


Architettura degli elaboratori – II
Turno 1 (cognomi A-G)
Introduzione

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento sul Patterson 6th edition: capitolo 4.1-4.4.


Sommario

- Introduzione
- Administratives
- La CPU a ciclo singolo

Obbiettivo di un'architettura

Elabora in modo adeguato un input per produrre l'output.

- Le unità di *ingresso* (tastiera, mouse, **rete, interfacce con dispositivi di acquisizione**, ecc.) permettono all'elaboratore di acquisire informazioni e richieste dall'ambiente esterno (non solo controllo diretto dall'utente).
- L'architettura di elaborazione.

- Le unità di *uscita* (terminale grafico, stampanti, rete, ecc.) consentono al calcolatore di comunicare i risultati ottenuti dall'elaborazione all'ambiente esterno.

Input ==> Elaborazione ==> Output

Cosa fa un elaboratore?

- Algoritmi (sequenza di operazioni elementari –> ***istruzioni elementari***).
  - Calcoli (calcolatore).
  - **Operazioni logiche per controllo del flusso dell'elaborazione**
  - Trasferimento con la memoria
  - I/O

⇓

- **Programma** (Ada (Byron) Lovelace, 1830) = *Algoritmi in Software (calcolo della serie di Bernulli, linguaggio «Ada»)*

Come lo fa? *Hardware: calcoli e operazioni logiche vengono implementare da un' **architetture di elaborazione***. Circuiti costituiti da ***porte logiche***. Sono architetture ***sequenziali*** (una istruzione alla volta)

Terza rivoluzione della nostra civiltà: la rivoluzione agricola, la rivoluzione industriale e la rivoluzione dell'informatica (e delle biotecnologie).

Architetture LOAD/STORE

- **CPU + registri + memoria locale (dentro la CPU - cache)**

load Ra Rb	Ra ← mem[Rb]	lw $r2, cost($r1)
add Rd Ra Rc	Rd ← Ra + Rc	add $r4,$r2,$r3
store Rd Rb	mem[Rb] ← Rd	sw $r4, cost($r1)

Architetture LOAD/STORE: gli operandi dell'ALU possono provenire **soltanto** dai registri ad uso generale contenuti nella CPU e **non** possono provenire dalla memoria principale. Sono necessarie apposite istruzioni di:

- *caricamento (LOAD)* dei dati da memoria ai registri;
- *memorizzazione (STORE)* dei dati dai registri alla memoria.

- **Memoria**

Uno degli obiettivi del corso è capire come sia stato reso efficiente il sistema di memoria.

Famiglie di architetture

- CISC (Complex Instruction Set Computer)
- RISC (Reduced Instruction Set Computer)

"Reduced" è un elemento di forza! (cf. rasoio di Occam: "the simplest the best" – "Entities should not be multiplied without necessity" - legge della parsomonia)

CPU di tipo CISC

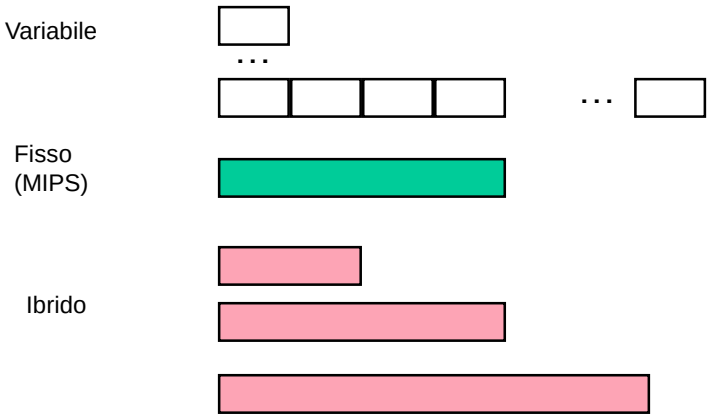
- Caratterizzate da elevata complessità delle istruzioni eseguibili ed elevato numero di istruzioni che costituiscono l'insieme delle istruzioni.
- Numerose modalità di indirizzamento per gli **operandi** dell'*ALU* che possono provenire da registri oppure da memoria, nel qual caso l'indirizzamento può essere diretto, indiretto, con registro base, ecc.
- Dimensione *variabile* delle istruzioni a seconda della modalità di indirizzamento di ogni operando ⇒ complessità di gestione della fase di prelievo o *fetch* in quanto a priori non è nota la lunghezza dell'istruzione da caricare.
- Elevata complessità della *CPU* stessa (cioè dell'hardware relativo) in termini degli elementi che la compongono con la conseguenza di rallentare i tempi di esecuzione delle operazioni. Elevata profondità dell'albero delle porte logiche, utilizzato per la decodifica.

Utilizzo architettura Intel 80x86: le 10 istruzioni più frequenti

° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

° **Simple instructions dominate instruction frequency** => RISC

I diversi formati di istruzioni



Il formato fisso consente di massimizzare la velocità, il formato ibrido consente di minimizzare la lunghezza del codice.

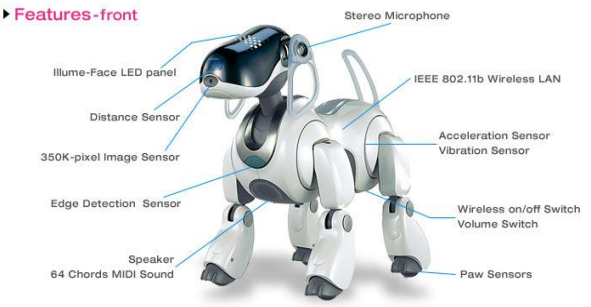
Architetture di tipo RISC

- Ispirate al principio di eseguire soltanto istruzioni semplici: le operazioni complesse vengono scomposte in una serie di istruzioni più semplici da eseguire in un ciclo base ridotto, con l'obiettivo di migliorare le prestazioni ottenibili dalle *CPU CISC*.
- Caratterizzate da istruzioni molto semplificate.
- Gli operandi dell'*ALU* possono provenire dai registri ma *non* dalla memoria. Per il trasferimento dei dati da memoria ai registri e viceversa si utilizzano delle apposite operazioni di caricamento (*load*) e di memorizzazione (*store)*
  ⇒ ***architetture load/store.***
- *CPU* relativamente semplice ⇒ si riducono i tempi di esecuzione delle singole istruzioni, che sono però meno potenti delle istruzioni *CISC*.
- Dimensione *fissa* delle istruzioni ⇒ più semplice la gestione della fase di prelievo *(fetch)* e della codifica delle istruzioni da eseguire
- **Architetture RISC sono architetture load/store native**

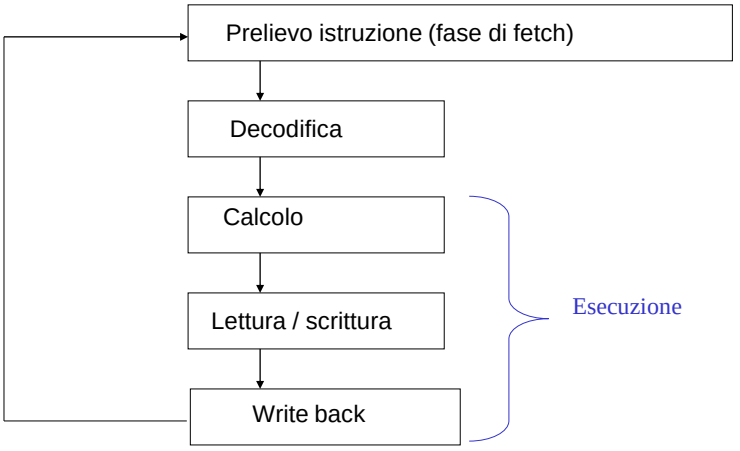
Architettura MIPS

- Architettura MIPS appartiene alla famiglia delle architetture **RISC (Reduced Instruction Set Computer)** sviluppate dal 1980 in poi
  - Esempi: Sun Sparc, HP PA-RISC, IBM Power PC, DEC Alpha, Silicon Graphics, AIBO-Sony, ARM (e.g. Exynos 2100: tre core Cortex-A78 ad alte prestazioni e quattro Cortex-A55 a basso consumo energetico: Samsung S21), **Risc V**.

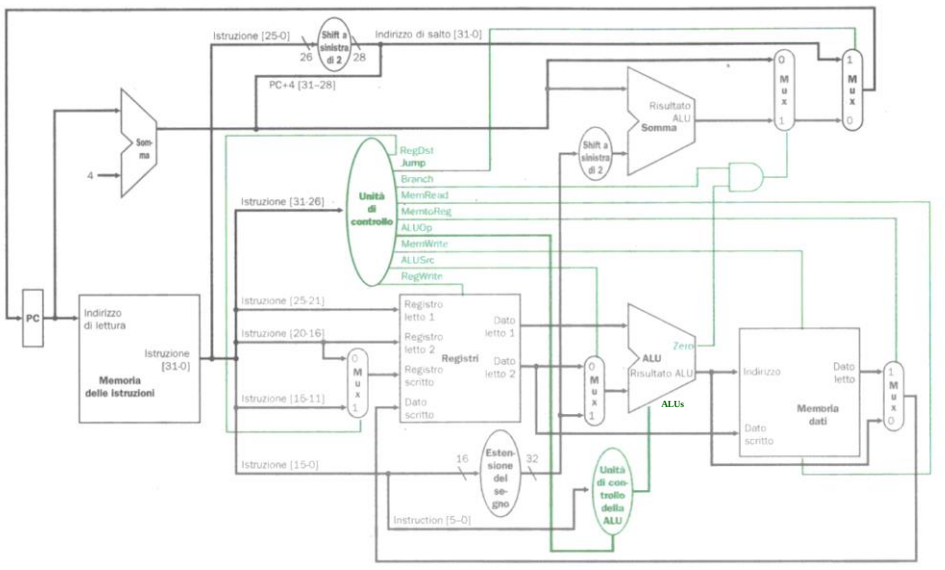
- Principali obiettivi delle architetture RISC:
  - Semplificare la progettazione dell'hardware e del compilatore
  - Massimizzare le prestazioni
  - Minimizzare i costi



Ciclo di esecuzione di un'istruzione



MIPS a singolo ciclo



Sommario

- Introduzione
- Administratives
- La CPU a ciclo singolo

Architetture II (6cfu)

Turno 1 - Cognomi A-G (turno 2 – Cognomi H-Z prof. Nicola Basilico)

Docente: Prof. N. Alberto Borghese: alberto.borghese@unimi.it
Laboratorio Assembler (configurazione attuale):
Prof. Marco Tarini (Turno 1)
Dott. Massimo Rivolta (Turno 2)

Orario e aule per la parte di teoria (se riaprirà la didattica in presenza):

Lunedì	Ore 10.30-12.30	Aula 202, via Celoria 20
Mercoledì	Ore 08.30-10.30	Aula 100, via Celoria 20

Orario di ricevimento: su appuntamento.

Materiale sensibile delle lezioni caricato su Ariel.

Strumento principale di contatto: email!

Programma

Sito principale:
http://borghese.di.unimi.it/Teaching/Architettura_II/_Arch_II.html

Programma (syllabus + slide):
http://borghese.di.unimi.it/Teaching/Architettura_II/Programma_2021-2022.html

Argomenti principali:
CPU (avanzate)
Gerarchie di memoria e memoria virtuale
Interconnessioni

Esame

Parte teorica (2/3 del voto).

- Prova scritta + orale. Appelli ogni 1 / 2 / 3 mesi, al di fuori dal periodo delle lezioni.
- L'orale sarà in presenza e inizierà con la correzione della prova scritta.
- I compitini sostituiranno integralmente l'esame -> l'orale è facoltativo.
- Per sostenere l'esame occorre iscriversi sul SIFA all'**appello scritto**. Non è permesso portare il telefonino nell'aula dello scritto.

Parte di laboratorio (1/3 del voto).

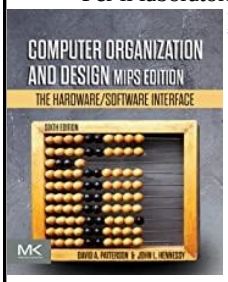
Materiale didattico

http://borghese.di.unimi.it/Teaching/Architettura_II/References.rtf

Testo di base (è disponibile sia in inglese che in italiano):
"Computer Organization & Design: The Hardware/Software Interface (**MIPS edition**)", D.A. Patterson and J.L. Hennessy, Morgan Kaufmann Publishers, **Sixth Edition**, 2018. Sono disponibili anche una versione RISC V e una versione ARM, che **non** sono state adottate. La sesta edizione presenta cambiamenti minimi rispetto alla quinta edizione.

Struttura e progetto dei calcolatori: l'interfaccia hardware-software, D.A. Patterson and J.L. Hennessy, Quinta edizione, Zanichelli, 2022 (Nota: la quinta edizione Zanichelli è la traduzione della sesta edizione inglese).

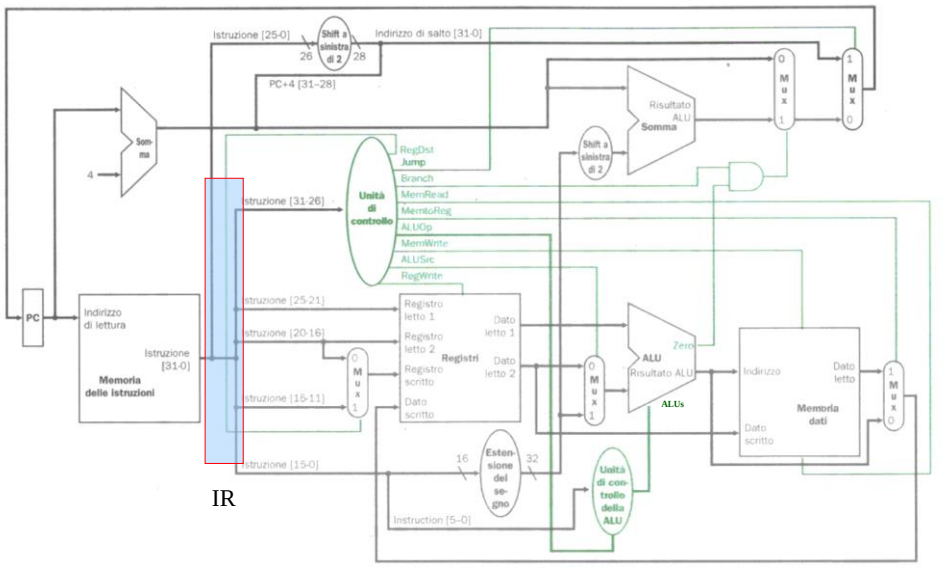
Per il laboratorio viene utilizzato il SW MARS di emulazione dell'assembler MIPS: https://courses.missouristate.edu/KenVollmar/MARS/



Sommario

- Introduzione
- Administratives
- La CPU a ciclo singolo

MIPS a singolo ciclo (di clock)



IR

I componenti di un'architettura

CPU

- Banco di registri *(Register File)* ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale.
- Registro *Program counter (PC).* Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzione;
- Registro *Instruction Register (IR)*. Contiene l'istruzione in corso di esecuzione. Questo registro verrà utilizzato più avanti nelle architetture multi-ciclo.
- Unità per l'esecuzione delle operazioni aritmetico-logiche (*Arithmetic Logic Unit - ALU*). I dati forniti all'*ALU* possono provenire da registri oppure direttamente dalla memoria, a seconda delle modalità di indirizzamento previste;
- Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile *(Floating Point Unit – FPU)*, sommatori ausiliari, ecc.;
- ***Unità di controllo***. Controlla il flusso e determina le operazioni di ciascun blocco.

MEMORIA PRINCIPALE

MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Tipi di istruzioni

- Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:
  - ***Istruzioni aritmetico-logiche;***
  - ***Istruzioni di trasferimento da/verso la memoria (load/store);***
  - ***Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;***
  - Istruzioni di trasferimento in ingresso/uscita (I/O).

Definizione di un'ISA (Instruction Set Architecture)

Definizione del funzionamento: insieme delle istruzioni (interfaccia verso i linguaggi ad alto livello).

- Tipologia di istruzioni.
- Meccanismo di funzionamento.

Definizione del formato: codifica delle istruzioni (interfaccia verso l'HW).

- Formato delle istruzioni.
- Suddivisione in gruppi omogenei dei bit che costituiscono l'istruzione.

Alcune domande:
- Come e dove si specifica il tipo di istruzione?
- Come e dove si specifica da dove vengono letti i dati?
- Come e dove si specifica dove si scrivono i dati prodotti?
- Come viene gestita la memoria in lettura e scrittura?
- Come vengono gestiti i salti?

Codifica (modulare) delle istruzioni *architettura*

- Tutte le istruzioni MIPS hanno la **stessa** dimensione **(32 bit) – Architettura RISC.**
- Le istruzioni vengono suddivise in gruppi di bit funzionali (campi).
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit **(6 bit)** più significativi **(codice operativo - *OPCODE*)**
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register) – Lavorano su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate) – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - **Tipo J (jump) – Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

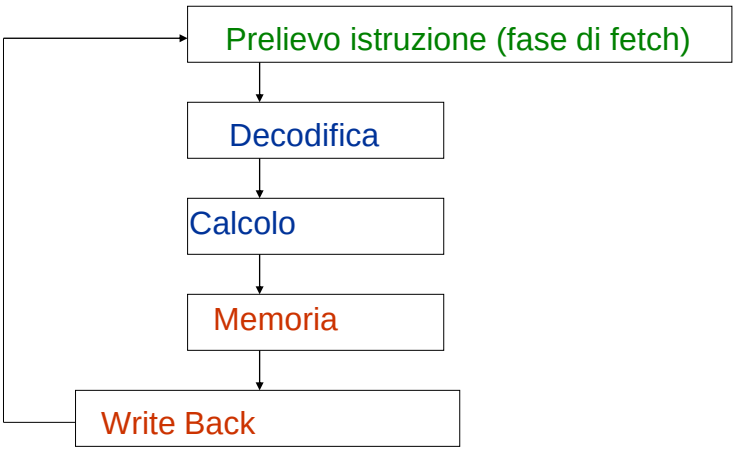
	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	indirizzo				

Istruzioni

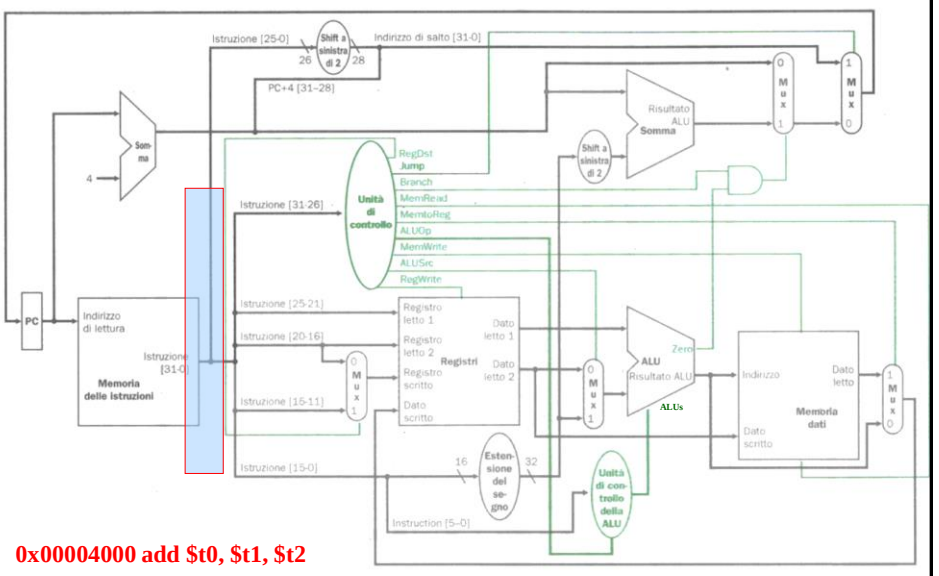
`add $s1, $s2, $s3`	000000	10010	10011	10001	00000	100000	
`beq $s1, $s2, -100`	000100	10001	10010	1111	1111	1110	0111
`lw $t0, 32 ($s3)`	100011	10011	01000	0000	0000	0010	0000
`sw $t0, 32 ($s3)`	101011	10011	01000	0000	0000	0010	0000
`addi $t0, $s3, 64`	001000	10011	01000	0000	0000	0100	0000
`j 0x80000`	000010	00 0000 0100 0000 0000 0000 0000					

CodOp

Ciclo di esecuzione di un'istruzione

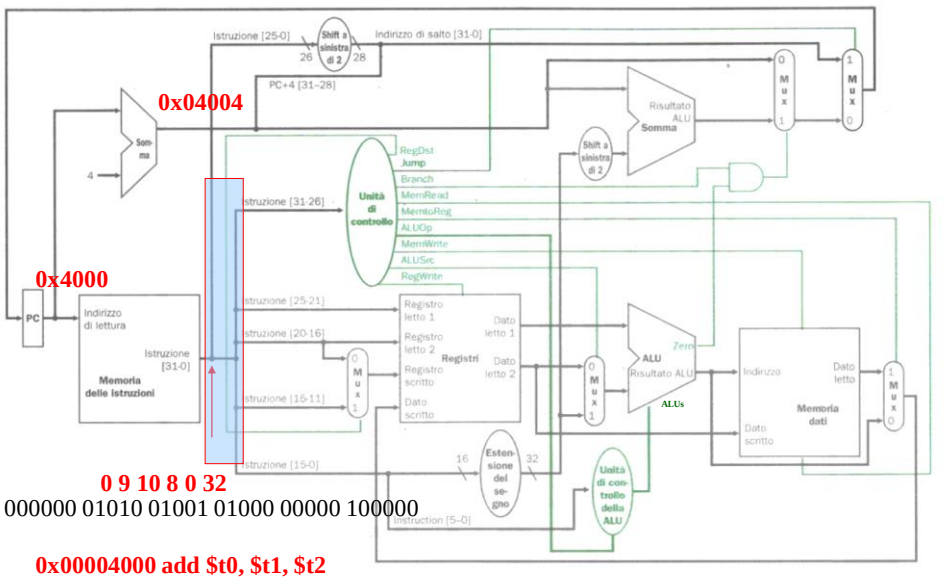


MIPS a singolo ciclo (di clock)

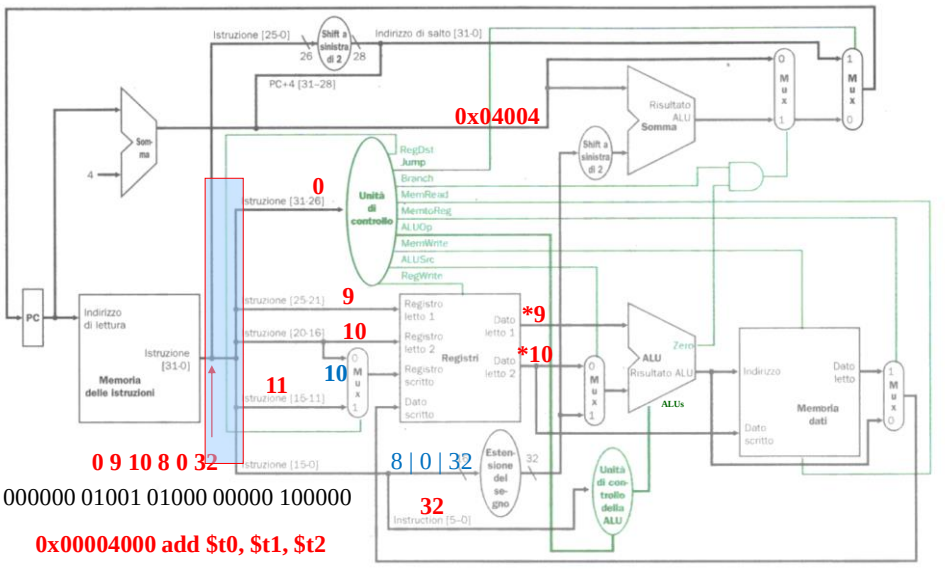


0x00004000 add $t0, $t1, $t2

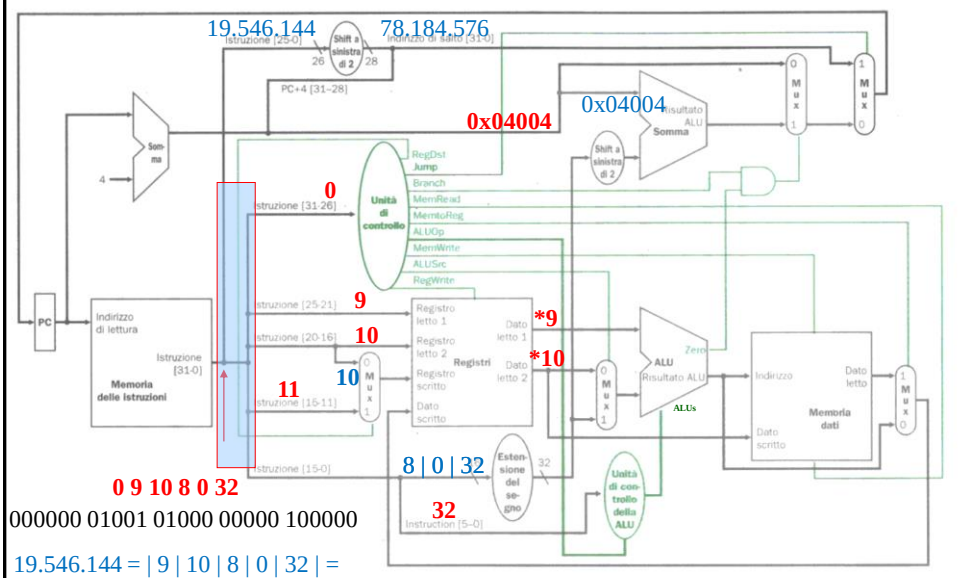
add: fase di fetch



add: fase di decode



add: fase di decode (jump related)

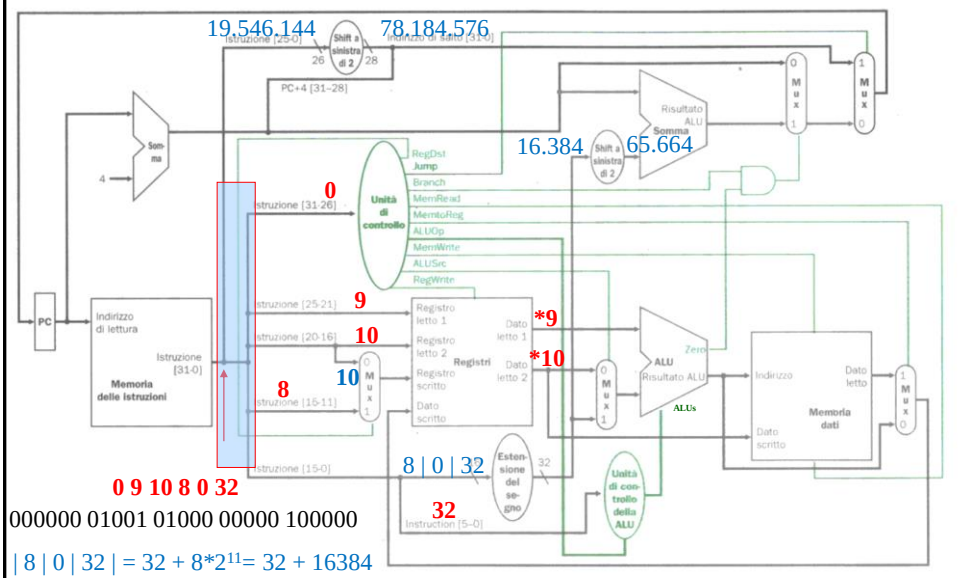


000000 01009 01000 00000 100000

19.546.144 = | 9 | 10 | 8 | 0 | 32 | =
= 32 + 0*2^6 + 8*2^{11} + 10*2^{16} + 9*2^{21}

26 bit meno significativi

add: fase di decode (beq related)



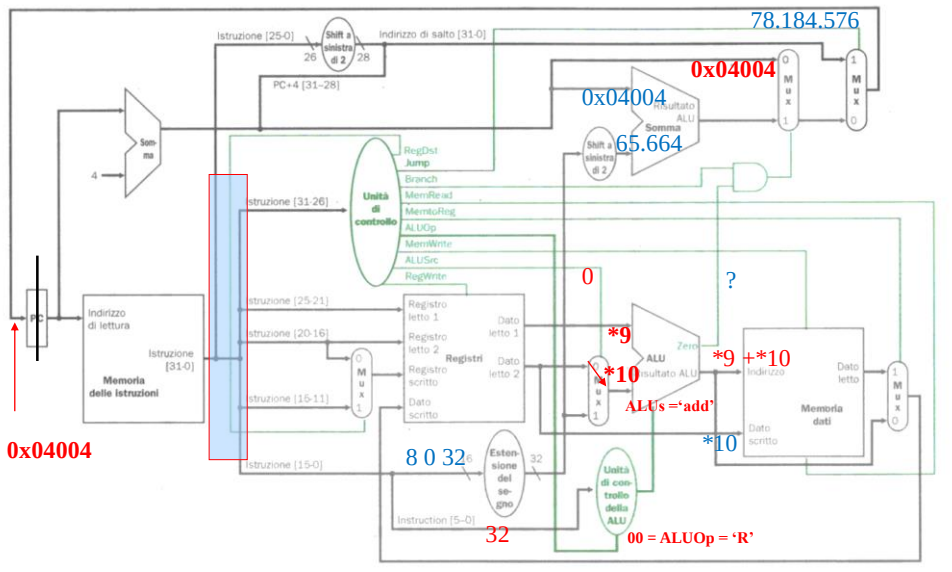
000000 01009 01000 00000 100000

| 8 | 0 | 32 | = 32 + 8*2^{11}= 32 + 16384
= 16416

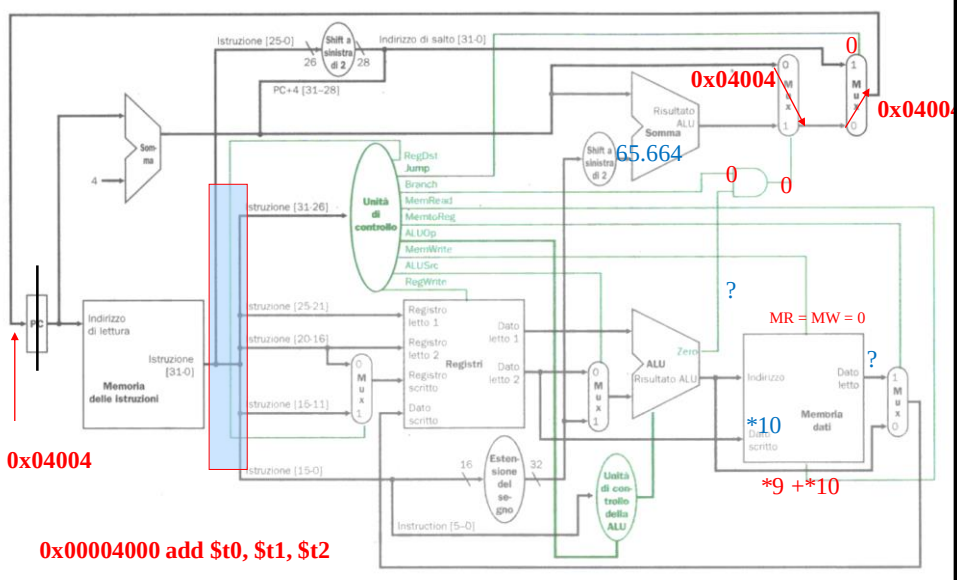
16 bit meno significativi

add: fase di exe (calcolo)

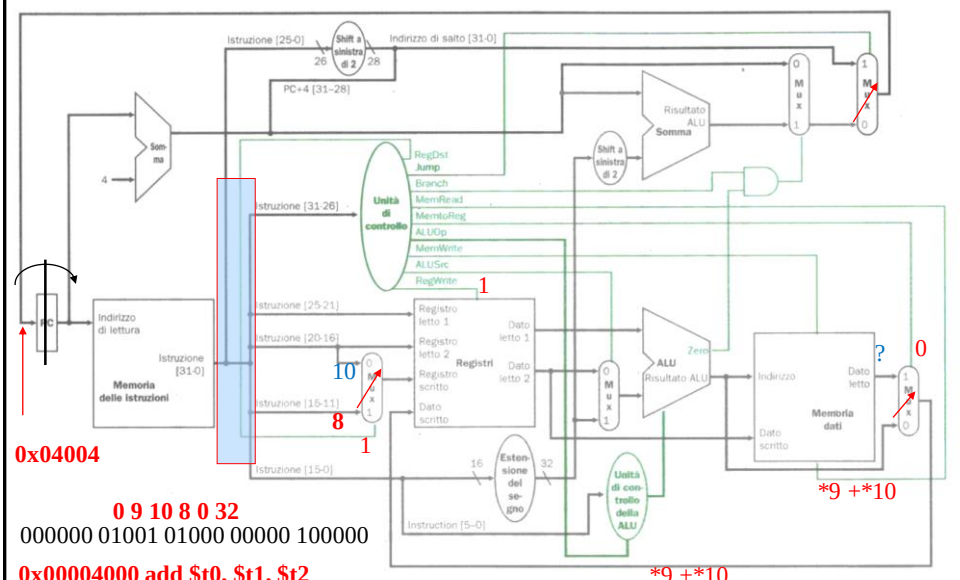
0x00004000 add $t0, $t1, $t2



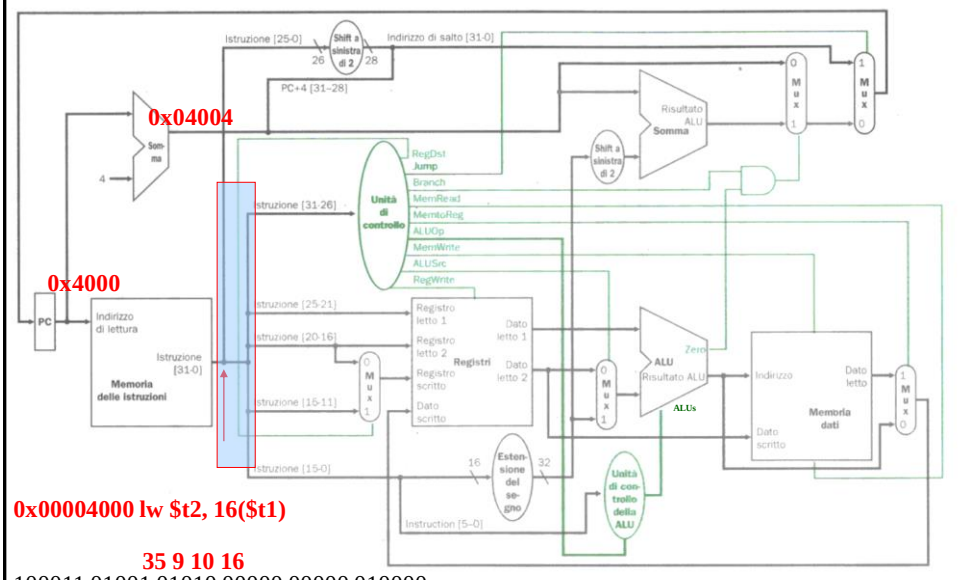
add: fase di memoria (idle)



add: fase di write back

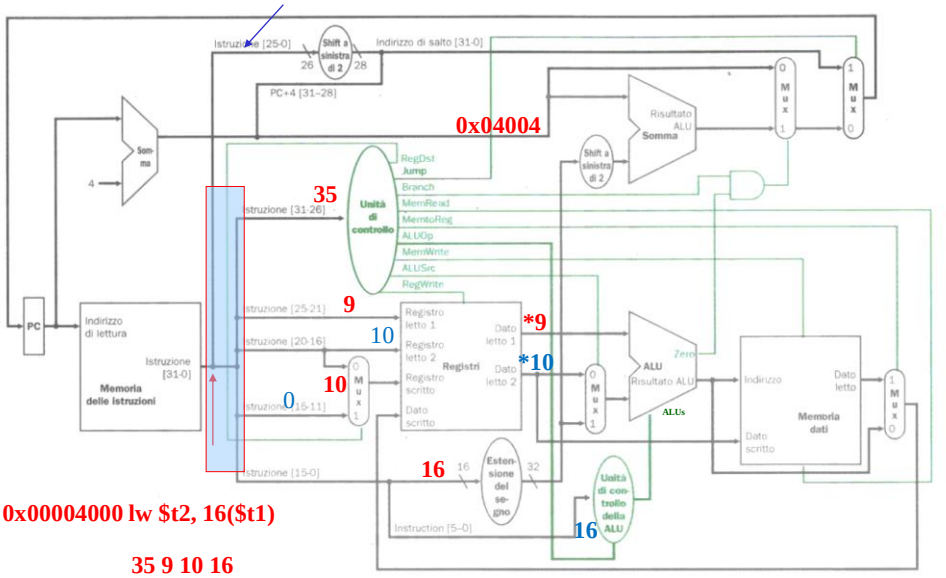


lw: fase di fetch



lw: fase di decode

34.603.008 = $9*2^{21} + 10*2^{16} + 16$

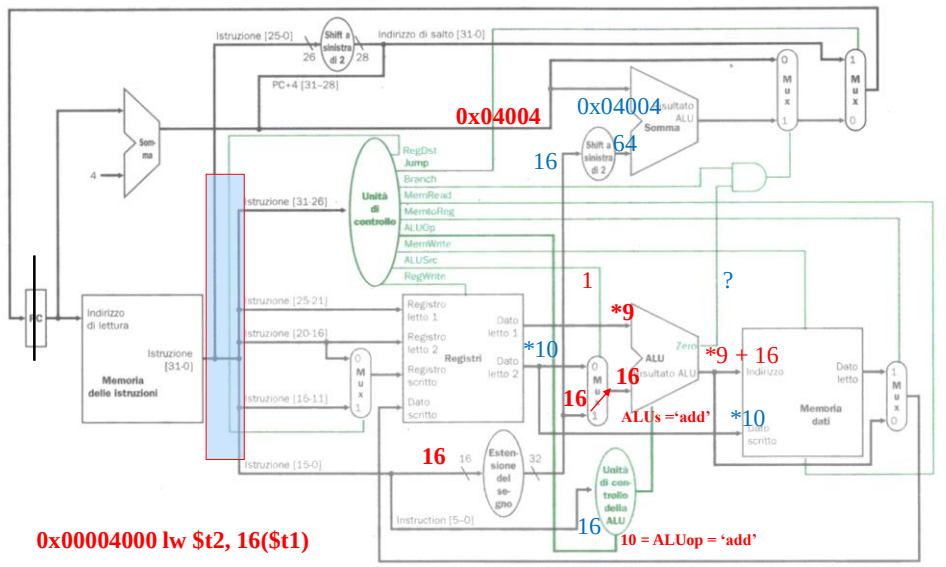


0x00004000 lw $t2, 16($t1)

35 9 10 16

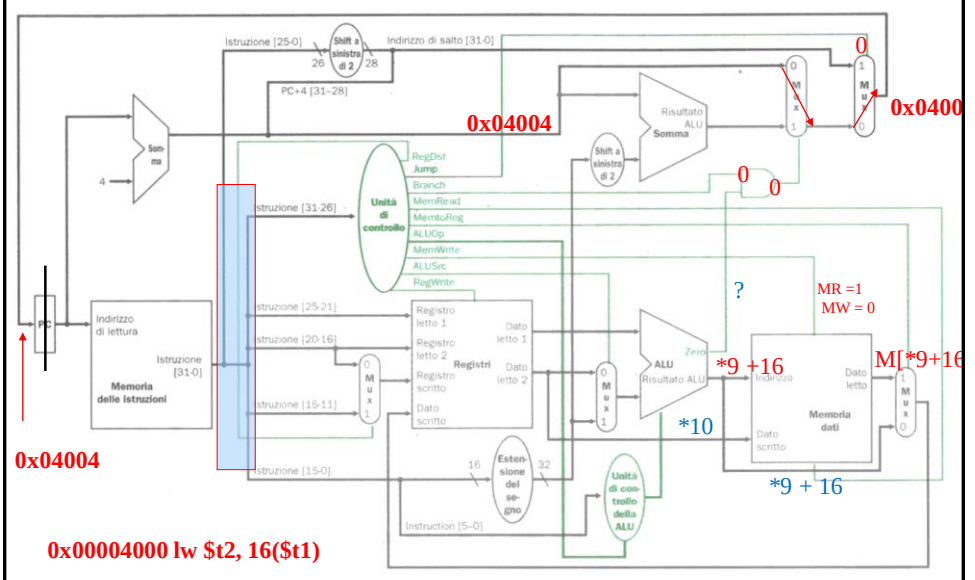
100011 01001 01010 00000 00000 010000

lw: fase di exe (calcolo)

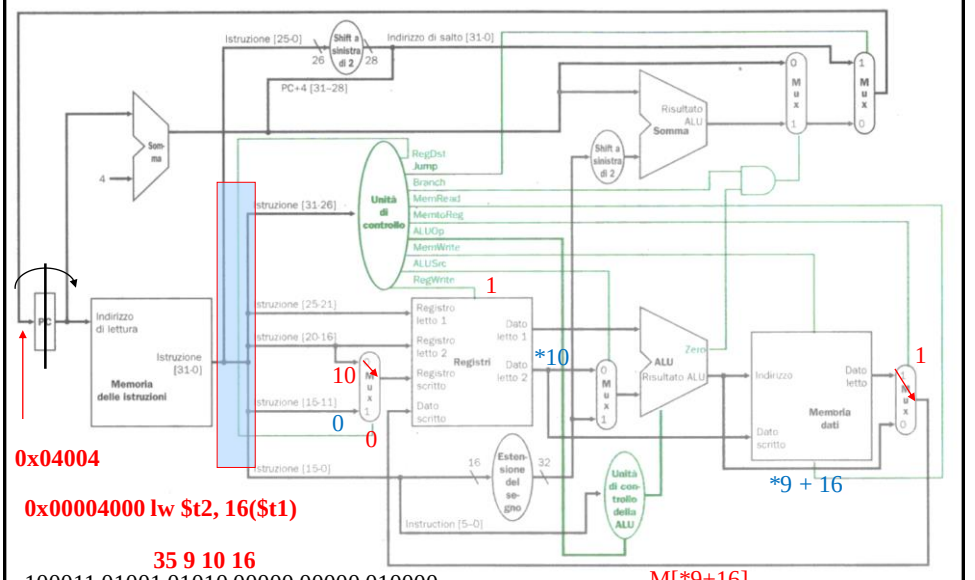


0x00004000 lw $t2, 16($t1)

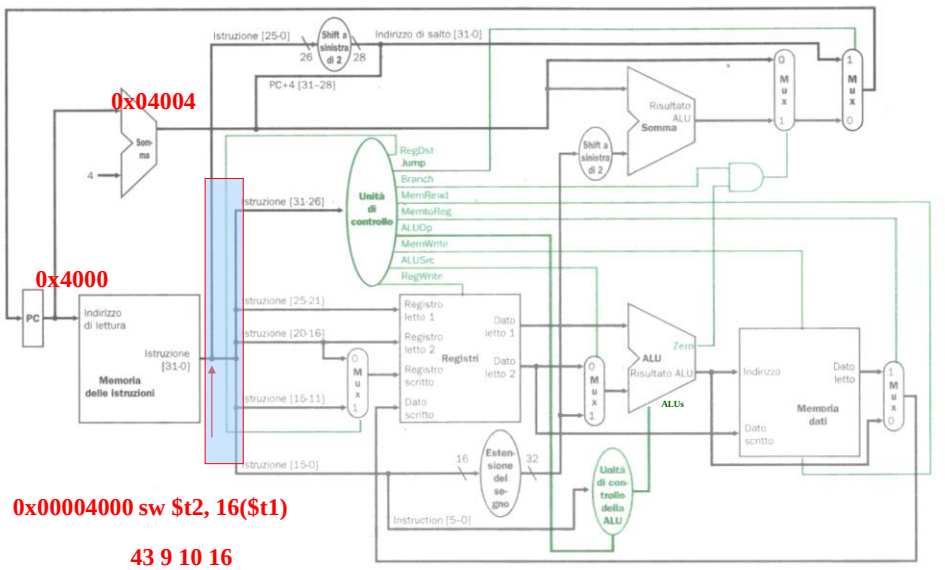
lw: fase di memoria



lw: fase di write back



sw: fase di fetch



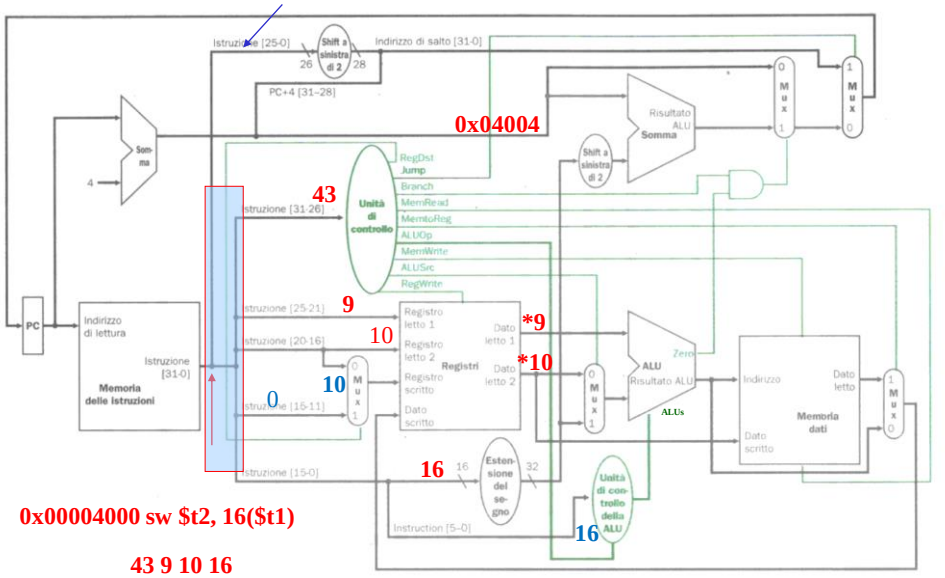
0x00004000 sw $t2, 16($t1)

43 9 10 16

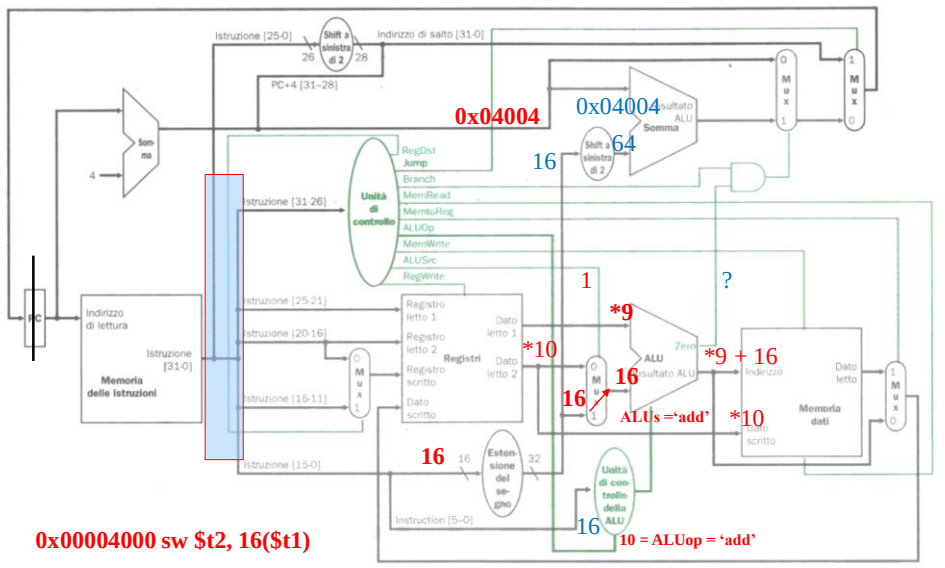
101011 01001 01010 00000 00000 010000

sw: fase di decode

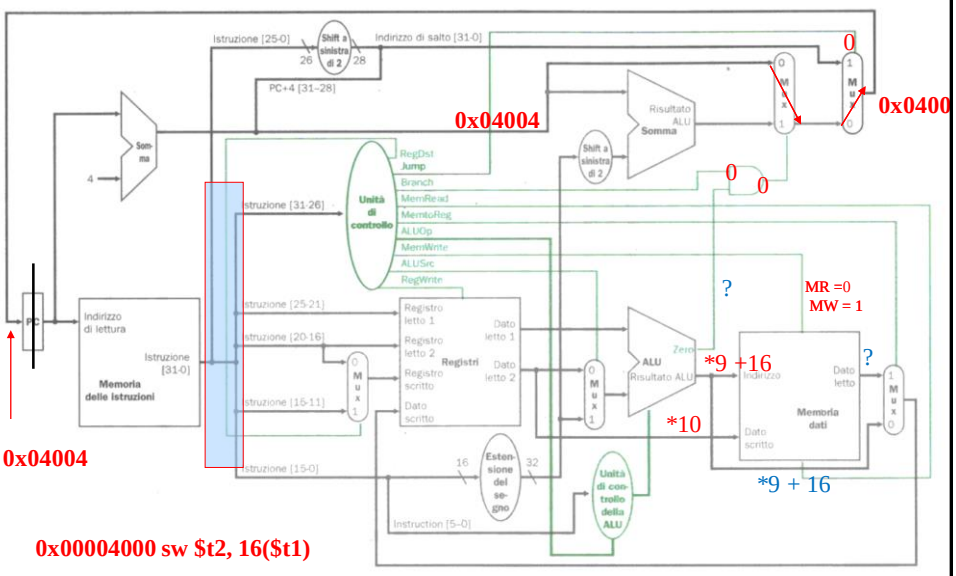
34.603.008 = $9*2^{21} + 10*2^{16} + 16$



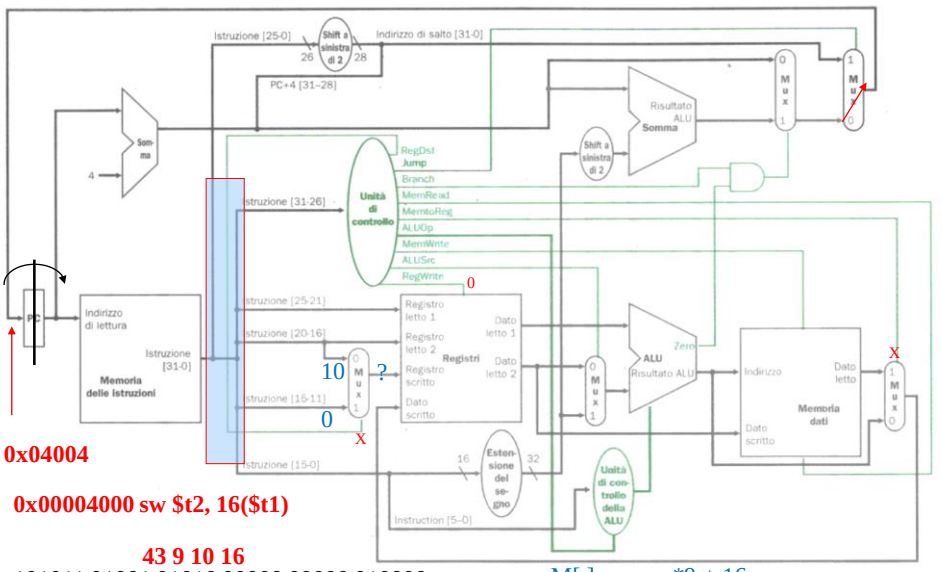
sw: fase di exe (calcolo)



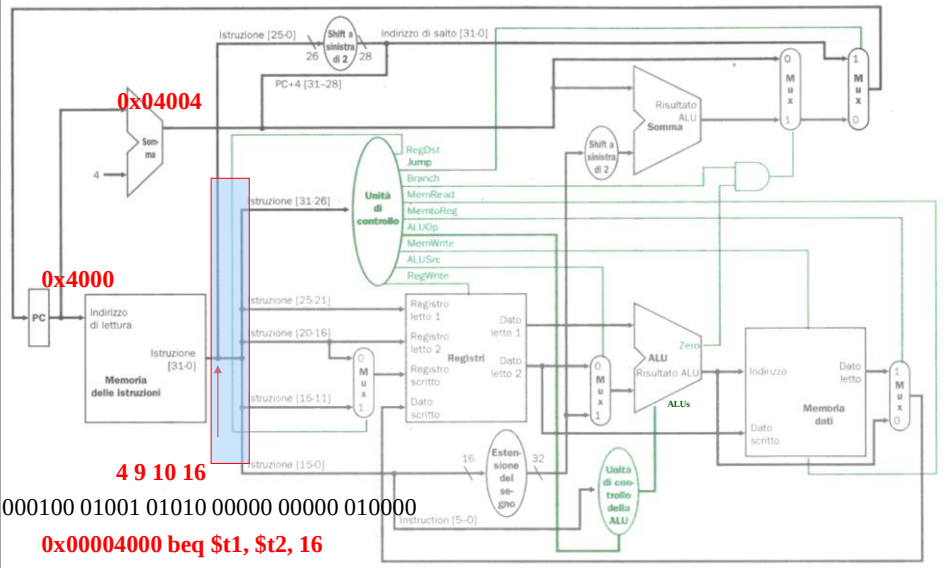
sw: fase di memoria



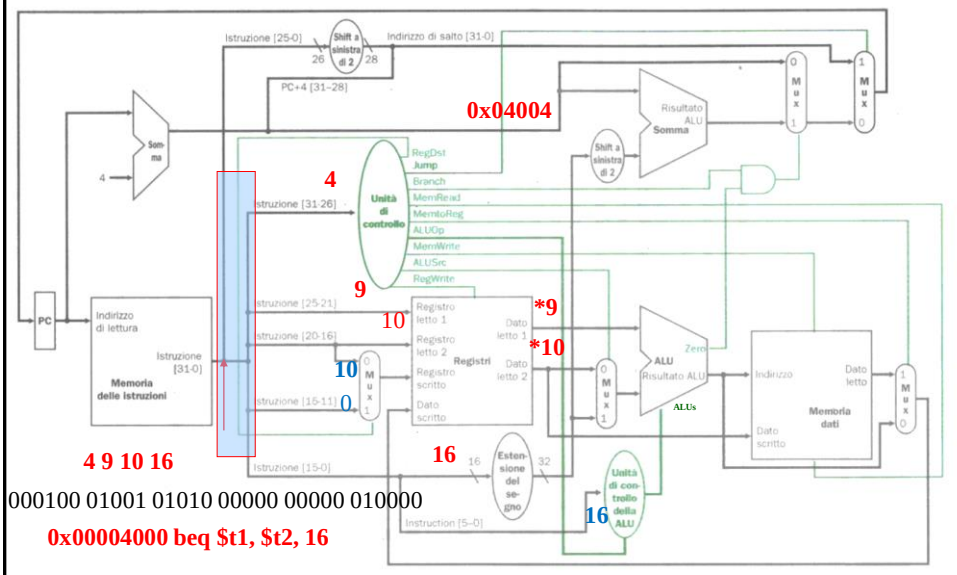
sw: fase di write back (idle)



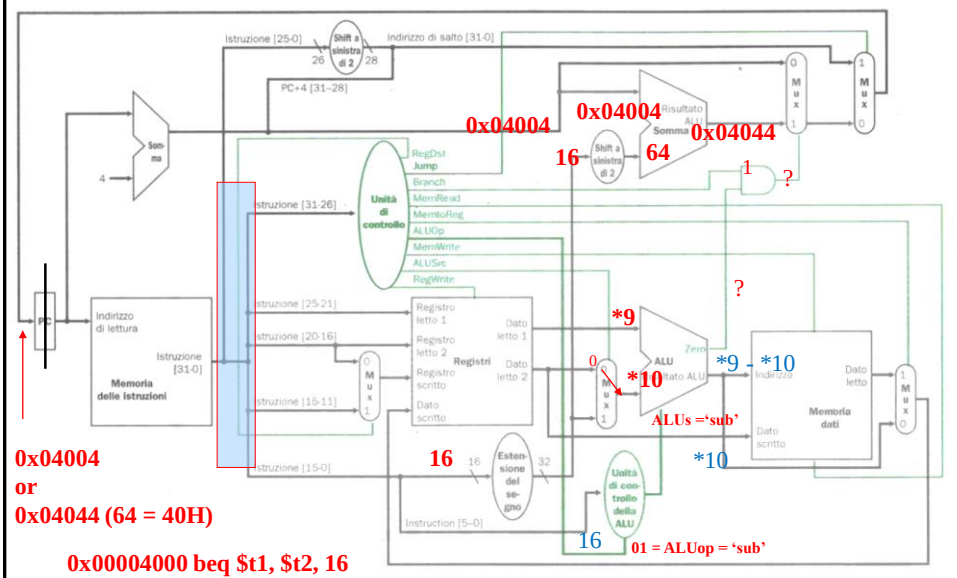
beq: fase di fetch



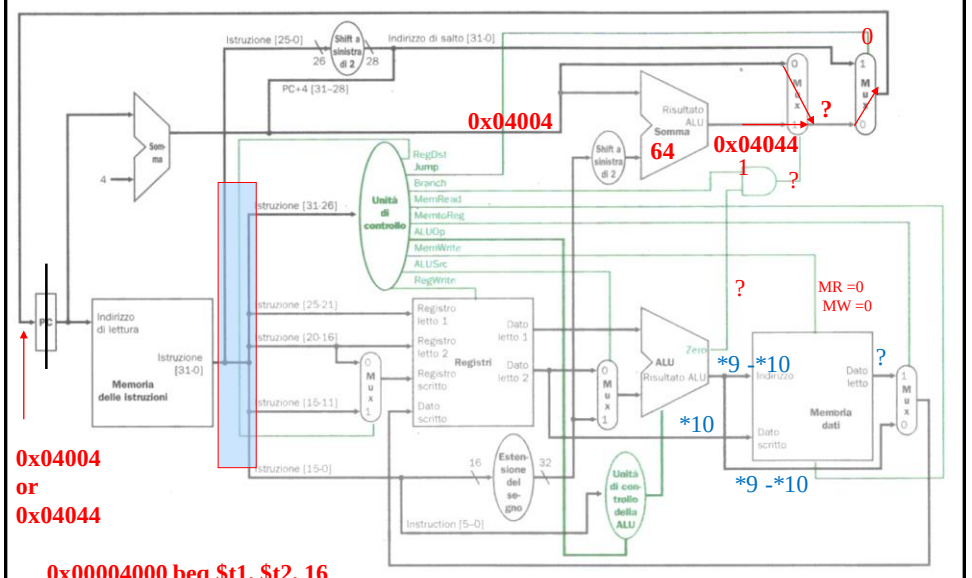
beq: fase di decode



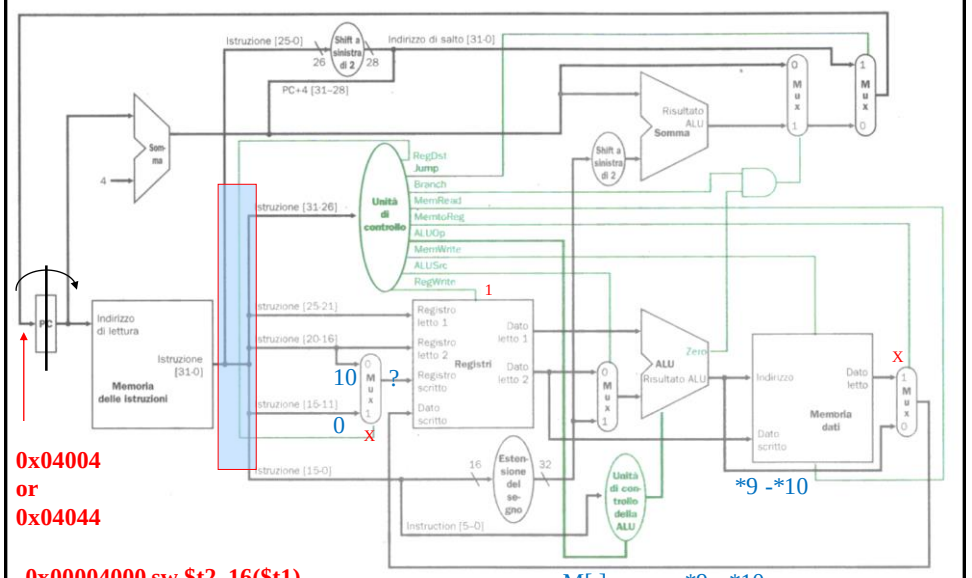
beq: fase di exe (calcolo)



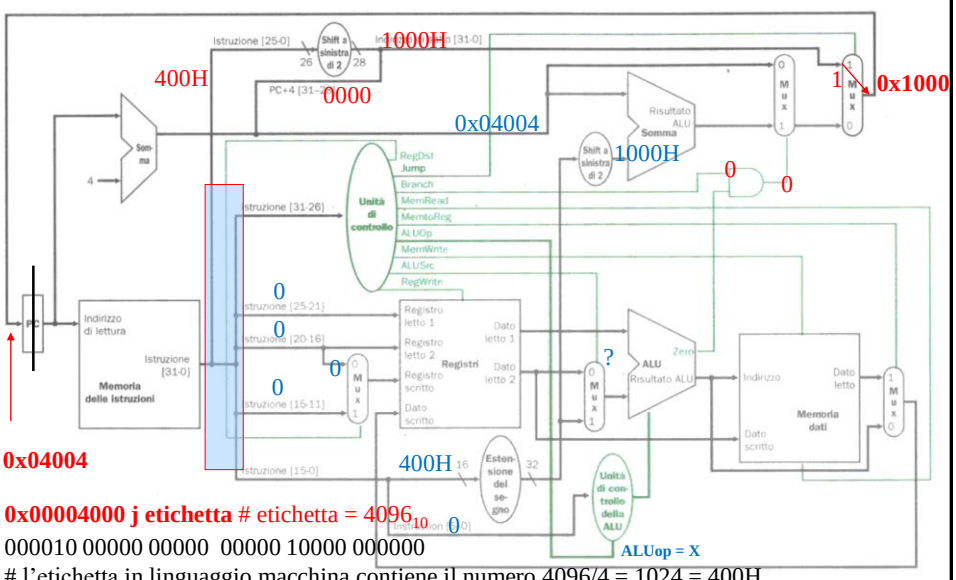
beq: fase di memoria



beq: fase di write back (idle)



Jump: fase di exe (calcolo)

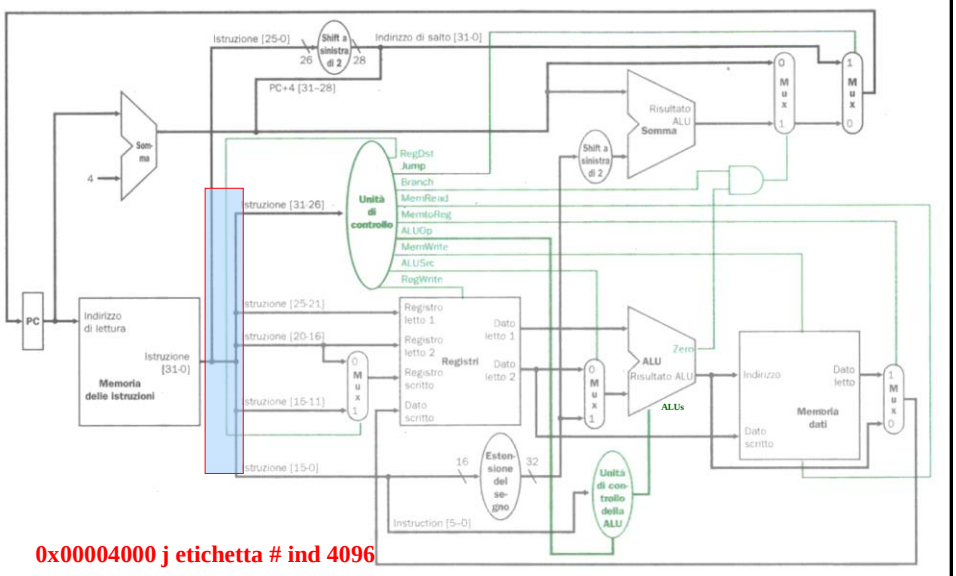


0x00004000 j etichetta # etichetta = 4096_{10}

000010 00000 00000 00000 10000 000000

l'etichetta in linguaggio macchina contiene il numero 4096/4 = 1024 = 400H

Jump: fase di memoria e WB idle



0x00004000 j etichetta # ind 4096

Osservazioni

Il ciclo di esecuzione di un'istruzione si compie in un unico ciclo di clock.

⇩

Ogni unità funzionale può essere utilizzata 1 sola volta.

⇩

Duplicazione Memoria: Memoria dati e memoria istruzioni.
Triplicazione ALU: 3 ALU: 2 sommatori + 1 general purpose.

⇩

Il periodo del clock è determinato dal cammino critico dell'istruzione più lunga: lw.
Utilizzare un clock per ogni fase di esecuzione porterebbe ad un tempo di esecuzione variabile per le diverse istruzioni: beq = 3 cicli di clock, aritmetche e sw, 4 cicli di clock, lw 5 cicli di clock. Si può fare di meglio.

Alcuni esercizi

Specificare il contenuto di tutti i bus durante l'esecuzione delle seguenti istruzioni:

```
0x00000400 or $s5, $t2, $t1

0x00000400 sw $s1, 1024($s0)

0x00010000 add $t4, $s5, $s1

0x00010000 addi $t1, $t4, 100

0x00000400 beq $s2, $s1, label
```

Si supponga label l'istruzione all'indirizzo 0x000410 e che il salto venga preso.

Sommario

- Introduzione
- Administratives
- La CPU a ciclo singolo