



L1 • RIPASSO GENERALE

RAPPRESENTAZIONE NUMERICA >

I numeri di qualsiasi base seguono la forma

$$b_n, b_{n-1}, \dots, b_2, b_1, b_0, b_1, b_2, \dots, b_k$$

ES: BN: 0100.100 → cifre usabili {0,1}

$$\text{OCT: } 12.3 \rightarrow \{0,1,2,3,4,5,6,7\}$$

$$\text{DEC: } 125.5 \rightarrow \{0,1,2,3, \dots, 8,9\}$$

$$\text{HEX: } AB.5C \rightarrow \{0,1,2,3, \dots, 9, A, B, C, D, E, F\}$$

BASE N → DEC >

Sommatoria del valore posizionale moltiplicato per la base elevata alla posizione

$$\text{ES: BN: } 0100.100 = 1 \cdot 2^2 + 1 \cdot 2^1 = 4 + 1 = 4.5$$

$$\text{"OCT: } 12.3 = 1 \cdot 8^1 + 2 \cdot 8^0 + 3 \cdot 8^{-1} = 8 + 2 + \frac{3}{8} = 10.375$$

$$\begin{aligned} \text{"HEX: } 125.5 &= 1 \cdot 16^2 + 2 \cdot 16^1 + 5 \cdot 16^0 + 5 \cdot 16^{-1} \\ &= (2 \cdot 16 + 2) \cdot 16 + 5 + 5 \cdot 16^{-1} \\ &= 288 + \frac{5}{16} = 288.3125 \end{aligned}$$

DEC → BASE N >

Dividere il num per la base segnando i resti. Il numero sarà dal basso verso l'alto, da sx→dx

$$\text{ES: DEC: } 4.5 \text{ BN} = 100.1$$

$$\begin{array}{r} 4 | 2 \\ 2 | 0 \\ 1 | 0 \\ 0 | 1 \end{array} \quad \begin{array}{l} 4 = 100 \\ 0.5 = 2^{-1} = 0.1 \end{array}$$

$$\text{ES: DEC: } 10.375 \text{ = } 12.3$$

$$\begin{array}{r} 10 | 8 \\ 8 | 1 \\ 1 | 2 \\ 0 | 1 \end{array} \quad \begin{array}{l} 8 = 12 \\ 3.75 = 12 \\ 8 = 0 \\ 0 = 0 \end{array}$$

BIN → OCT > OCT → BIN

Si trasformano "in decimale" a gruppi di 3:

In maniera inversa, si "traducono" a gruppi ==

$$\text{ES: } 011010.100 = 32.4$$

$$\begin{aligned} \text{"OCT: } 253.2 &= 010 \quad 5 = 101 \quad 3 = 011 \\ &= 010 \quad 101 \quad 011 \quad 010 \end{aligned}$$

L1 • RIPASSO GENERALE

2

BIN → HEX

"Trascurare" in decimali e grappichi li. Vale anche in maniera inversa

$$\text{ES: "BW: } \underline{\underline{0001}} \underline{\underline{1010}} \underline{\underline{0000}} = \underline{\underline{1A8}} \quad \text{ES: "HEX: ABH" } = 1010 \ 1011. \ 0100$$

$$1D = \begin{smallmatrix} 1 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{smallmatrix} \quad h = 0100$$

$$11 = \begin{smallmatrix} 1 & 0 & 1 & 1 \end{smallmatrix}$$

BIN: $101 \circ 2^m$ > SHIFT ARITHMETICO

Comportamento analogo come DEC quando si moltiplica per 10

N.B.: Quando si è in complemento a 2 aggiungere il al posto di O.

$$ES: "DEC: \tau \cdot 8 \text{ IN BIN?}"$$

$$8 = 2^3 \text{ e } \tau = \frac{0111}{\begin{smallmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \end{smallmatrix}} \text{ Sposto o DX} \quad 01110000 = 32 + 16 + 8 = 56$$

ES: "DEC : -7.8 IN B'N?
 $8 = 2^3$ e $\exists = 0$ eel sposto a dx 100100
 in $C_2 = 1001$ Bi-converte 011100 = $32 + 16 + 8 = 56$.

BW: DIV PER 2^m > SHIFT ARITMETICO

Stesso comportamento in DEC di quando si divide per 10

$$\begin{array}{ll} \text{ES: "DEC: } \mp 1/4 \text{ W BIN?"} & \text{"DEC: } -\mp 1/4 \text{ IN BIN?"} \\ \underline{4=2^2} \quad \mp = 0.111 \Rightarrow 0.111 & \mp = 0.111 \Rightarrow 30.01 \\ & \downarrow \\ & -\mp = 100.1 \quad \downarrow \\ & 01.11 \end{array}$$

COMPLEMENTO A 2 ➤ RANGE DI RAPPRESENTAZIONE

Dato un C_2 a n bit segue la formula $\sum_{i=0}^{n-1} 2^{n-i-1}$ \leftarrow inclusi

ES: "Con 3 bit" $[2^{-2}, 2^2 - 1]$ • "Con 5 bit" $[2^{-4}, 2^4 - 1]$
 $[2^{-4}, 3]$ $[-16, 15]$

4 • RIPASSO GENERALE

COMPLEMENTO A 2 ➤ OVERFLOW

Lo è se 1 delle 2 condizioni è soddisfatta

- I numeri hanno stesso segno e il risultato ha segno opposto
- Le coppie di bit di riporto (n_1, n) e $(n-1, n-2)$ sono diverse.

ES: "10-6 in C_2 a 8 bit"

$$\begin{array}{r} \text{1} \cancel{\text{1}} \cancel{\text{1}} \\ \text{1} \text{0} \text{0} \text{0} \text{0} \text{1} \text{0} \text{0} \\ + \text{1} \text{0} \text{1} \text{0} \text{1} \text{0} \text{0} \\ \hline \text{0} \text{0} \text{0} \text{0} \text{1} \text{0} \text{0} \end{array}$$

\checkmark Scopro le ee e 10 10 = No overflow

$= 4$

ES: "-86-88 in C_2 a 8 bit"

$$\begin{array}{r} \text{1} \cancel{\text{0}} \cancel{\text{1}} \cancel{\text{1}} \\ \text{1} \text{0} \text{1} \text{0} \text{1} \text{0} \text{0} \text{0} \\ + \text{1} \text{0} \text{1} \text{1} \text{1} \text{0} \text{1} \text{1} \\ \hline \text{0} \text{1} \text{1} \text{0} \text{0} \text{1} \text{0} \text{1} \end{array}$$

$\cancel{=}$ Overflow!

CODIFICA: MODULO + SEGNO ➤

Consiste nel considerare il 1° bit come segno e il resto dei bit come valore assoluto del numero.

ES: "10001" → -1

\downarrow • 0 = Pos, 1 = Neg

CODIFICA: COMPLEMENTO A 1 ➤

Consiste nell'invertire tutti i bit di un numero.
(000 = +0).

NR: Se nelle somme è presente un carry in $n+1$ allora va aggiunto al risultato.

ES "5 ⇒ 0101"

$C_1 \Rightarrow 0101 (+5) (-5) \Rightarrow 1010$

$\begin{array}{r} \text{1} \text{0} \text{1} \text{0} \text{1} + (+5) \\ \text{1} \text{1} \text{0} \text{0} \\ \hline \text{0} \text{0} \text{0} \text{1} + \\ \text{1} \\ \hline \text{0} \text{0} \text{1} \text{0} \end{array}$

L2 • VON NEUMANN

ARCHITETTURA DI VON NEUMANN ➤ (DEF)

Nel '50 durante la seconda guerra mondiale, consentì a differenza delle altre architetture.

ARCH. VON NEUMANN ➤ STRUTTURA: CPU

CPU sta per Central Processing Unit.

Si occupa di eseguire delle operazioni. È formata da:

- **CONTROL UNIT**

- Riconosce l'istruzione da eseguire.
- Coordinia le altre componenti della CPU per eseguire l'istruzione.
- "Si prepara" per l'istruzione successiva.
È importante notare che alcune istruzioni le esegue direttamente la control unit

- **ALU: Arithmetic Logic Unit**

Parte della CPU che si occupa di eseguire operazioni logico-aritmetiche

- somme / sottrazioni
- LESS
- AND, OR, NOT...
- overflow

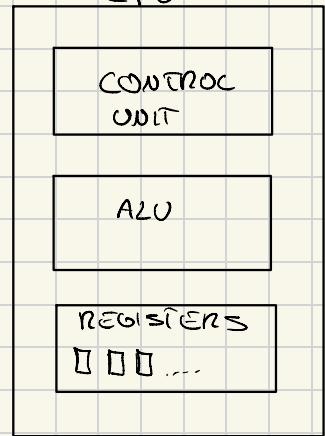
- **REGISTERS**

Sono la memoria della CPU. Il numero, tipologia e dimensioni variano dell'architettura della CPU.

Effettivamente vengono usati da Control Unit e ALU come "memoria di comodo" ad alta velocità al posto di usare la memoria principale. Registri importanti:

- **PC: Program Counter** => Contiene l'indirizzo della prossima istruzione da eseguire

- **IR: Instruction Register** => Contiene il contenuto (solo ISTR) della prossima istruzione da eseguire



ANAL. VON NEUMANN ➤ STRUTURA: MEM. PRIMARIA
 È una memoria volatile contenente dati ed istruzioni meno performante dei registri.

ANAL. VON NEUMANN ➤ STRUTURA: BUS DI SISTEMA
 Sistema di comunicazione ad alta velocità nell'architettura

DATA PATH ➤

È un workflow di esecuzione che descrive il processo di lavorazione delle CPU dall'input all'output.
Varie a seconda dell'Instruction Set Architecture

In RISC-V:

- I dati di **input** da lavorare si trovano in **registri**
- I **output** lavorati

FLUSSO: FETCH-DECODE-EXECUTE ➤

L'algoritmo di esecuzione delle istruzioni è:

1. (FETCH): Si prende la prossima istruzione da eseguire dalla memoria primaria.

La si salva nel registro Instruction Registry

2. (FETCH): Si salva l'indirizzo dell'istruzione nel registro di Program Counter

3. (DECODE): Si determina il tipo di istruzione da eseguire.

4. (DECODE): SE l'istruzione contiene una word, allora se serve viene salvata nei registri.

5. (EXECUTE): Si esegue l'istruzione. Si torna al fetching dell'istr. successiva.

CPU > PRESTAZIONI

Si misurano con:

- TEMPO DI ESECUZIONE / RISPOSTA (PROGRAMMA)

Il tempo tra l'inizio e la fine di un programma.

- THROUGHPUT / LARGHEZZA DI BANDA

Il numero di tasks risolti nell'unità di tempo.

- TEMPO DI ESECUZIONE DELLA CPU

Tempo effettivo impiegato dalla CPU per eseguire un certo Task. Divisibile in

- TEMPO DI CPU UTENTE

Tempo impiegato ad eseguire istruzioni interne

- TEMPO DI CPU DI SISTEMA

Tempo impiegato ad eseguire istruzioni che coinvolgono il sistema operativo. (es: file, stampanti ecc.)

- PERIODO / FREQUENZA DI CLOCK

Numero di cicli di clock nell'unità di tempo.

ES: 5 Ghz con 1 periodo di clock = ∞ nanosec

CPU > MISURARE LE PRESTAZIONI

Si misurano con formule differenti:

$$\text{Tempo di CPU} = \frac{\text{cicli}}{\text{nº di clock impiegati per il programma}} * \text{durata di clock}$$

di un programma

oppure

$$\text{Tempo di CPU} = \frac{\text{cicli}}{\text{nº di clock impiegati per il programma}} / \text{frequenza di clock}$$

di un programma

$$\text{Tempo di CPU} = \frac{\text{nº di istruzioni} * \text{CPI}}{\text{frequenza di clock}}$$

CPI = Il "peso" in
numeri di clock di
una istruzione ISA.

CPU > MISURARE LE PRESTAZIONI

- ISTRUZIONI AL SECONDO

Si usa la formula:

$$\text{Istr./s} = \frac{\text{frequenza Hz}}{\text{CPI}}$$

ES: Calcolare il numero di istruzioni al secondo
di P1 con freq. a 3 GHz e CPI 1.5

$$P_1 = \frac{3}{1.5} = 2$$

RISC-V > (INTRO)

Tipologia di architettura che sta per Reduced Instruction Set Computer.

È un tipo di architettura open source e senza fees.

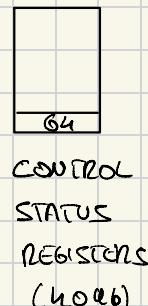
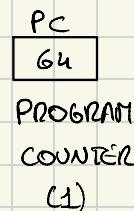
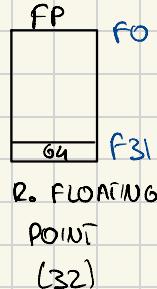
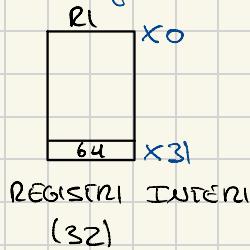
[NEL CORSO]

Nel corso si vedranno le

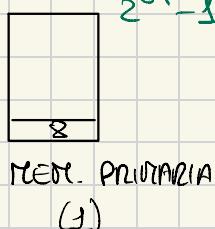
- RISC-V I : Principalmente, con gli istruzioni $I = \text{Integer}$
- RISC-V M : Supporto a moltiplicazioni / divisioni $M = \text{Multiply}$
- RISC-V F : $\leq \quad \leq$ num virgo (o mobile) $F = \text{Floating point}$
Gbit

RISC-V > TIPOLOGIA DEI REGISTRI (RISC-V I)

I registri sono divisibili in



(accesso solo in Kernel mode)



WORD = 32 bit

DOUBLE WORD = 64 bit

REGISTRI > REGISTRI INTERI

- Registri per gli interi
 - Quantità: 32, indicati con x0 .. x31
 - Dimensione: 64 bit
 - ...

x0	zero
x1	Return address (ra)
x2	Stack pointer (sp)
x3	Global pointer (gp)
x4	Thread pointer (tp)
x8	Frame pointer (fp)
x10-x17	Registri usati per il passaggio di parametri nelle procedure e valori di ritorno
x5-x7, x28-x31	Registri temporanei, non salvati in caso di chiamata
x8-x9, x18-x27	Registri da salvare: il contenuto va preservato se utilizzati dalla procedura chiamata

ND: x8 appare 2 volte perché non è detto che venga usato.

ISTRUZIONE DI ADD ➤

SYNTAX ➤ add a, b, c [ASSEMBLY] $\equiv a = b + c$ [C]

Somma i valori dei registri e li salva su un altro registro.

$$\text{ES: add } xs, x1, x2 \Rightarrow xs = x1 + x2$$

$$\text{add } xs, x0, x3 \Rightarrow xs = 0 + x3 \Rightarrow xs = x3$$

ISTRUZIONE DI SUB ➤

SYNTAX ➤ sub a, b, c [ASSEMBLY] $\equiv a = b - c$ [C]

Sottrae i valori dei registri e li salva su un altro registro.

$$\text{ES: sub } xs, x1, x2 \Rightarrow xs = x1 - x2$$

$$\text{sub } xs, x0, xs \Rightarrow xs = 0 - xs \Rightarrow xs = -xs$$

UNITÀ DI MISURA ➤

Essendo il bit, l'unità più piccola di informazione in informatica, esso può essere raggruppato in Bytes, ovvero 8 bit.

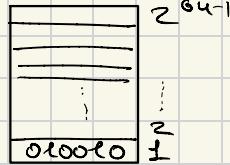
Sono anche in uso i seguenti prefissi per multipli binari:

Nome	Fattore	Equivalenti	F. Equivalenti
(Ki) Kibi	2^{10}	Kilo	10^3
(Mi) Mebi	2^{20}	Mega	10^6
(Gi) Gibi	2^{30}	Giga	10^9
(Ti) Tebi	2^{40}	Tera	10^{12}
(Pi) Petabi	2^{50}	Peta	10^{15}
(Ei) Exabi	2^{60}	Exa	10^{18}
(Zi) Zettabi	2^{70}	Zetta	10^{21}
(Yi) Yottabi	2^{80}	Yotta	10^{24}

NEUTRINO > MEMORIA

Quando si esegue un programma, si caricano dati ed istruzioni in memoria.

La memoria può essere astratta come un vettore a 1 dimensione con i dati:



MEMORIA > ACCESSO DATI

Per eseguire delle operazioni, i dati possono essere caricati nei registri.

Sarà il compilatore a decidere se generare istruzioni assembly con dei dati in memoria o meno.

È importante sapere che:

- Generalmente il compilatore inserisce i dati più utilizzati. Anche se l'utente nel codice può "suggerire".
- L'ALU non può interagire con la memoria

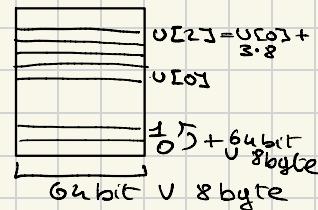
ISTRUZIONI > LOAD "DOUBLE WORD"

SYNTAX > $ld \quad a, b$ $\rightarrow a = \text{reg. dest}, b = \text{ind. dato memoria}$
Carica un dato dalla memoria principale tramite l'indirizzo di memoria del secondo parametro nel primo.

NRB: Nei vettori, per avere l'i-esimo el. bisogna aggiungere un offset.

In byte basta sommare e sottrarre

8



ES: "Caricare in un registro il 3° elemento di un array con base U[0] in x20 con indici crescenti"

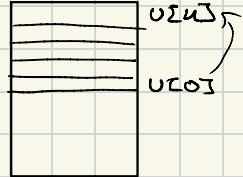
- (1) Definisco se sommare o togliere \rightarrow crescente = sommare + 3 * 8
- (2) Definisco l'istruzione $ld \quad x10 \quad x20(x20)$

ISTRUZIONI > STORE

SYNTAX > **sd src offset(dest)**

Salvo i dati dal registro src nella memoria principale usando l'indirizzo specificato su dest.

ES "Salvare i dati dal registro x10 nel 4° elemento dell'array in memoria con ind. base x20"

sd x10 32(x20)

LOAD > OTTIMIZZAZIONE

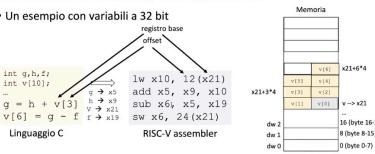
Per le load, si può scegliere se usare o meno l'estensione del segno con i comandi che variano con u. Ci sono anche le varianti:

		byte	bit
-	l b / lbu	byte	1
-	l h / lhu	half-word	2
-	l w / lwu	word	4
-	l d / ldu	d word	8

NB: Bisogna stare attenti sui moltiplicatori perché per dati inferiori a una double word, potrebbero essere distribuiti nella stessa cella

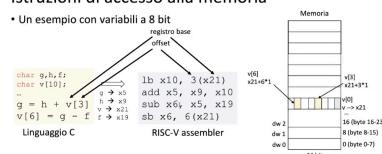
Istruzioni di accesso alla memoria

- Un esempio con variabili a 32 bit



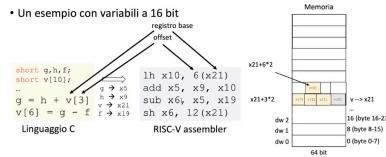
Istruzioni di accesso alla memoria

- Un esempio con variabili a 8 bit



Istruzioni di accesso alla memoria

- Un esempio con variabili a 16 bit



ISTRUZIONI > ADD IMMEDIATE

SYNTAX > addi dest, addi const ← costante

Si usa quando si vuole sommare una costante non troppo grande evitando una load.

La costante ha range [-2048, 2047] in complemento a 2.

ES: "Definisci nel registro x23 il valore 12"

addi x23 x0, 12

↳ vanno sempre come 3° param.

LINGUAGGIO MACCHINA >

Il linguaggio assembly viene tradotto in linguaggio macchina.
Le istruzioni possono essere:

• ISTRUZIONI REGISTRO (32)

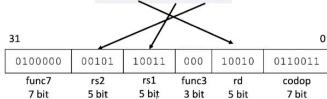
Sono formate da 32 bit, descritte ($Dx \rightarrow s2c$)

- cod op (7) => Contiene la tipologia di istruzione
- rd (5) => Contiene l'indirizzo del registro di destinazione.
- funz 3 (3) => viene usata in congiunzione a cod op per rappresentare l'istruzione.
- rs2 (5) => Primo registro di source
- rs2 (5) => Secondo == =
- funz 7 (7) => Come funz 3

Formato di tipo R (registro)

- Esempio

sub x18, x19, x5



- Per specificare uno dei 32 registri sono necessari 5 bit
- codop + func7 + func3 indicano l'istruzione rappresentata

LINGUAGGI MACCHINA ➤

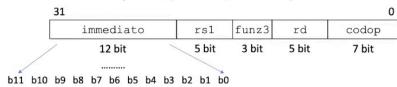
• ISTRUZIONI IMMEDIATE (32)

Dichiarano istruzioni quali load, addi ($DX \rightarrow SX$)

- cod op (7): Il cod op.
- rd (5): Reg. di dest
- funz3 (3):
- rs1 (5):
- immediate (12): Contiene costanti nel range

$$[-2^{12}, +2^{12}-1]$$

Formato di tipo I (immediato)



- Permette di codificare le istruzioni che richiedono il caricamento dalla memoria o una costante, come load, addi, andi, ori, ...
- Sono presenti 12 bit perché con 5 bit l'intervallo di rappresentazione per costanti e (soprattutto) offset sarebbe stato troppo ridotto
- Il campo immediato
 - Rappresentato in complemento a due
 - Valori possibili: da -2048 a +2047

LINGUAGGI MACCREINA ➔

• ISTRUZIONE "S" - SCORE

Contiene solo l'istruzione di store in quanto per regioni progettuali l'ovvero di doppio indirizzo "di source" è più facile creare un tipo a parte di istruzioni.

È formato da 32 bit:

- **Codop** (7) : Identifica il tipo di operazione.
 - **immediato_1** (5) : Contiene la parte dell'offset meno significativa.
 - **funz_3** (3) : Contiene bit di supporto a codop.
 - **rs1** (5) : Contiene l'indirizzo del registro di source1 "src".
 - **rs2** (5) : Contiene l'indirizzo del registro di source2 "dest".
 - **funz_7** (7) : Altri bit di supporto a codop.

ISPRACONI > SHIFT LOGIC

Syntax: SIRIUS es sub, sub, sub, sub y se xsl xss

- In [I] decide la direzione, a destra o / sinistra |
 - i (opt) consente di uscire una costante al posto del 3° param.

Effettua lo shift logico (aggiunge 0) a destra (sinistra) nel 1° parametro partendo dal 2° parametro per U elementi specificato nel 3° parametro.

$$\begin{array}{l} \text{ES: } \downarrow S11 \quad xq \quad xs \quad x3 \\ xq = \underbrace{xs + "0..0"}_{x3} \end{array} \quad \begin{array}{l} srl \quad xq \quad xs \quad x3 \\ xq = xs + \underbrace{"0..0"}_{x3} \end{array}$$

$x_d = "00000" + x_S$

ISTRUZIONI > SHIFT ARITMETICO

SYNTAX > $sra / srai \geq dest, x_1 x_2/c$

Effettua lo shift aritmetico di un num. ovvero:

- Se è positivo, aggiunge 0.
- Se è negativo, aggiunge 1

ES: $sra \quad x_a \quad x_{10} \quad x_{11}$
 $srai \quad x_a \quad x_{10} \quad S$ $sra \quad x_a \quad x_{10} \quad x_{11}$
 $srai \quad x_a \quad x_{10} \quad S$

ISTRUZIONI > LOGICA: AND

SYNTAX > $and \quad x_{res}, x_a, x_b \quad \vee andi \quad x_{res}, x_a, c$

Effettua una AND logica dai due registri source e salva il risultato nel registro risultato.

ES: $and \quad x_7, x_1, x_2$

Può essere usata con le maschere in CLECK

ISTRUZIONI > LOGICA: OR

SYNTAX > $or \quad x_{res}, x_a, x_b \quad \vee ori \quad x_{res}, x_a, c$ Effettua una OR logica fra x_a, x_b e setta in x_{res} .

NB! Può essere usata con maschere in SET.

Operazioni logiche		
• L'istruzione and permette di selezionare alcuni bit del primo operando indicandoli all'interno di una maschera (secondo operando)		
• Esempio (su 32 bit, per egesigenze di spazio)		
and x5, x6, x7		Selezione
		Risultato
		Maschera
		Risultato

ISTRUZIONI > LOGICA: XOR

SYNTAX > $xor \quad x_{res}, x_a, x_b \quad \vee xor \quad x_{res}, x_a, x_b$

Effettua una XOR

ES $xor \quad x_1, x_2, x_3 \quad xor \quad x_1, x_2, 3$

ISTRUZIONI > LOGICA: NOT

SYNTAX > $not \quad x_a, x_b \quad x_a = \overline{x_b}$

Nega il secondo e salva sul primo

ES: $xor \quad x_1, x_2$

LS •

ISTRUZIONI > BRANCHE (DEF)

Consentono di codificare le variazioni del flusso del programma. Sono più istruzioni:

- **breq / bne** → Controllano equivalenza
- **bgt / blt** → = grandezza

Seguono tutti lo stesso: [breq/bgt..] x_1, x_2, L
 ovvero, se la condizione viene soddisfatta, allora il program counter viene modificato e si salta verso l'etichetta specificata (offset relativo)

ISTRUZIONI > BRANCH IF-EQUALS

SYNTAX > **breq[U] x_1, x_2 ET**

- x_1, x_2 → Registri da confrontare se sono uguali
- ET → Etichette dove fare il salto se True
- U → (opzionale) fa il confronto senza contare segni

ES: "breq $x_1, x_2 L1$ " "breq $x_1, x_2 L2$ "

Se $x_1 == x_2$ salta a

L1

Se $x_1 == x_2$ (No segno) salta

a L2

ISTRUZIONI > BRANCH IF NOT EQUALS

SYNTAX > **bne[U] x_1, x_2 ET**

Stesso discorso riguardo a equals

ISTRUZIONI > BRANCH IF GREATER THAN

SYNTAX > **bgt[E|e][U] x_1, x_2 ET**

- E → Indica che fa il confronto escludendo $x_1 == x_2$
- e → ~~E~~ includendo $x_1 == x_2$
- U → unsigned

ESEMPIO DI IMPLEMENTAZIONE ➤

Salvi condizionati: ciclo while

```

long v[10], k, i;
while (v[i]==k)
{
    i = i+1;
}

```

LOOP: slli x10,x22,3 ← Salva in x10 l'indirizzo della doubleword v[i]
add x10,x10,x25
ld x9,0(x10) ← Carica dalla memoria il valore contenuto nella doubleword v[i]
bne x9,x24,ENDLOOP ← Se v[i] != k, allora il ciclo termina
...
addi x22,x22,1 ← Incrementa i e torna alla valutazione della condizione del ciclo while
beq x0,x0,LOOP

ENDLOOP:

RISC-V Instruction Set

61

LINGUAGGI MACCHINA ➤

• ISTRUZIONI "SR"

Sono istruzioni che contengono le istruzioni di salto: bit

- | | | |
|-------------|-----|-------------------------------|
| - cod op | (7) | - Codice operazione |
| - imm[11] | (3) | - 11° bit immediato |
| - == [3:4] | (4) | - Dal 3° al 4° bit immediato |
| - FUnz3 | (3) | - Supporto a codop |
| - RS1 | (5) | - 1° registro di source |
| - RS2 | (5) | - 2° == == |
| - imm[5:10] | (5) | - dal 5° al 10° bit immediato |
| - imm[12] | (5) | - 12° == == |

NB: L'immediato occupa 13 bit con l'ultimo bit sempre a 0 [-2048, 4095] e rappresenta solo numeri pari

Questo perché le istruzioni occupano 32 bit (con alcune da 36) quindi non possono uscire dispari.

LS •

ISTRUZIONI > MOLTIPLICAZIONI

SYNTAX > mul [h] > dest, x1, x2

Moltiplica usando un registro interno da 128 bit
 per avere tutto il risultato si aggiunge h

ISTRUZIONI > DIVISIONI

SYNTAX > div > dest x1, x2

Effettua la divisione intera e riporta il risultato

ISTRUZIONI > DIVISIONI (SOLO RESTO)

SYNTAX > rem > dest x1,x2

Riporta il resto della divisione

PROCEDURE > (INTRODUZIONE)

Le procedure sono blocchi di codice riutilizzabili ecc.

Nell'architettura si distinguono in:

- chiamante : Procedura che chiama
- chiamata : Procedura che viene chiamata

Il flusso sarebbe :

1. La chiamante chiama

1.a (Se ci sono) si passano i dati

1.b Si salva l'istruzione successiva all'invocazione

1.c Si trasferisce il controllo

2. La chiamata viene chiamata

2.a (Se ci sono) prende i dati

2.b fa quello che deve fare

2.c (Se c'è) salva il risultato

2.d Restituisce il controllo

LS •

4

ISTRUZIONI > JAL

SYNTAX > jal >dest ET oppure jal ET

Effettua il salto all'etichetta ET ed aggiorna
in >dest l'indirizzo dell'istruzione successiva dove
proseguire.

NOTA MOLTO BENE!

USARE SEMPRE la pseudo istruzione SENZA >dest, così
va di default a x1 (ra). Per convenzione

SPECIFICARE ALTRI REGISTRI = NON PASSARE ARQUITETTURALE

LINGUAGGI MACCHINA >

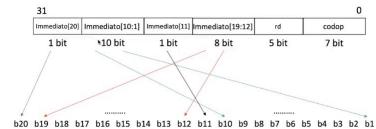
• ISTRUZIONI "J"

Dedicate a jal

- codop 7
- rd 5
- imm [12:0] 8
- imm [2:1] 1
- imm [1:10] 10
- imm [20] 1

JAL e linguaggio macchina

- Viene introdotto un nuovo tipo: J



ISTRUZIONI > JALR / JR

Syntax > jalr rd offset(rs)

fa un salto verso rd dove rs.

Anche qui si usa sempre

jr ra ovvero ritorna all'
istruzione dentro ra=x1

Modifica del flusso di programma:
ritorno al chiamante

Jump-and-link register

jalr rd, offset(rs1)

• Salta ad un indirizzo qualsiasi

• $x[rd] = PC + 4; PC = x[rs1] + sext(offset) \& -1$

Segno esteso a 64bit

Bit 0 azzero

• La procedura chiamata come ultima istruzione esegue

