è gestito grazie all'utilizzo della **pila** (**stack** o *canasta*)
  - Nel caso di un numero non determinabile di chiamate a subroutine è fondamentale salvare l'indirizzo di ritorno nello stack
- Lo stack è una zona di memoria riservata per il passaggio di parametri e la memorizzazione di informazioni gestita nella **modalità LIFO** (*Last in First Out*): ovvero l'ultimo elemento immesso nella pila è anche il primo ad uscire

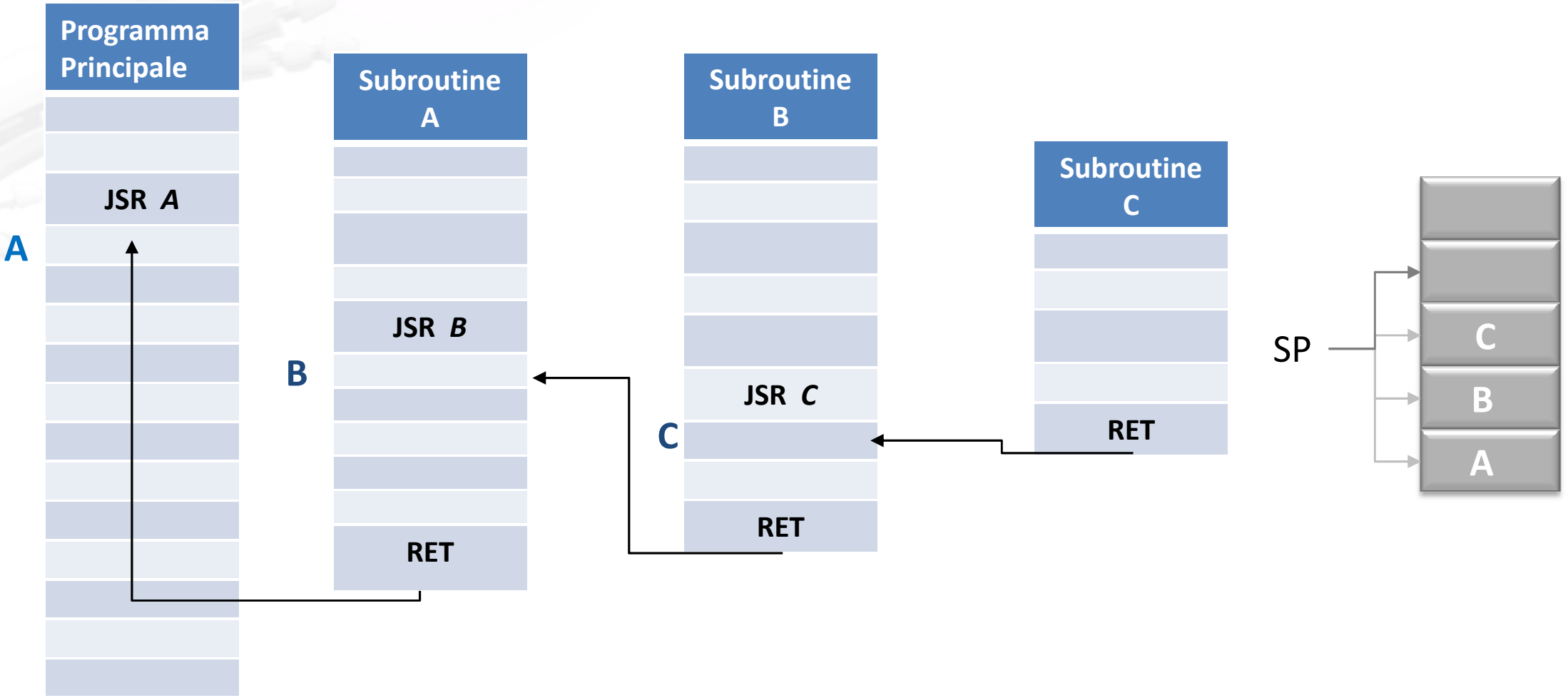
ISTRUZIONI DI SALTO

Annidamento di subroutine



ISTRUZIONI DI SALTO

Annidamento di subroutine



Architettura degli elaboratori

Istruzioni I/O

ISTRUZIONI I/O

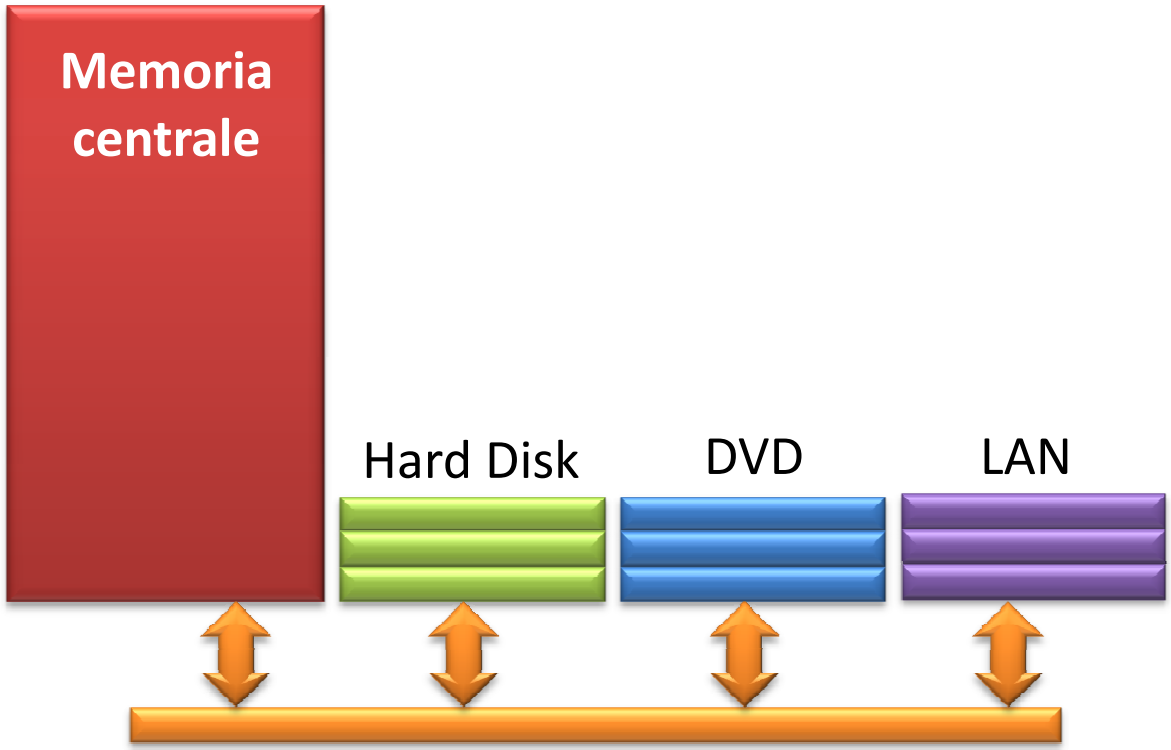
Generalità

- Per interagire con i dispostivi di I/O si può ricorre ad un **set di istruzioni dedicato** o si può riservare un'**area di memoria agli scambi con i dispositivi** di I/O (IO a porte, *port-mapped* I/O) ed operare con le istruzioni della macchina (IO programmato)

ISTRUZIONI I/O

Generalità

- Nel caso di un **set di istruzioni dedicato** (IO port-mapped) si specificano le locazioni riservate per ogni periferica e inoltre le operazioni da svolgere (scrittura/lettura), il dato che deve essere scambiato e l'indirizzo in cui bisogna posizionare o da cui è necessario prelevare l'informazione



ISTRUZIONI I/O

ISTRUZIONI: IN OUT Intel x86

- Istruzioni Input/Ouput Port x86

IN	
Sintassi	**Significato**
IN {b,w,l} [AL|AX|EAX], <port_address>	Trasferisce un dato dal dispositivo identificato dall'indirizzo <port_address> in un registro <AL|AX|EAX > a seconda della grandezza del dato (8bit AL, 16bit AX e 32bit EAX)
Esempi	
INb 255	Trasferisce il dato di 8bit presente lungo la periferica il cui indirizzo è 255 nel registro AL
INw (%DX)	Trasferisce il dato di 16bit presente lungo la periferica il cui indirizzo è specificato nel registro DX al registro AX

ISTRUZIONI I/O

ISTRUZIONI: IN OUT Intel x86

- Istruzioni Input/Ouput Port x86

OUT	
Sintassi	**Significato**
OUT {b,w,l} [AL|AX|EAX], <port_address>	Trasferisce un dato dal registro <AL|AX|EAX > a seconda della grandezza del dato (8bit AL, 16bit AX e 32bit EAX) al dispositivo identificato dall'indirizzo <port_address>
Esempi	
OUTw 255	Trasferisce il dato di 16bit contenuto nel registro AX alla periferica con indirizzo 255
OUTl (%DX)	Trasferisce il dato di 32bit nel registro EAX alla periferica il cui indirizzo è specificato nel registro DX

ISTRUZIONI I/O

I/O IBM1130: set istruzione dedicato

- Il processore IBM 1130 utilizzava l'**istruzione XIO** per interagire con le periferiche

XIO <address>

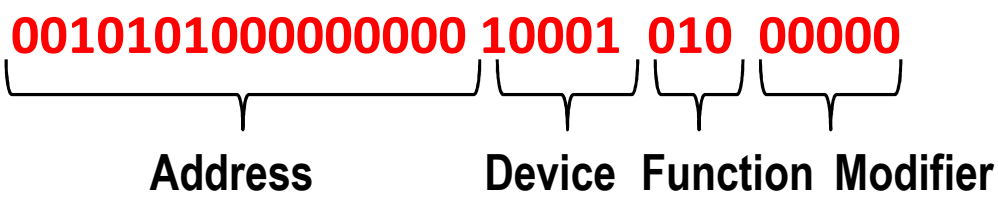
- XIO specifica un indirizzo <address> in cui è presente un **Input/Output Control Commands** (IOCC's) ovvero un codice con dei campi in cui si specificava:
  - **Address**: l'indirizzo in cui volere trasferire il dato
  - **Device**: il dispositivo con cui si voleva interagire (es.: 00001: tastiera;00010, lettore di schede perforate; 00110, stampante; 10001, Unità disco magnetico IBM2311)
  - **Function**: l'operazione da compiere (es.: 001, *write*:trasferimento di una parola dalla memoria alla periferica; 010, *read*: trasferimento di una parola dalla periferica alla memoria)
  - **Modifier**: campo per informazioni supplementari o specifiche funzioni relative al dispositivo (ad esempio lo spostamento della testina di una disco magnetico da una traccia ad un'altra)

ISTRUZIONI I/O

I/O IBM1130: set istruzione dedicato

XIO <addres>||<register>

XIO 1000



Lettura di una parola proveniente dall'Unità a disco magnetico e posizionamento all'indirizzo 10752 (0010101000000000)

Spostamento di una parola dall'Unità Disco magnetico
Alla locazione di memoria 10752

ISTRUZIONI I/O

I/O Programmato: ARM Cortex-M

- Nell'IO programmato si riservano delle aree della memoria ai diversi dispositivi e lo scambio dell'informazione avviene mediante le semplici operazioni di trasferimento

Esempio:

DEV1 EQU 0x40010000

#Definizione dell'indirizzo del DISPOSITIVO

LDR r1,DEV1

#Caricamento dell'indirizzo nel registro R1

LDRB r0,[r1]

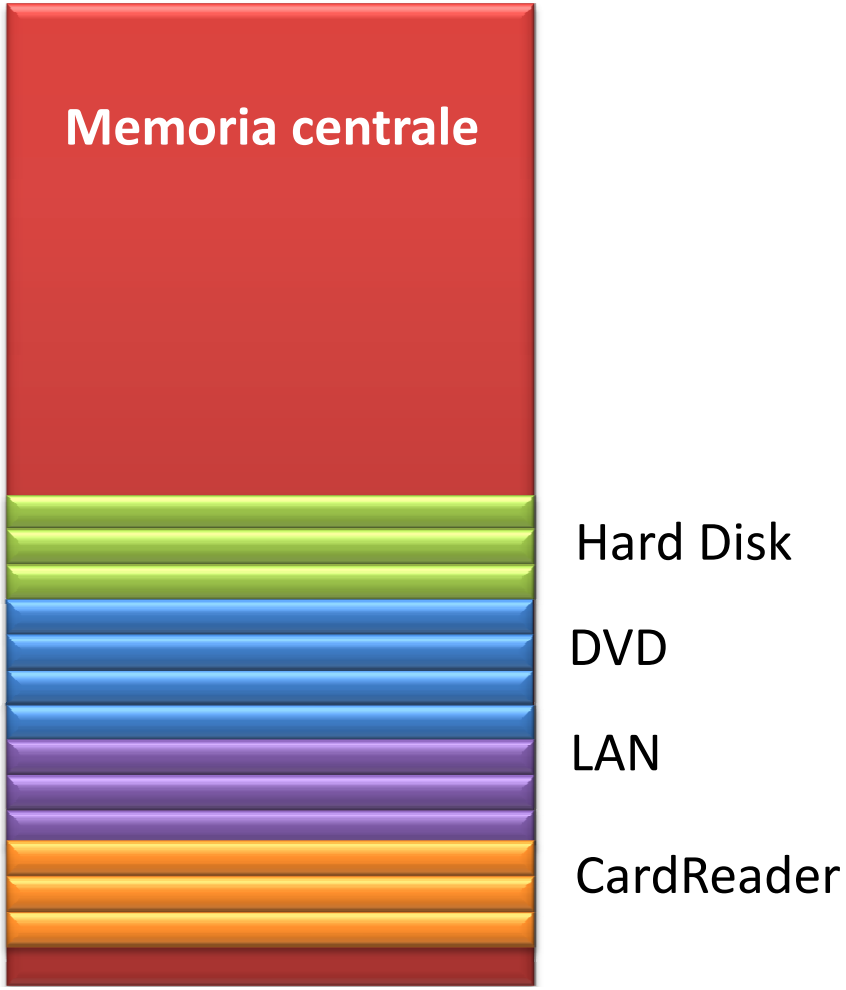
#Lettura di un byte dal DISPOSITIVO1

MOV r0,#8

#Impostazione del valore 8 in R0

STRB r0,[r1]

#Scrittura del valore nel dispositivo



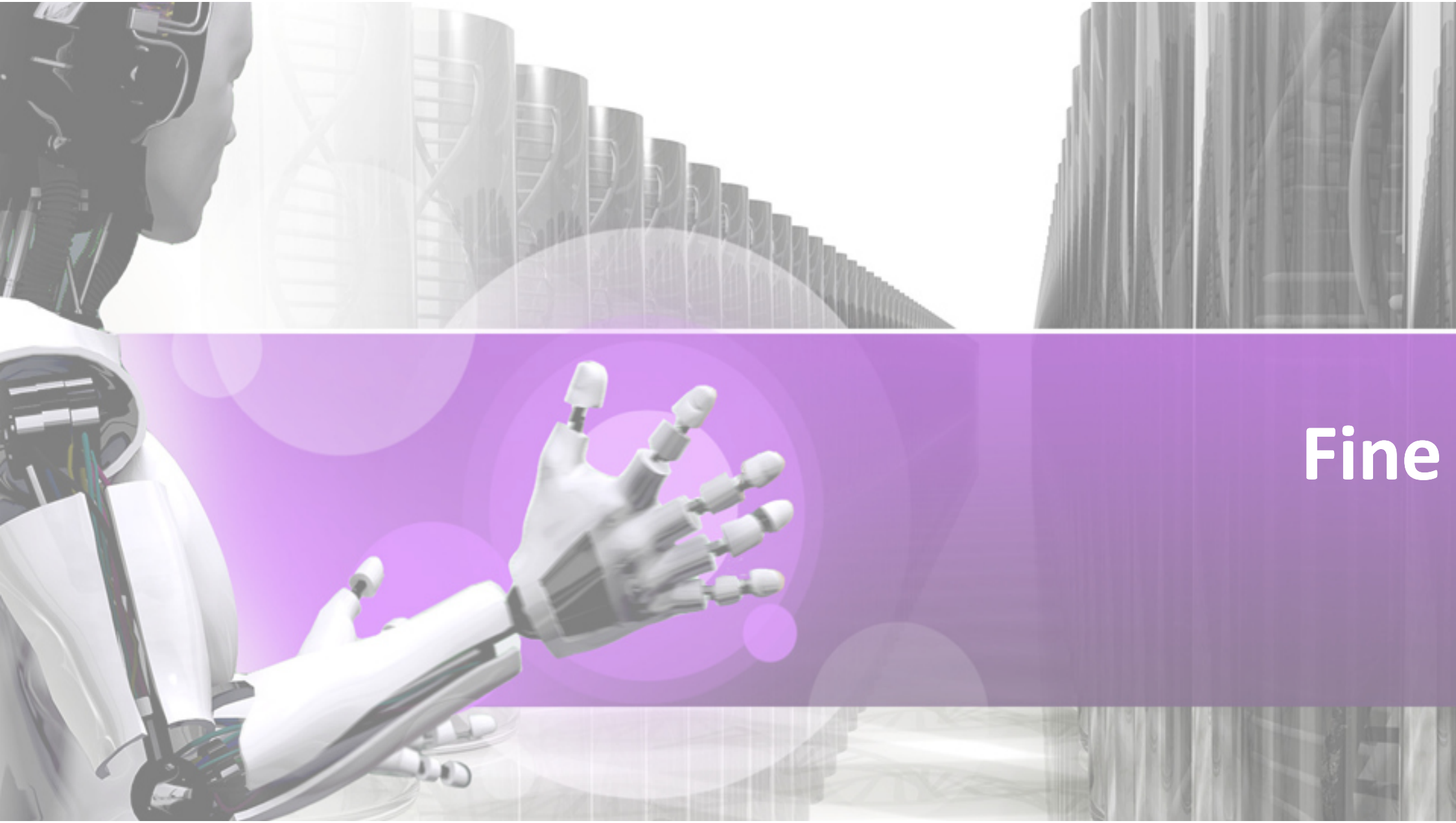
Architettura degli elaboratori

Istruzioni di controllo della macchina

ISTRUZIONI CONTROLLO MACCHINA

- Le **istruzioni di comando** (o istruzioni di controllo macchina) non operano né sui dati né sui registri né interessano il contatore di programma, ma intervengono direttamente sullo stato della CPU
- Le istruzioni di comando sono caratteristiche di ogni CPU: il loro numero può variare da poche unità, per macchine semplici, a decine per macchine complesse

CODICE	Commento
HALT	Interruzione di sistema
NOP	Nessuna operazione. ***È utile per il Delay Slot del pipeling***
BREAK	Interruzione di programma


Fine