

La macchina programmata Instruction Set Architecture

**Schema base di esecuzione
Istruzioni macchina**

Claudia Raibulet
claudia.raibulet@unimib.it

00000001000010010101000000100000

add \$10, \$8, \$9

$X + Y = Z$ oppure $Z = X + Y$

Argomenti per il primo compitino (I)

- **Rappresentazione dell'informazione**
 - sistemi numerici
 - rappresentazione dell'informazione non numerica,
 - rappresentazione dei numeri interi con e senza segno,
 - rappresentazione dei numeri in virgola fissa e mobile.
- **Instruction Set Architecture**
 - schema di von Neumann,
 - CPU, registri, ALU e memoria,
 - ciclo fondamentale di esecuzione di una istruzione (fetch/decode/execute),
 - tipi e formati di istruzioni MIPS32,
 - modalità di indirizzamento.



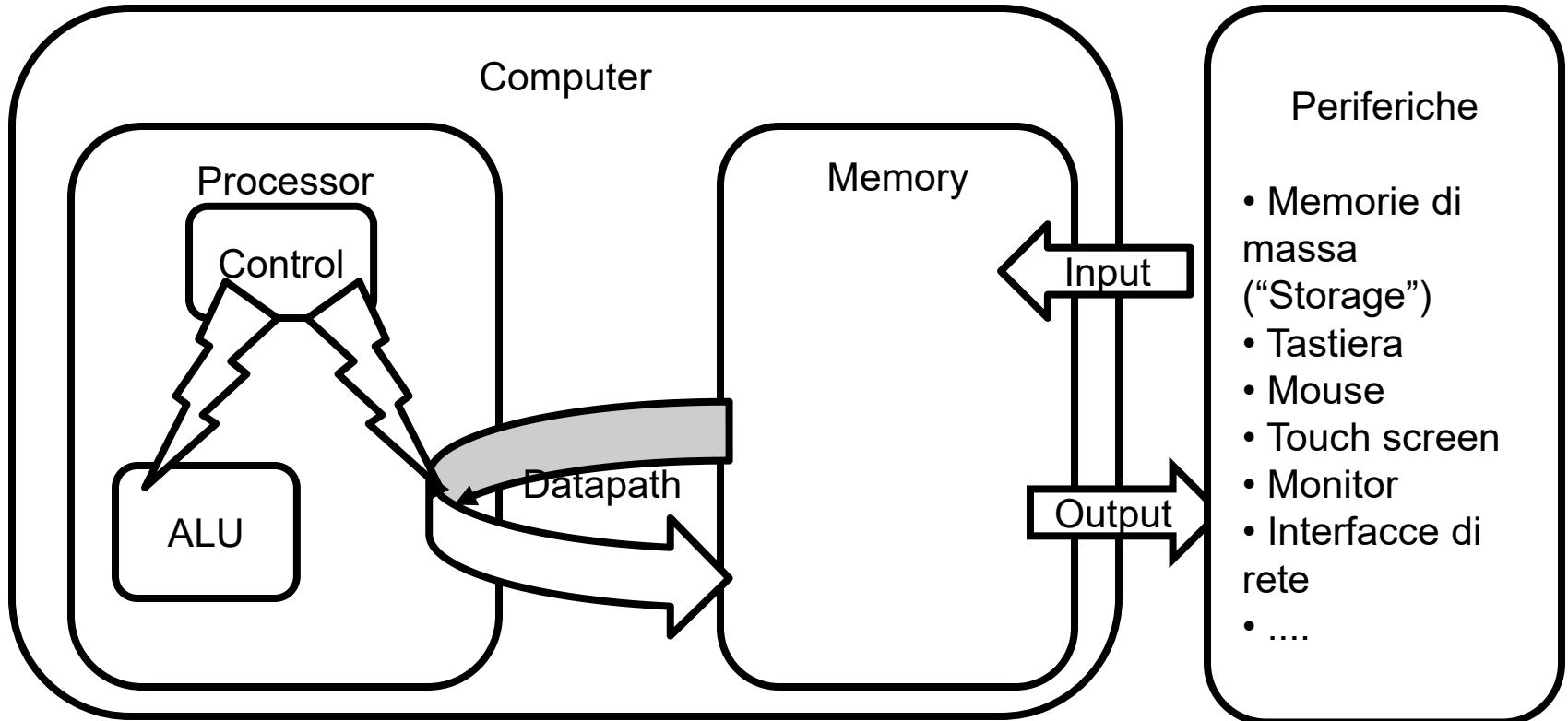
Argomenti per il primo compitino (II)

- **Linguaggio Assembly**
 - formato simbolico delle istruzioni,
 - catena di programmazione (compilatore, assembler, linker, loader, etc.),
 - pseudo-istruzioni e direttive dell'assemblatore,
 - scrittura di semplici programmi assembly,
 - convenzioni programmatiche (memoria, nomi dei registri, etc.).
- **Circuiti logici**
 - reti combinatorie,
 - reti sequenziali e FSM (Finite State Machine),
 - rassegna di circuiti notevoli (decoder, multiplexer, register file, ALU, etc.).

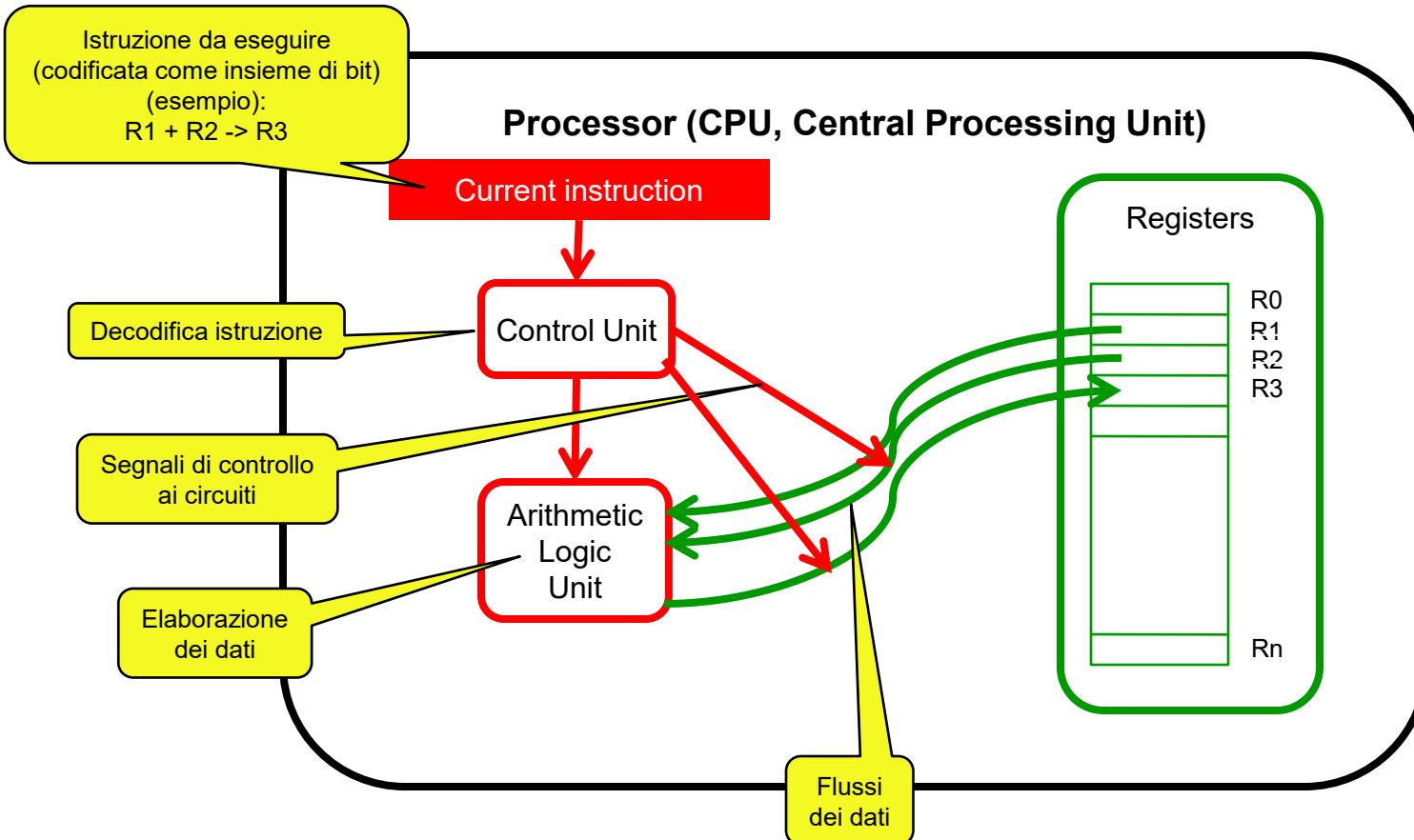
Outline

- **Componenti di un computer (“big picture”)**
- **Tre aspetti:**
 - **Cosa fa (“Instruction Set Architecture”, ISA)**
 - **Come si programma (Assembly language)**
 - **Come è fatto (circuiti e datapath)**
- **Esecuzione di una istruzione**
 - Schema di principio MOLTO semplificato (ma generale!)
 - Esempio: somma tra registri
- **Esecuzione di un programma memorizzato**
 - Ciclo elementare (fetch/execute)
- **Formato delle istruzioni**
 - Esempio: istruzioni MIPS R-type
- **Rappresentazione simbolica (Assembly)**
 - Catena programmativa
- **Simulatori: SPIM, QtSPIM, MARS**

“The big picture”



Processor: istruzione elementare su registri



Commenti sulla slide precedente

- E' uno schema di principio ultra-semplificato ...
- ...ma cattura gli aspetti fondamentali del comportamento di un processore ("Von Neumann Machine")
- La sequenza delle operazioni è puramente logica
- A livello ISA "si vedono" Registri e Memoria (vedi seguito)
- **Hardware: circuiti combinatori e sequenziali**
 - Vedere lezioni future su circuiti, datapath ecc.
 - Circuiti pilotati dal clock
 - Tempo di esecuzione di una istruzione: "almeno" un ciclo di clock
 - Frequenza del clock: $\approx 1 \text{ GHz} = 10^9 \text{ Hz}$
 - *Ragionare sugli ordini di grandezza dei tempi!!!*
- **Software: interprete (e.g., SPIM)**
- **Memoria: un insieme di "celle"**
 - Che contengono bit...
 - interpretabili in modi diversi (dati o istruzioni)

Commenti:

- **Registri di CPU:**
 - *dentro* il processore
 - pochi (e.g., 32)
 - veloci (coincide con il clock)
 - costosi
 - NB: questi sono quelli che si chiamano di solito “Registri”
- **Memoria**
 - grande (e.g., 1GB = 1000000000 byte)
 - (relativamente) lenta (alcuni cicli di clock)
 - mediamente costosa
 - NB: anche la memoria è un insieme di registri, chiamati di solito “celle” o “locazioni”
- **Memoria di massa (“Storage”)**
 - e’ una periferica
 - molto grande (da 100 GigaB a TeraByte)
 - lenta (millisecondi)
 - poco costosa
 - persistente

- **RISC (Reduced Instruction Set Computing)**
 - Poche istruzioni semplici
 - Struttura circuitalmente semplice
 - Esecuzione veloce di una singola istruzione
 - Occorrono più istruzioni per fare cose anche semplici
 - Esempio: **MIPS**, ARM, PowerPC...
 - MIPS (Microprocessor without Interlocked Pipeline Stages)
- **CISC (Complex Instruction Set Computing)**
 - Istruzioni complesse
 - Struttura circuitalmente complicata
 - Esecuzione più lenta di una singola istruzione
 - Occorrono meno istruzioni
 - Esempio: Intel x86 e tutti i derivati (Pentium ecc.)
- **Nel seguito del corso: MIPS (come esempio!)**
 - SPIM = **simulatore sw** di MIPS
 - versioni per diversi sistemi (Windows, Mac, Linux)
 - MARS = **simulatore sw** di MIPS
 - Scritto in Java (portabile su diversi sistemi)

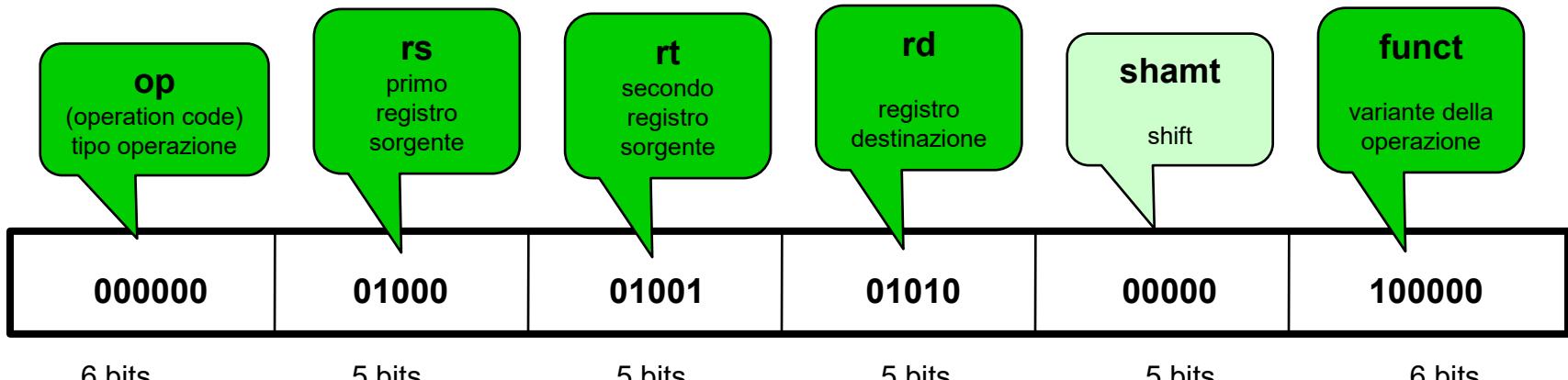
Architettura MIPS e linguaggio macchina

- **RISC: semplice, uniforme e facile da capire**
- **32 registri di 32 bit**
- **Istruzioni di 32 bit**
- **Quali operazioni devono essere rappresentate dalle istruzioni?**
- **Manipolazioni di dati solo sui registri**
- **Trasferimento di dati tra memoria e registri**
- **Alterazione del flusso di controllo (“salti”)**
- **Problema architettonico:**
 - come esprimere tutti i tipi di operazioni in 32 bit...
 - ...mantenendo il più possibile omogenea la struttura delle istruzioni?
- **Tre formati delle istruzioni**
 - R-type
 - I-type
 - J-type
 - **Attenzione:** i tre formati non coincidono esattamente con le tre tipologie di operazioni

Istruzione R-Type

quello che si vuole fare (in italiano)
numeri decimali (per comodità)

“somma i contenuti del registro 8 e del registro 9 e metti il risultato nel registro 10”



00000001000010010101000000100000
I bit che sono in memoria e sono trasferiti a UC

L'Unità di Controllo “sa” come interpretare i singoli campi

0000 0001 0000 1001 0101 0000 0010 0000

gli stessi spaccettati in quartetti (il computer “non lo sa”)

0x 1 0 9 5 0 2 0

rappresentazione esadecimale utile (ahimè) per gli umani

Formato simbolico (assembly)

- Un formato un po' più comodo
- “**linguaggio sorgente**” (un programma è un testo)
- **add \$10, \$8, \$9**
 - codici operativi simbolici
 - nomi simbolici di registri (\$xxx)
 - nomi simbolici di variabili (che diventeranno indirizzi)
- **(attenzione all'ordine degli operandi...)**
- usato da chi programma
- qualcuno (e.g., un programma **assemblatore**) traduce da **sorgente** a **esegibile**
- **quello che viene caricato in memoria ed eseguito è sempre l'eseguibile (codice macchina)**
- **Il simulatore si occupa anche di queste cose, ma non facciamo confusione fra linguaggio sorgente e linguaggio macchina!!!**

Importante!

- **Appendice A – pagina 24 – registri**
- **Appendice A – pagina 50 – opcode/funct per le istruzioni MIPS**

Register name	Number	Usage
\$zero	0	constant 0
\$at	1	reserved for assembler
\$v0	2	expression evaluation and results of a function
\$v1	3	expression evaluation and results of a function
\$a0	4	argument 1
\$a1	5	argument 2
\$a2	6	argument 3
\$a3	7	argument 4
\$t0	8	temporary (not preserved across call)
\$t1	9	temporary (not preserved across call)
\$t2	10	temporary (not preserved across call)
\$t3	11	temporary (not preserved across call)
\$t4	12	temporary (not preserved across call)
\$t5	13	temporary (not preserved across call)
\$t6	14	temporary (not preserved across call)
\$t7	15	temporary (not preserved across call)
\$s0	16	saved temporary (preserved across call)
\$s1	17	saved temporary (preserved across call)
\$s2	18	saved temporary (preserved across call)
\$s3	19	saved temporary (preserved across call)
\$s4	20	saved temporary (preserved across call)
\$s5	21	saved temporary (preserved across call)
\$s6	22	saved temporary (preserved across call)
\$s7	23	saved temporary (preserved across call)
\$t8	24	temporary (not preserved across call)
\$t9	25	temporary (not preserved across call)
\$k0	26	reserved for OS kernel
\$k1	27	reserved for OS kernel
\$gp	28	pointer to global area
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address (used by function call)

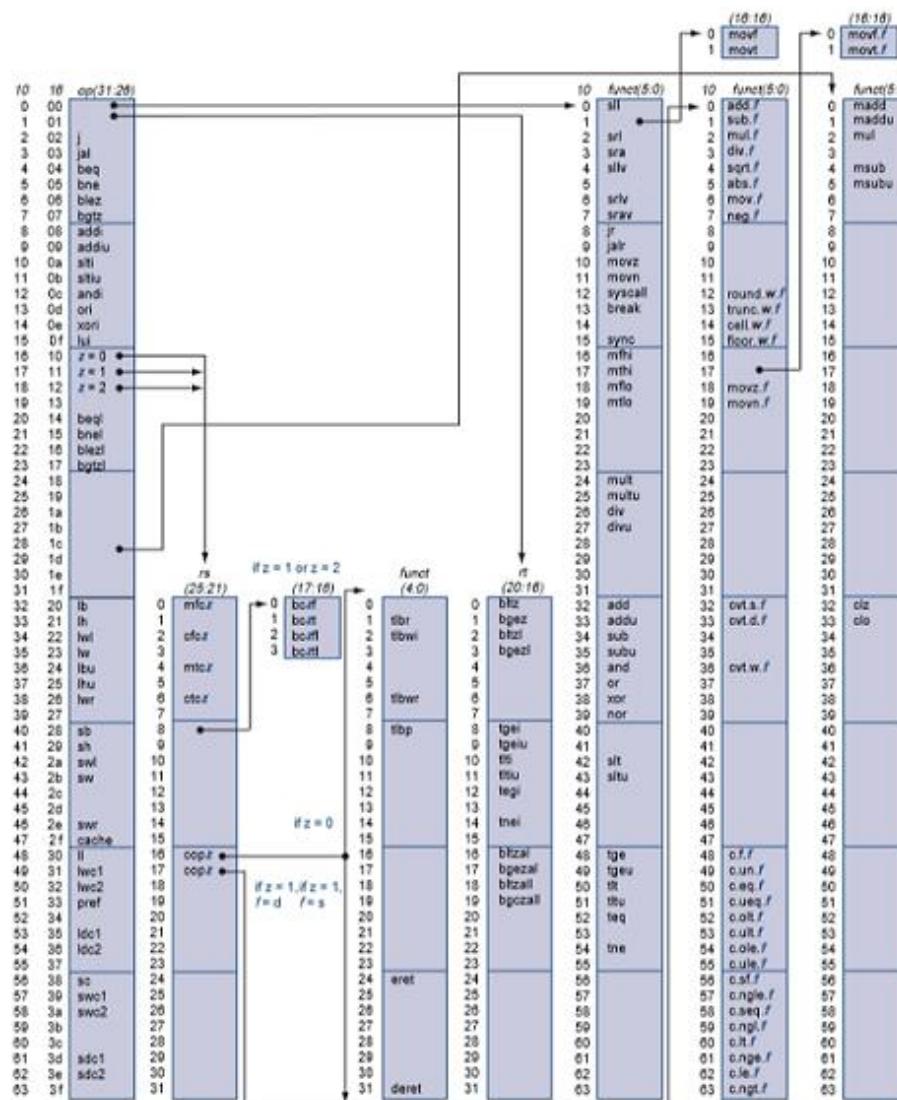


FIGURE A.10.2 MIPS opcode map. The values of each field are shown to its left. The first column shows the values in base 10 and the second shows base 16 for the op field (bits 31 to 26) in the third column. This op field completely specifies the MIPS operation except for 6 op values: 0, 1, 16, 17, 18, and 19. These operations are determined by other fields, identified by pointers. The last field (funct) uses “f” to mean “s” if rs = 16 and op = 17 or “d” if rs = 17 and op = 17. The second field (rs) uses “z” to mean “0”, “1”, “2”, or “3” if op = 16, 17, 18, or 19, respectively. If rs = 16, the operation is specified elsewhere; if z = 0, the operations are specified in the fourth field (bits 4 to 0); if z = 1, then the operations are in the last field with f = s. If rs = 17 and z = 1, then the operations are in the last field with f = d.

Esercizio 1

- Si chiede di calcolare la differenza tra i contenuti del registro 8 e del registro 9 e di mettere il risultato nel registro 10. Si chiede di specificare la rappresentazione binaria ed esadecimale dell'istruzione.
- **LINK:**
- <https://forms.gle/DjPMCKZeqUr8ykmj6>

Esercizio 1 - Soluzione

- Si cerca nel file Appendice A l'istruzione per il calcolo della differenza tra due valori che si trovano nei registri; tale istruzione si chiama sub (subtract).
- Si nota che il formato e' molto simile all'operazione add.
- Quindi **sub rd, rs, rt** va tradotta nel nostro caso in **sub \$10, \$8, \$9**.
- Si cerca nel file Appendice A il opcode e il funct code alla pagina 50. Essendo una versione dell'operazione add il opcode e' sempre 0 (rappresentato su 6 bit), mentre il funct code e' 34 in decimale, cioe' 100010 in binario su 6 bit.
- Dobbiamo codificare in binario i 3 registri (cioe' il numero di ogni registro, che rappresenta il nome del registro).
- **rd = 10**, quindi 01010 in binario su 5 bit
- **rs = 8**, quindi 01000 in binario su 5 bit
- **rt = 9**, quindi 01001 in binario su 5 bit
- Attenzione: sub rd, rs, rt -> ma nella rappresentazione in binario l'ordine dei registri e' diverso
- shamt e' zero anche in questo caso

sub \$10, \$8, \$9
000000 01000 01001 01010 00000 100010
0x01095022 ISA (1)

Esercizio 2

- Considerando che:
- Opcode=0x0
- rs=0xD
- rt=0xE
- rd=0xF
- Shamt=0x0
- Funct code=0x20
- Si chiede di indicare l'istruzione corrispondente e rappresentarla in binario e esadecimale.

- LINK:
- <https://forms.gle/aknSWsFtXrNowmYF6>

Esercizio 2 - Soluzione

- Considerando che:
 - Opcode=0x0
 - rs=0xD
 - rt=0xE
 - rd=0xF
 - Shamt=0x0
 - Funct code=0x20
 - Si chiede di indicare l'istruzione corrispondente e rappresentarla in binario e esadecimale.
-
- 000000 01101 01110 01111 00000 100000
 - 0000 0001 1010 1110 0111 1000 0010 0000
add \$15, \$13, \$14
0x 01AE7820

Esercizio 3

- A quale istruzione macchina MIPS corrisponde il codice esadecimale:

0x0232502A

- LINK:
- <https://forms.gle/nQhVJfUEZbbQ1FWs8>

Esercizio 3 - Soluzione

- A quale istruzione macchina MIPS corrisponde il codice esadecimale:

0x0232502A

- 0000 0010 0011 0010 0101 0000 0010 1010
- 000000 10001 10010 01010 00000 101010
- slt \$10, \$17, \$18

Esercizio 4

- **Si chiede di calcolare le somme:**
- **somma1 = a+a+a**
- **somma2= a+a+a+a**
- **Il valore a si trova nel registro \$8. Il risultato e' richiesto nel registro \$9.**

Esercizio 4 - Soluzione

- Si chiede di calcolare le somme:
- somma1 = a+a+a
- somma2= a+a+a+a
- Il valore a si trova nel registro \$8. Il risultato e' richiesto nel registro \$9.

Somma1: add \$9, \$8, \$8
 add \$9, \$9, \$8
Somma2: add \$9, \$8, \$8
 add \$9, \$9, \$9

Link - Esercizi

- **Esercizio 1:**

LINK:

<https://forms.gle/DjPMCKZeqUr8ykmj6>

- **Esercizio 2:**

LINK:

<https://forms.gle/aknSWsFtXrNowmYF6>

- **Esercizio 3:**

LINK:

<https://forms.gle/nQhVJfUEZbbQ1FWs8>

Esercizio EXTRA – per fine lezioni ISA

- Scrivere un programma che comprenda:
 - 1. Un main che:
 - 1. Definisce un array chiamato "array1" costituito da 10 interi; questo array va inizializzato come segue: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 - 2. Chiama una procedura PrintArray che stampa a video "array 1" partendo dall'ultimo elemento fino al primo elemento. Ogni elemento dovrà essere scritto su una riga seguito da ";". La procedura riceve in input l'indirizzo del primo elemento dell'array e la dimensione dell'array.
 - 3. Chiama una procedura BuildArray che costruisce un nuovo array " array2" partendo da un array ricevuto in input.
 - 4. Stampa a video l'array costruito dalla procedura BuildArray " array2".
 - 2. La procedura BuildArray che:
 - 1. Riceve come argomento un array e la sua dimensione.
 - 2. Costruisce un array di lunghezza doppia dell'array ricevuto come argomento. Gli elementi dell'array da costruire si calcolano come segue:
 - - il primo elemento dell'array si ottiene copiando il primo elemento dell'array ricevuto come argomento;
 - - il secondo elemento dell'array si calcola tramite la procedura DoubleIt passando come argomento l'elemento precedente dell'array, cioe' il primo in questo caso;
 - - il terzo elemento dell'array si ottiene copiando il secondo elemento dell'array ricevuto come argomento;
 - - il quarto elemento dell' array si calcola tramite la procedura DoubleIt passando come argomento l'elemento precedente dell'array, cioe' il secondo in questo caso;
 - Secondo array: 1, 2, 2, 4, 3, 6, 4, 8, 5, 10, 6, 12, 7, 14, 8, 16, 9, 18, 10, 20
 - 3. La procedura DoubleIt:
 - Riceve come argomento un numero e ritorna come valore il doppio del numero ricevuto come argomento.

Domanda

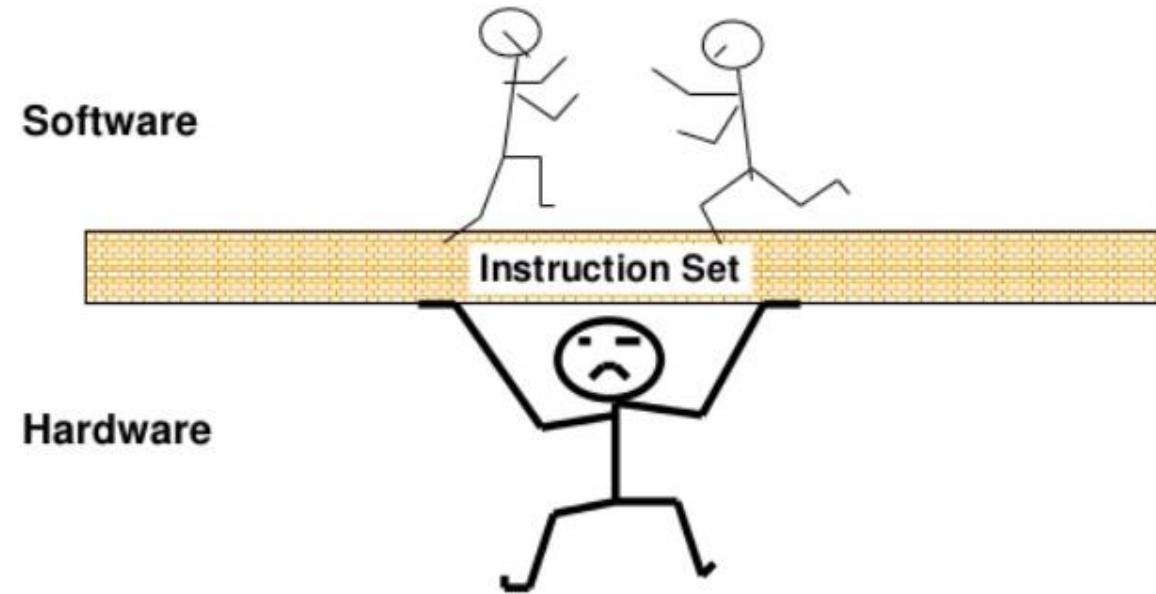
- Qual e' il libro del testo del corso?

**David Patterson, John Hennessy:
Computer Organization and Design,
The Hardware/Software Interface
5th edition, Morgan Kaufmann (Elsevier) /**

"Struttura e progetto dei calcolatori" Zanichelli

1 person clipped this slide

Instruction Set Architecture (ISA)



Source: Computer Architecture: A Quantitative Approach, J. L. Hennessy & D. A. Patterson, 3rd Edition.