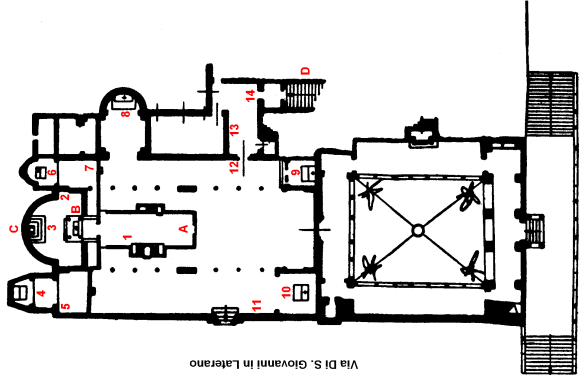


# Calcolatori Elettronici

## Parte VI: Microarchitettura di una CPU

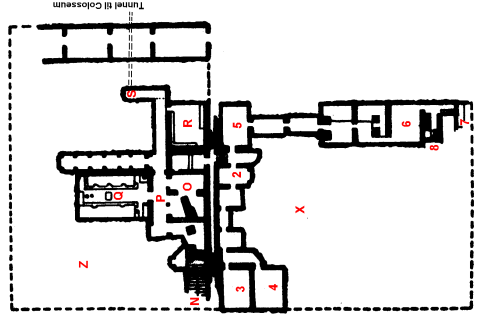
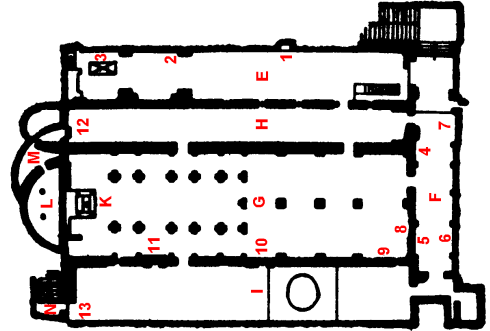
Prof. Riccardo Torlone  
Universita di Roma Tre

# Ritorniamo all'approccio di San Clemente..

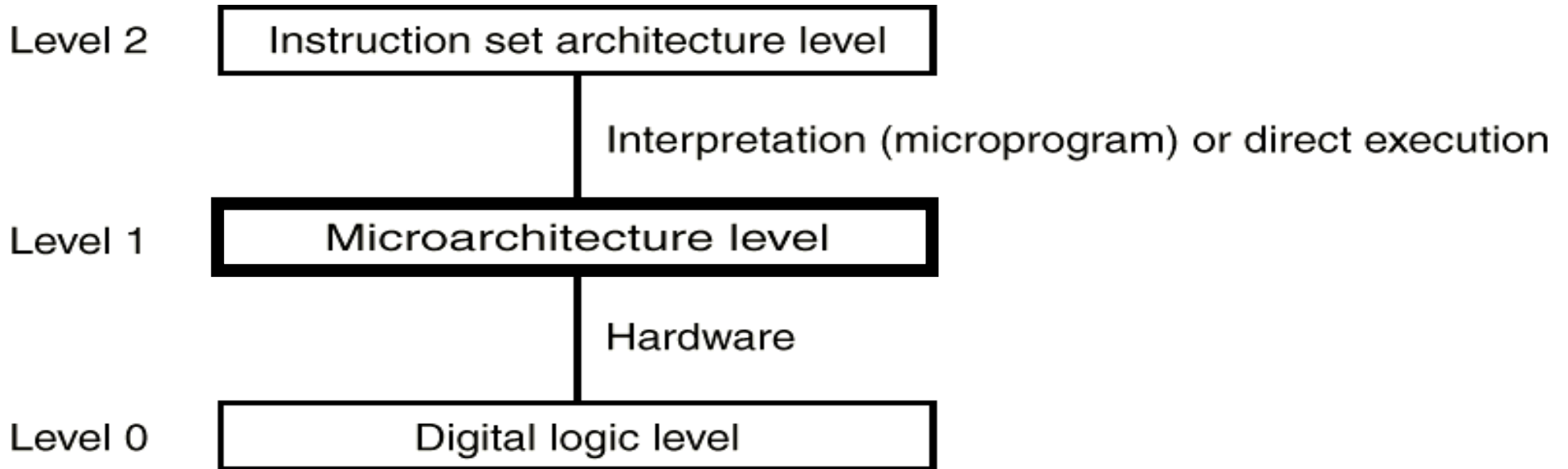


Via Di S. Giovanni in Laterano

Piazza S. Clemente

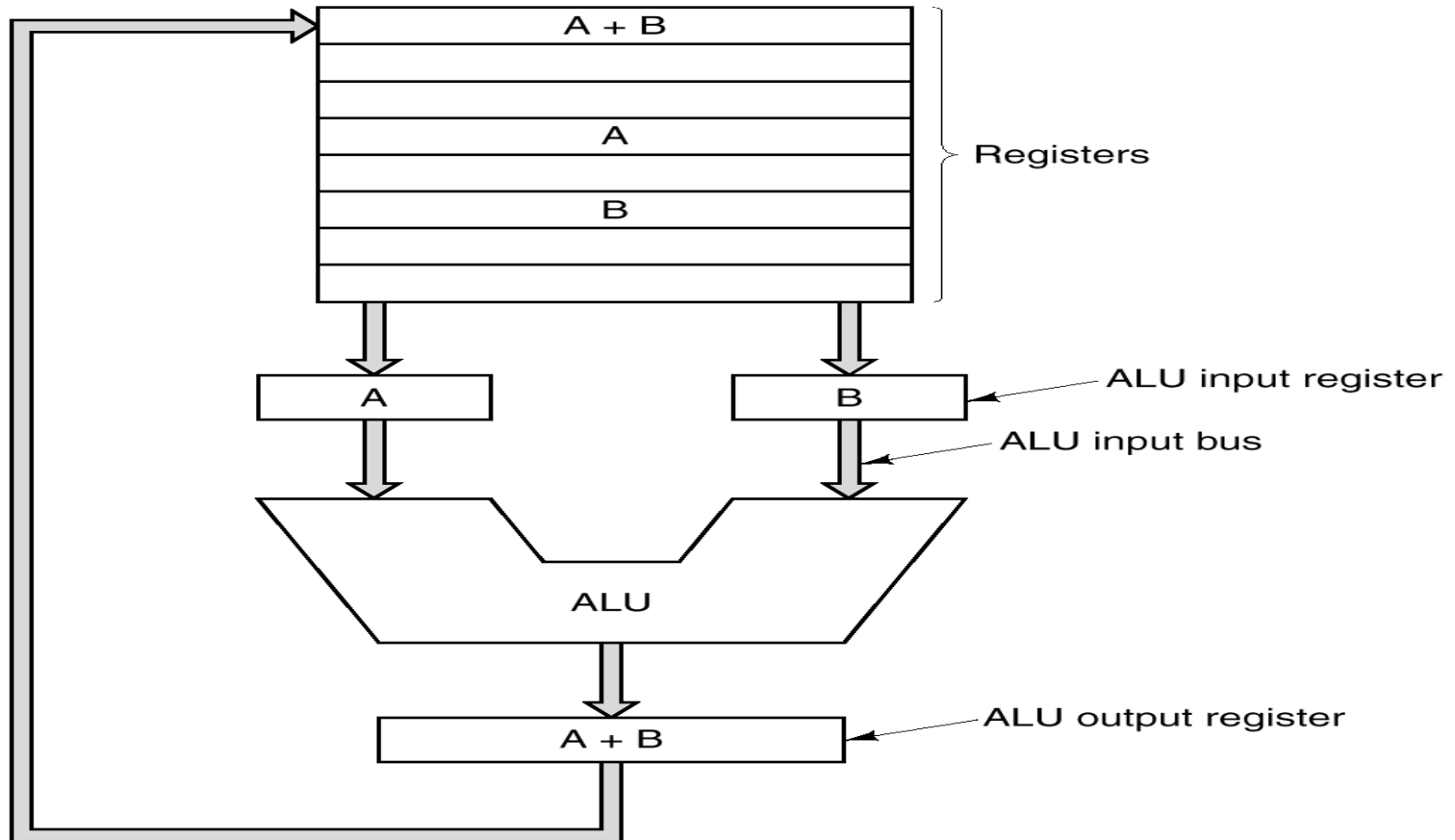


# Il livello della microarchitettura



- Al livello della **microarchitettura** studiamo come la CPU “implementa” le istruzioni macchina mediante i dispositivi digitali (*l'hardware*) a sua disposizione
- La descrizione considera i componenti di base della CPU (registri, ALU, ecc.) e il flusso dei dati tra di essi trascurandone i dettagli realizzativi

# Microarchitettura generica e data path



- La microarchitettura della CPU è tipicamente composta da alcuni registri, una ALU, dei bus interni e alcune linee "di controllo"
- Le istruzioni macchina comandano il funzionamento della CPU e il percorso dei dati (data path)

# Possibili implementazioni

## Esecuzione diretta delle istruzioni (RISC)

- Le istruzioni possono venire eseguite direttamente dalla microarchitettura
- Pro e contro:
  - Repertorio di istruzioni limitato
  - Progettazione dell'HW complessa
  - Esecuzione molto efficiente

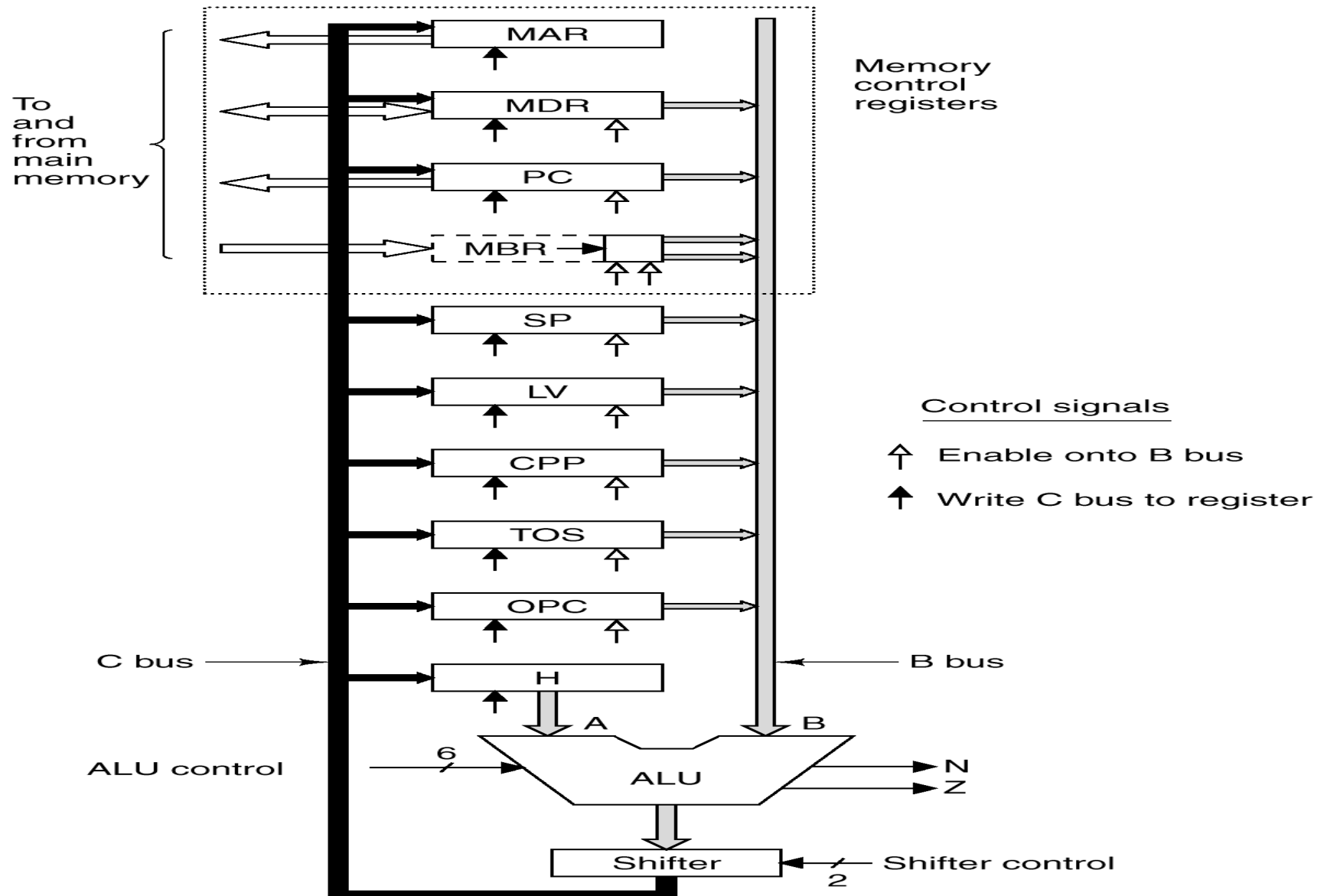
## Interpretazione delle istruzioni (CISC)

- La microarchitettura sa eseguire direttamente solo alcune semplici operazioni
- Ciascuna istruzione è scomposta in una successione di operazioni base poi eseguite dalla microarchitettura
- Pro e contro:
  - Repertorio di istruzioni esteso
  - HW più compatto
  - Flessibilità di progetto

# Un esempio di $\mu$ -architettura

- Implementazione di un JVM (Java Virtual Machine) con sole istruzioni su interi
- In questo corso ci limitiamo a:
  - La microarchitettura (data path)
  - La temporizzazione di esecuzione
  - L'accesso alla memoria (cache)
  - Il formato delle micro-istruzioni
  - La sezione di controllo
- Sul libro l'esempio è sviluppato fino alla definizione di un microprogramma completo per una JVM (con aritmetica intera)
- Questa ultima parte non fa parte del programma

# Il Cammino dei Dati nella JVM

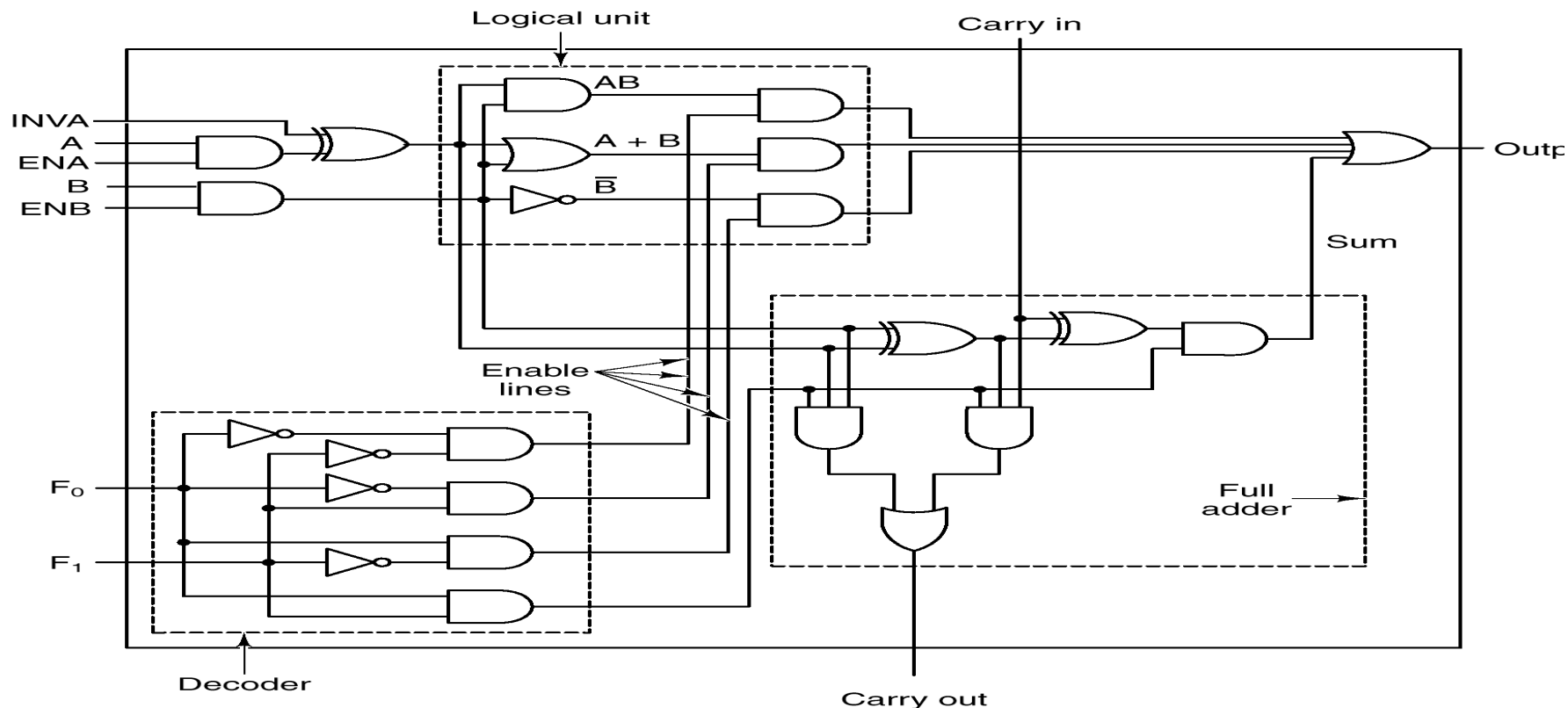


## Il Cammino dei Dati (2)

- **Registri:** contraddistinti da nomi simbolici ciascuno con una precisa funzione
- **Bus B:** presenta il contenuto di un registro all'ingresso B della ALU
- **ALU:** ha come ingressi il bus B e il registro H (*holding register*)
- **Shifter:** consente di effettuare vari tipi di *shift* sull'uscita della ALU
- **Bus C:** permette di caricare l'uscita dello *shifter* in uno o più registri
  
- **Segnali di controllo:**
  - *B bus enable:* trasferisce il contenuto di un registro sul bus B
  - *Write C bus:* trasferisce il contenuto dello *shifter* in uno o più registri
  - Controllo della ALU: seleziona una delle funzioni calcolabili dalla ALU
  - Controllo dello shifter: specifica se e come scalare l'uscita della ALU

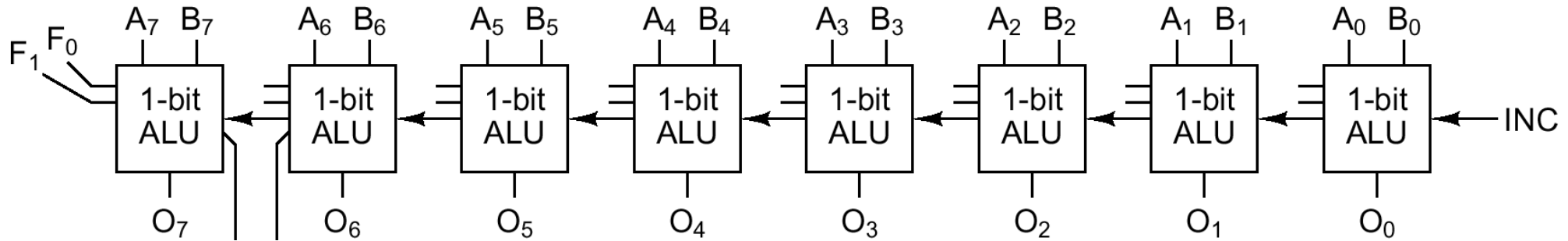


# Utilizziamo la ALU vista



- A e B sono bit omologhi degli operandi
- $F_0$  e  $F_1$  selezionano la funzione (00: AND), (01: OR), (10: NOT), (11: SUM)
- ENA ed ENB sono segnali di enable e INVA permette di negare A
- Default ENA=ENB=1 e INVA=0

# L'ALU è a 32 bit



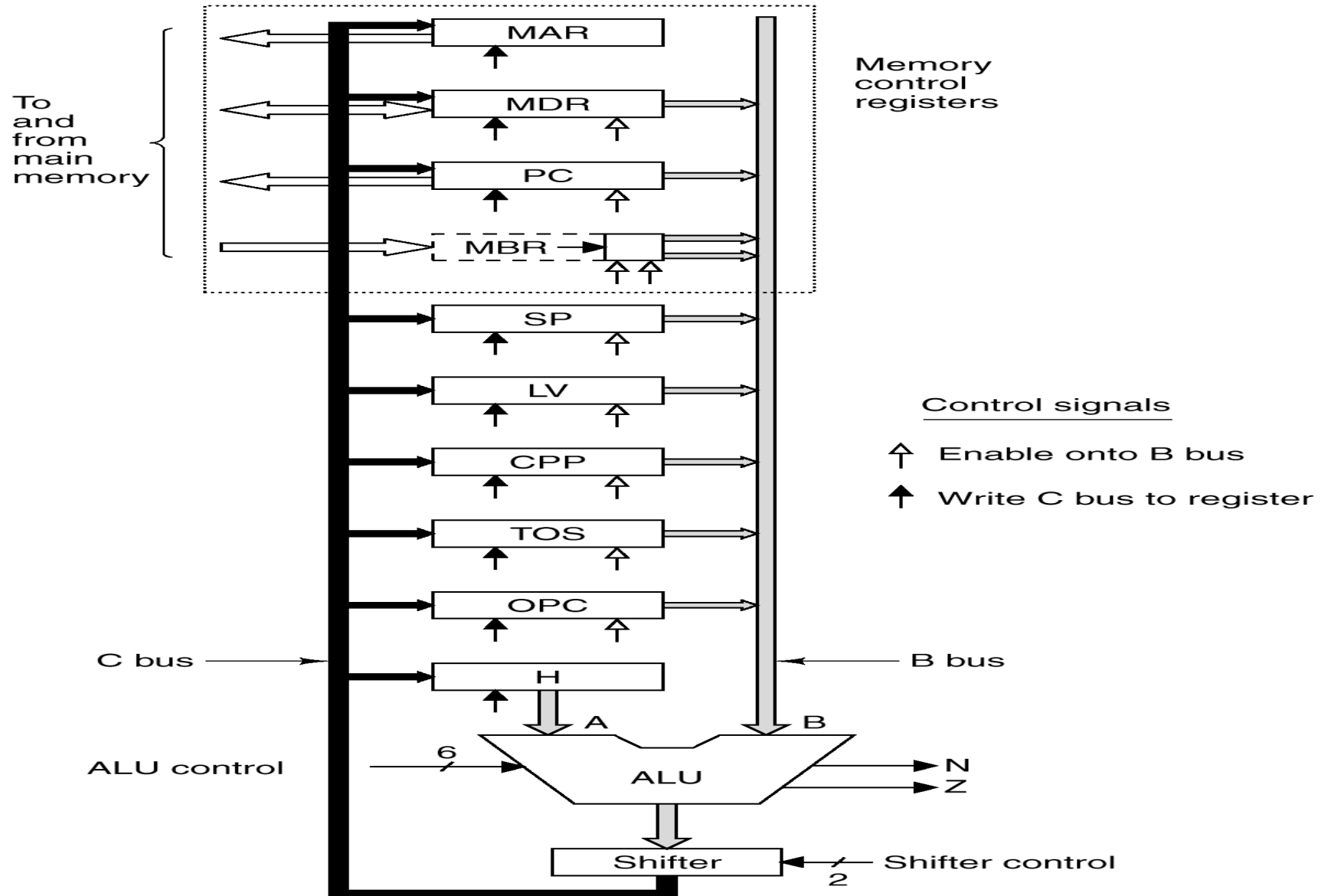
- Realizzata connettendo 32 ALU ad 1 bit (bit slices)
- INC incrementa la somma di 1 ( $A+1$ ,  $A+B+1$ )

# Funzioni della ALU

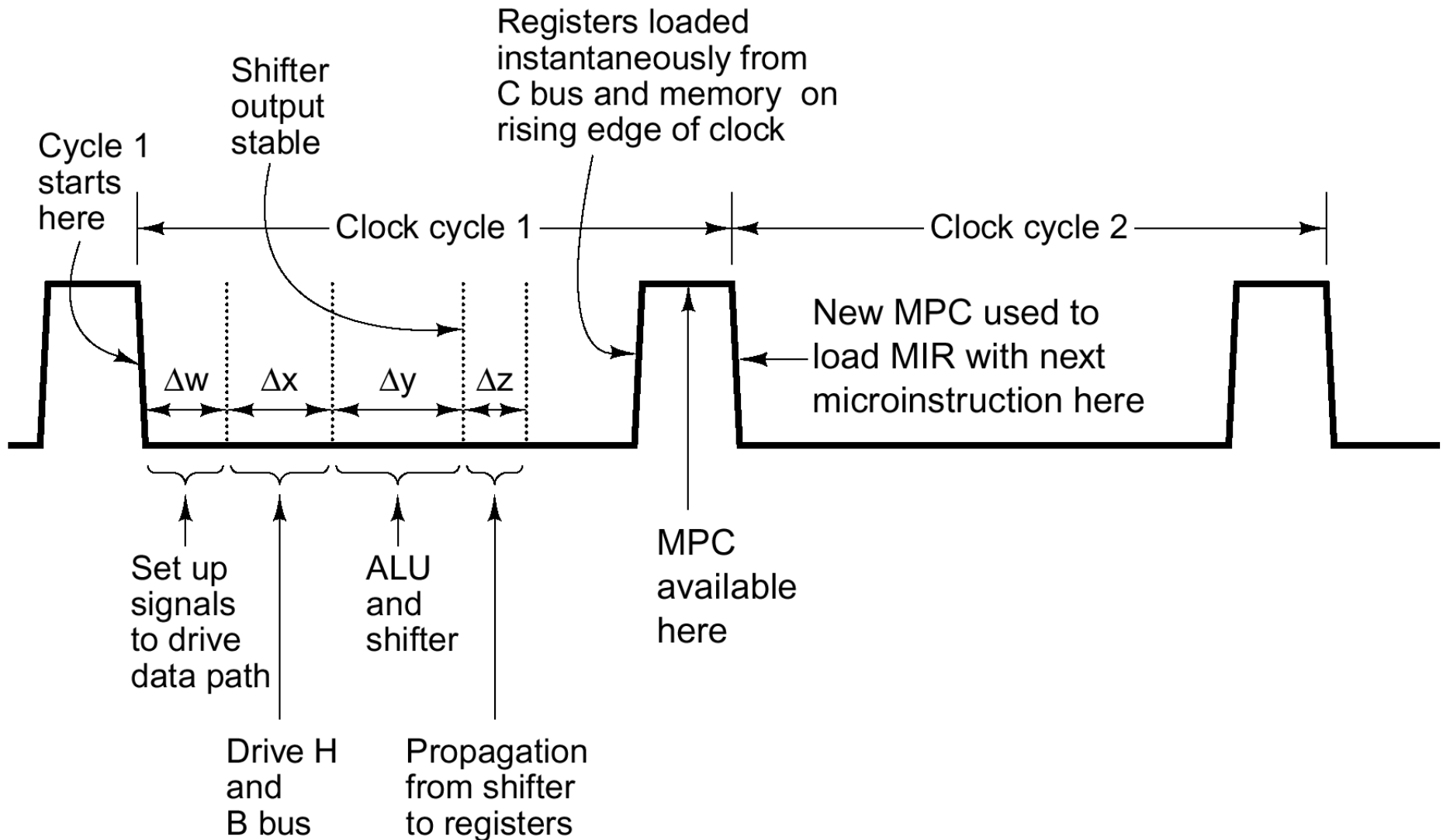
| $F_0$ | $F_1$ | ENA | ENB | INVA | INC | Function       |
|-------|-------|-----|-----|------|-----|----------------|
| 0     | 1     | 1   | 0   | 0    | 0   | A              |
| 0     | 1     | 0   | 1   | 0    | 0   | B              |
| 0     | 1     | 1   | 0   | 1    | 0   | $\overline{A}$ |
| 1     | 0     | 1   | 1   | 0    | 0   | $\overline{B}$ |
| 1     | 1     | 1   | 1   | 0    | 0   | A + B          |
| 1     | 1     | 1   | 1   | 0    | 1   | A + B + 1      |
| 1     | 1     | 1   | 0   | 0    | 1   | A + 1          |
| 1     | 1     | 0   | 1   | 0    | 1   | B + 1          |
| 1     | 1     | 1   | 1   | 1    | 1   | B - A          |
| 1     | 1     | 0   | 1   | 1    | 0   | B - 1          |
| 1     | 1     | 1   | 0   | 1    | 1   | -A             |
| 0     | 0     | 1   | 1   | 0    | 0   | A AND B        |
| 0     | 1     | 1   | 1   | 0    | 0   | A OR B         |
| 0     | 1     | 0   | 0   | 0    | 0   | 0              |
| 1     | 1     | 0   | 0   | 0    | 1   | 1              |
| 1     | 1     | 0   | 0   | 1    | 0   | -1             |

- ENA e ENB abilitano o inibiscono gli ingressi della ALU
- INVA e INC permettono di fare il C2 di A, utile per le sottrazioni
- Possibile incrementare sia A che B e generare le costanti 0,1 e -1

# Il Cammino dei Dati nella JVM



# Temporizzazione del ciclo base



# Temporizzazione del Ciclo

In ciascun ciclo di clock viene eseguita una microistruzione, cioè:

- 1) Caricamento di un registro sul bus B
- 2) Assestamento di ALU e shifter
- 3) Caricamento di registri dal bus C

Temporizzazione:

- Fronte di discesa: inizio del ciclo
- $\Delta w$ : tempo assestamento segnali di controllo
- $\Delta x$ : tempo propagazione lungo bus B
- $\Delta y$ : tempo assestamento ALU e shifter
- $\Delta z$ : tempo propagazione lungo bus C
- Fronte di salita: caricamento registri dal bus C

I tempi  $\Delta w$ ,  $\Delta x$ ,  $\Delta y$ ,  $\Delta z$ , possono essere pensati come sottocicli (impliciti)

# Accesso alla Memoria

Accesso parallelo a due memorie:

- Memoria Dati: 32 bit indirizzabili a word (in lettura e scrittura)
- Memoria Istruzioni: 8 bit indirizzabili a byte (solo in lettura)

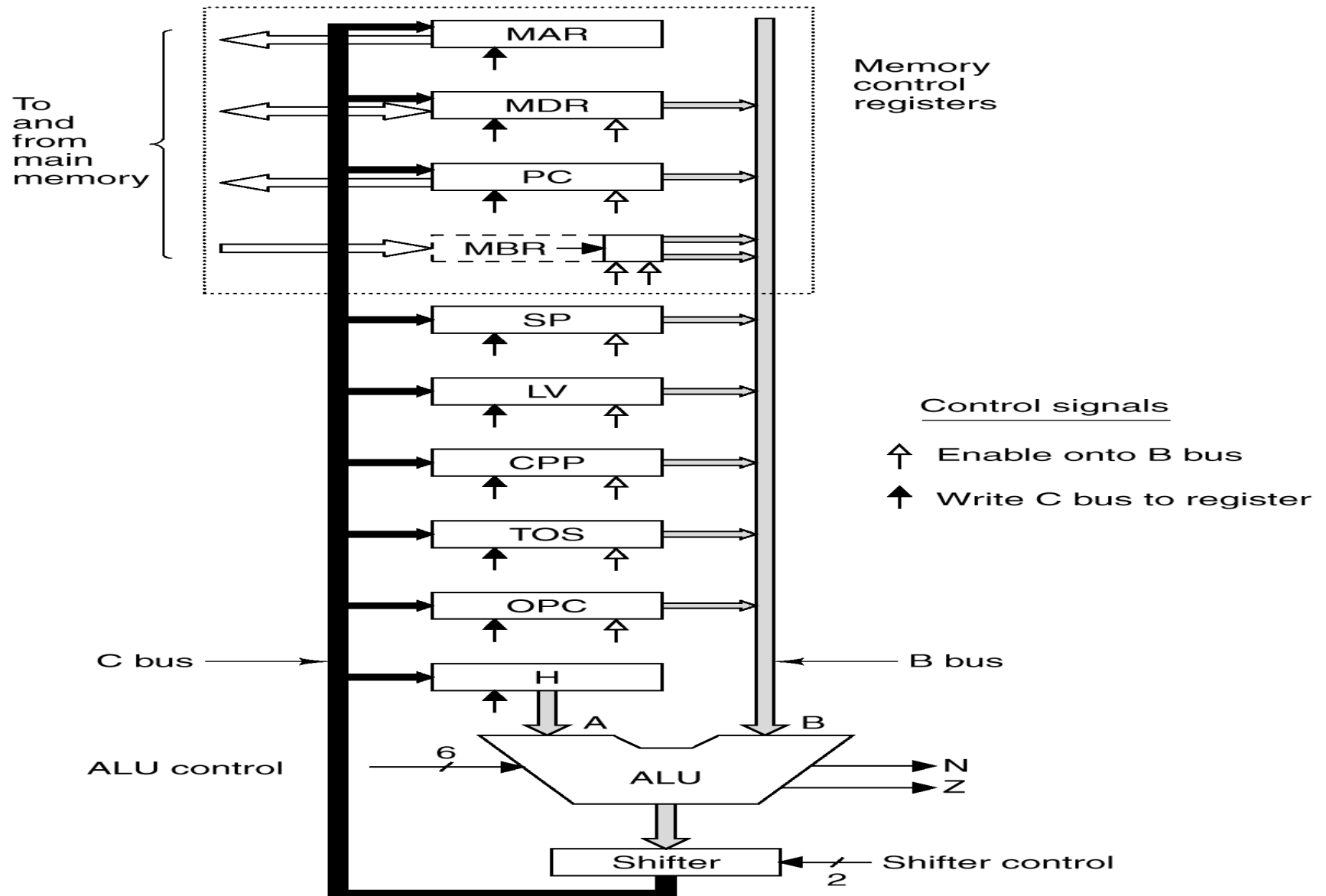
Registri coinvolti:

- MAR (Memory Address Register): contiene l'indirizzo della word dati
- MDR (Memory Data Register): contiene la word dati
- PC (Program Counter): contiene l'indirizzo del byte di codice
- MBR (Memory Buffer Register): riceve il byte di codice (sola lettura)

Caricamento di B da parte di MBR:

- Estensione a 32 bit con tutti 0
- Estensione del bit più significativo (sign extension)

# Il Cammino dei Dati nella JVM





# Struttura delle $\mu$ -istruzioni

Una  $\mu$ -istruzione da 36 bit deve contenere:

- Tutti i segnali di controllo da inviare al data path durante il ciclo
- Le informazioni per la scelta della  $\mu$ -istruzione successiva

Segnali di controllo:

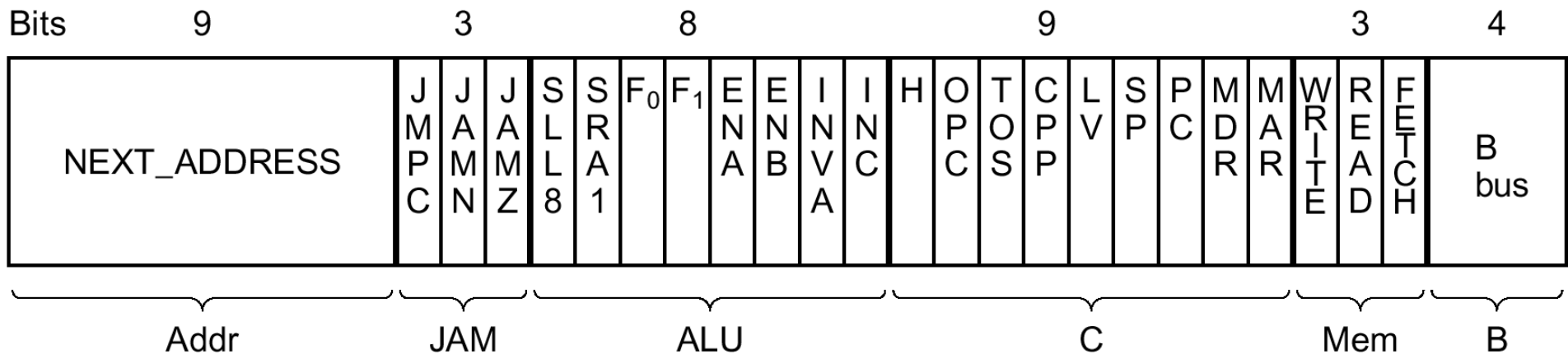
- 9 Selezione registri sul bus C
- 9 Selezione registro sul bus B
- 8 Funzioni ALU e shifter
- 2 Lettura e scrittura dati (MAR/MDR)
- 1 Lettura istruzioni (PC/MBR)

Selezione  $\mu$ -istruzione successiva:

- 9 Indirizzo  $\mu$ -istruzione (su 512)
- 3 Modalità di scelta

Dato che si invia su B solo un registro per volta, si codificano 9 segnali con 4

# Formato delle $\mu$ -istruzioni

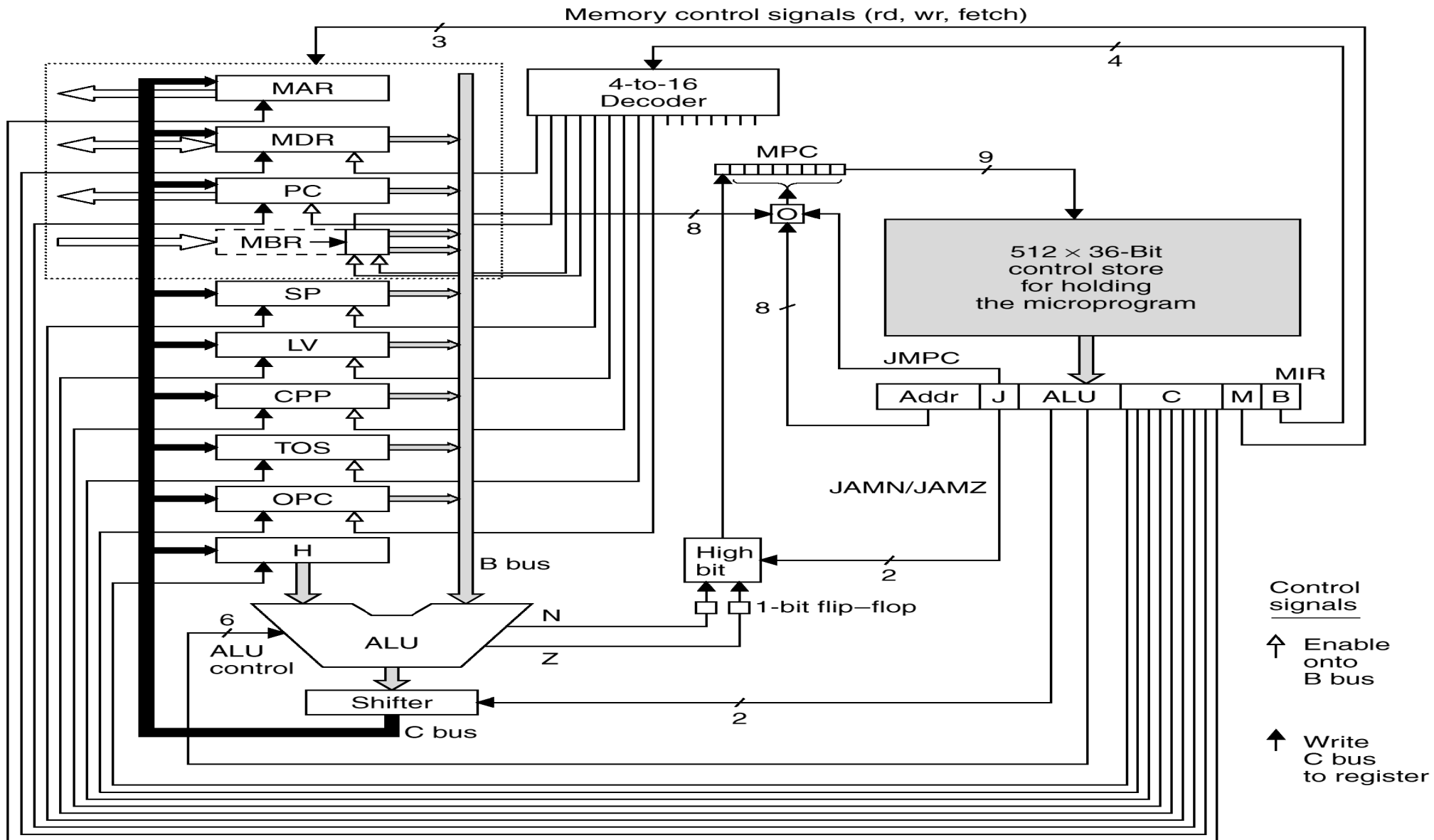


- **Addr:** Indirizzo prossima  $\mu$ -istruzione
- **JAM:** Scelta prossima  $\mu$ -istruzione
- **ALU:** Comandi ALU e shifter
- **C:** Registri da caricare da C
- **Mem:** Controllo memoria
- **B:** Registro da inviare su B

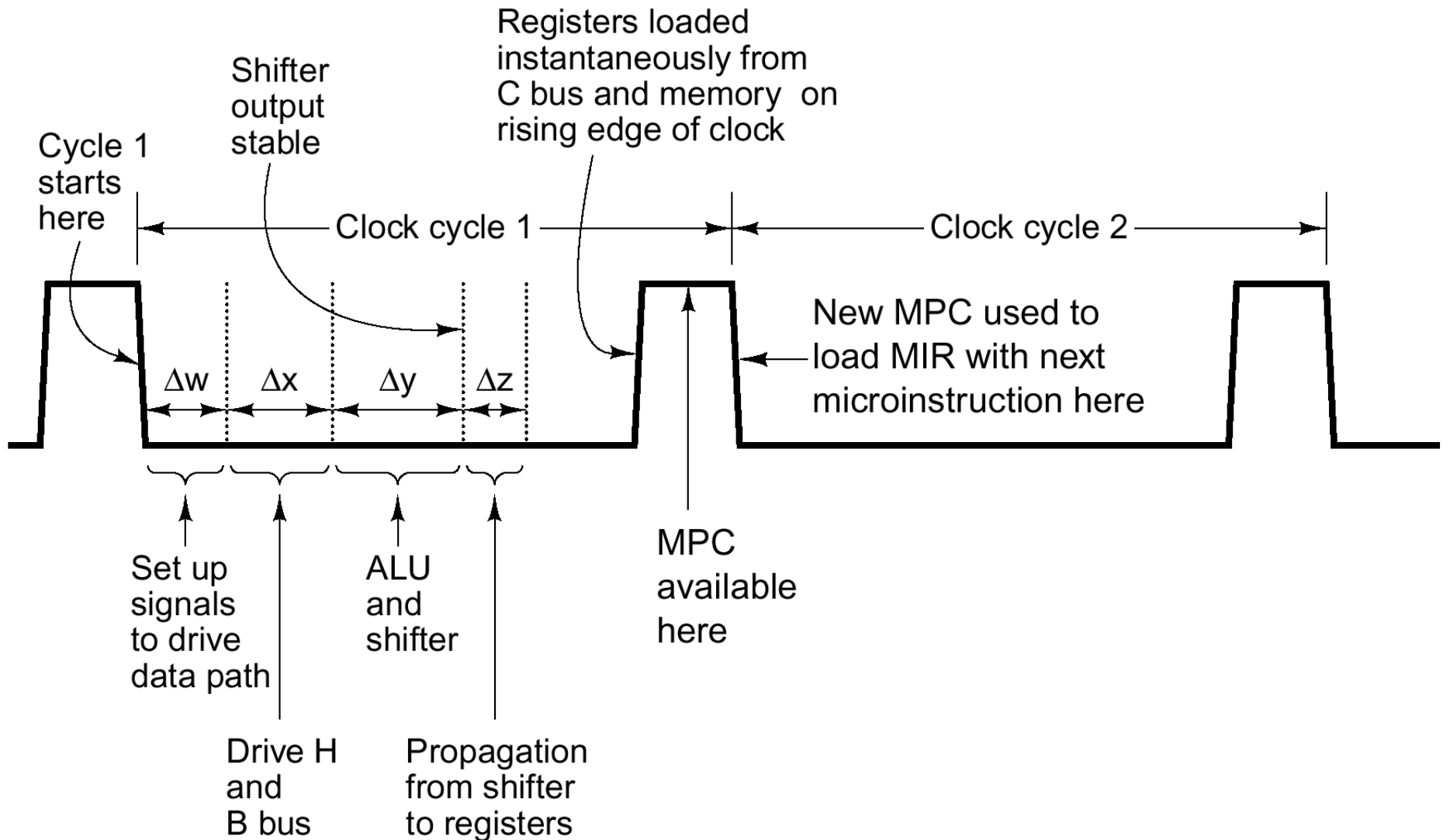
## B bus registers

|          |           |
|----------|-----------|
| 0 = MDR  | 5 = LV    |
| 1 = PC   | 6 = CPP   |
| 2 = MBR  | 7 = TOS   |
| 3 = MBRU | 8 = OPC   |
| 4 = SP   | 9-15 none |

# La Sezione di Controllo



# Temporizzazione del ciclo base



## La Sezione di Controllo (2)

- Control Store: è una ROM 512×36 bit che contiene le  $\mu$ -istruzioni
- MPC (Micro Program Counter): contiene l'indirizzo della prossima  $\mu$ -istruzione
- MIR (MicroInstruction Register): contiene la  $\mu$ -istruzione corrente
- Il contenuto di MPC diviene stabile sul livello alto del clock
- La microistruzione viene caricata in MIR sul fronte di discesa dell'impulso di clock
- Temporizzazione della memoria:
  - Inizio ciclo di memoria subito dopo il caricamento di MAR e di PC
  - Ciclo di memoria durante il successivo ciclo di clock
  - Dati disponibili in MDR e MBR all'inizio del ciclo ancora successivo

# Scelta della $\mu$ -istruzione

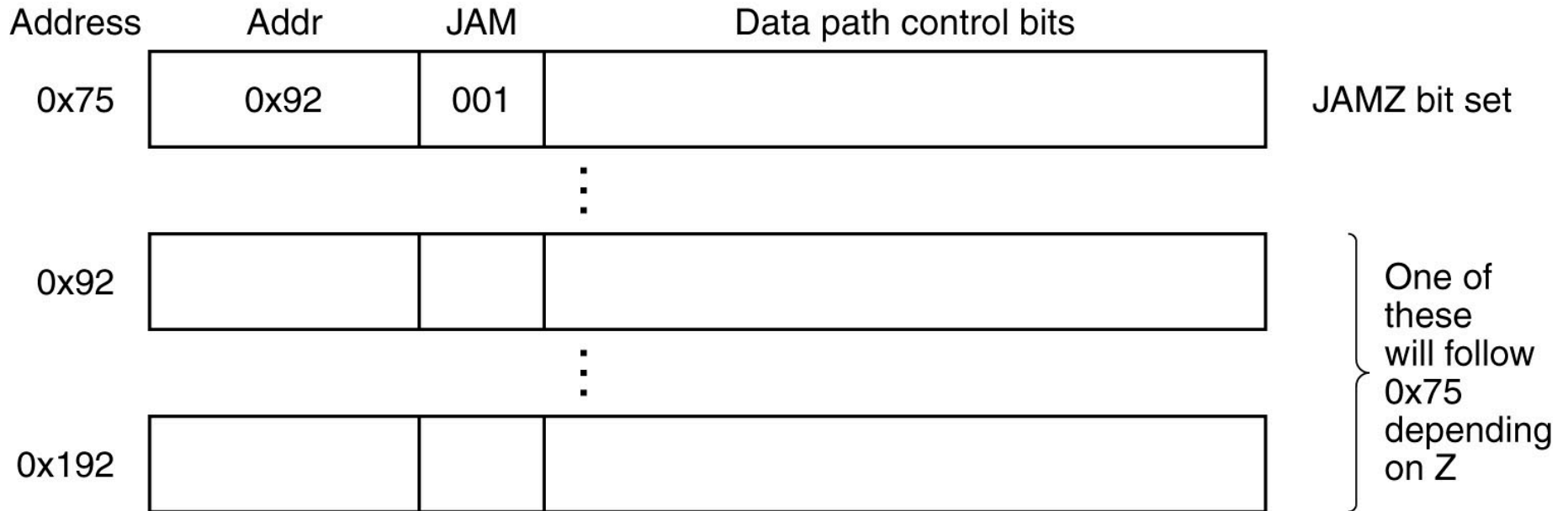
- Ciascuna  $\mu$ -istruzione indica sempre l'indirizzo della successiva (Addr)
- Notare: il default non è un'esecuzione sequenziale
- Il bit più alto di Addr (Addr[8]) è dato da:
  - (JAMZ AND Z) OR (JAMN AND N) OR Addr[8]
- Possibile realizzare salti condizionati

**ES**

- Addr = 0 1001 0010 (0x92)
- JAM [JAMPC,JAMN,JAMZ] = 001
- se Z=0 allora Addr = 0 1001 0010 (0x92)
- se Z=1 allora Addr = 1 1001 0010 (0x192)
- Se JMPC = 1 allora gli 8 bit bassi di Addr (tipicamente a 0) vanno in OR con il contenuto di MBR
- Possibile realizzare salti in tutto il Control Store

# Salti condizionati

Esempio di salto condizionato basato su Z



# Come migliorare le prestazioni

Migliorare le prestazioni significa massimizzare il rapporto:

$$\frac{\text{Velocità}}{\text{Prezzo}}$$

Da un punto di vista progettuale esistono tre approcci:

- Riduzione del numero di cicli necessari per eseguire una istruzione (introducendo hardware “dedicato”);
- *Aumento della frequenza del clock (semplificando l'organizzazione);*
- Introduzione del parallelismo (sovrapponendo l'esecuzione delle istruzioni).

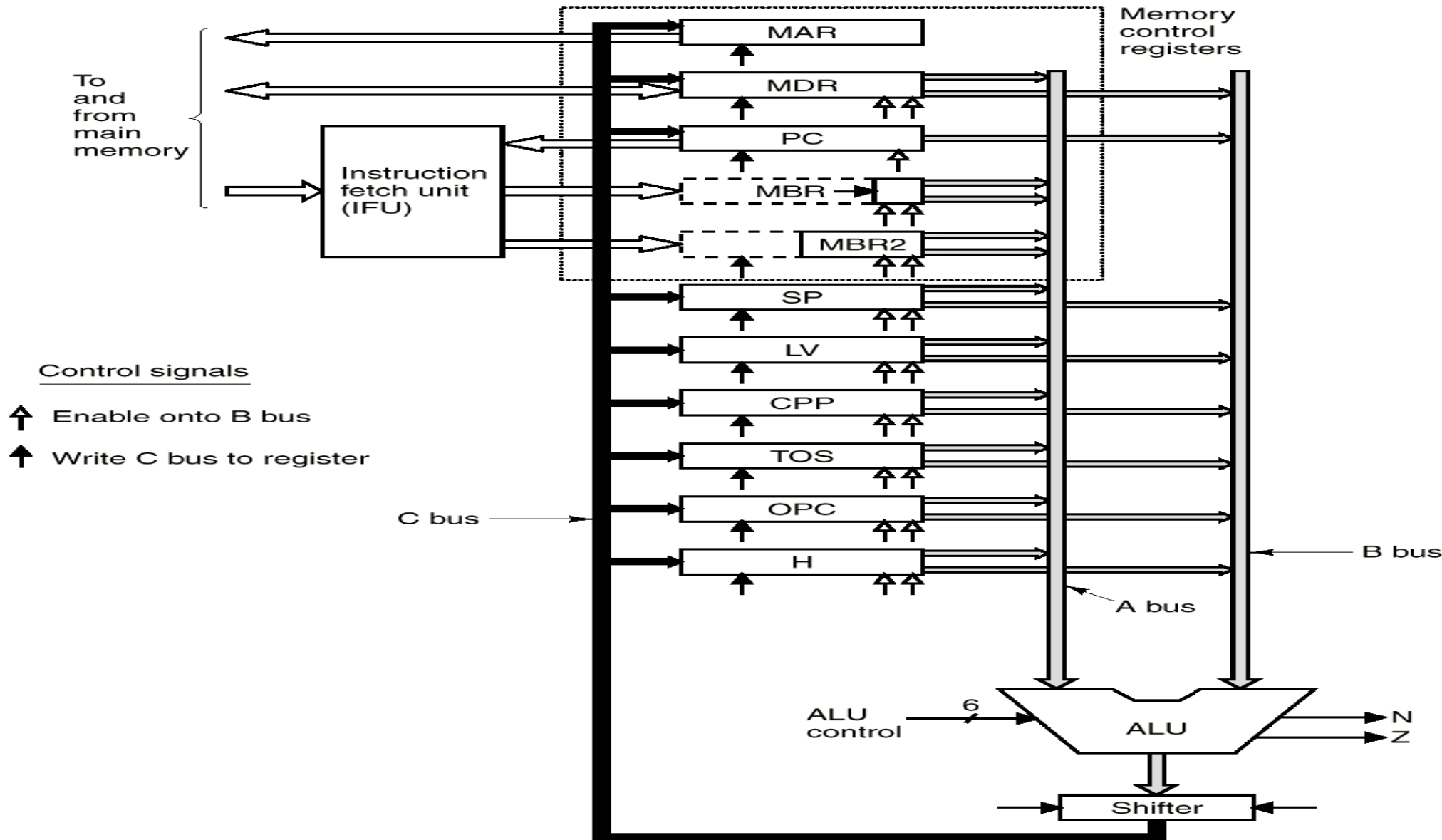


# Introduzione di componenti "dedicate"



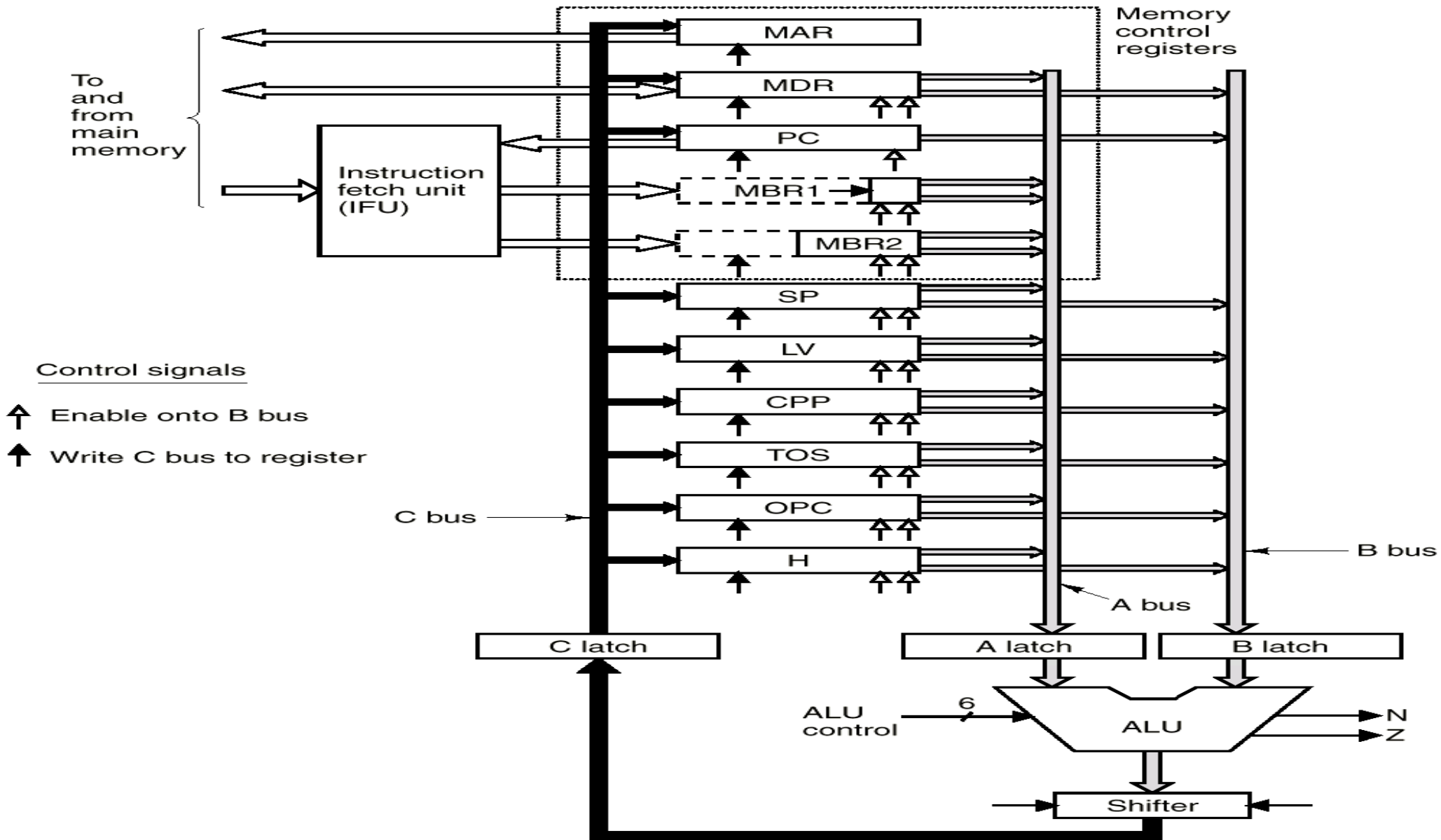


# Aumento del numero di Bus





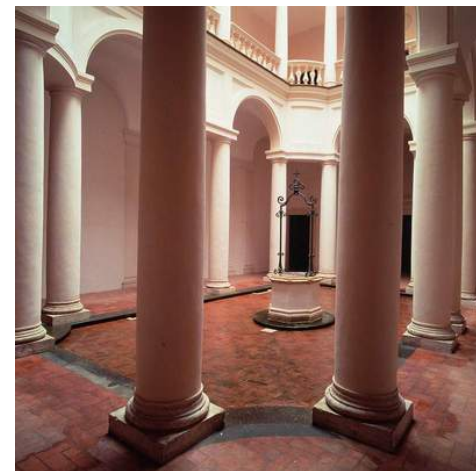
# Partizionamento del data-path



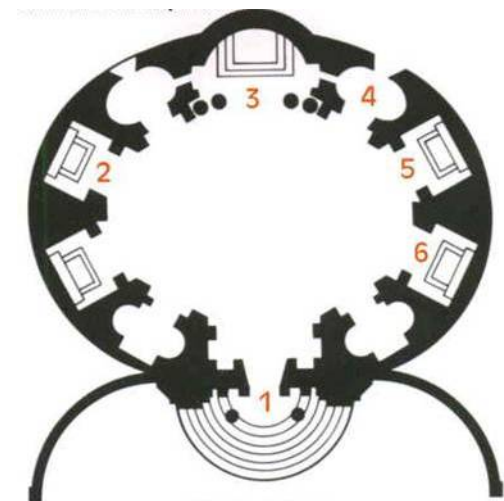




# Caching...

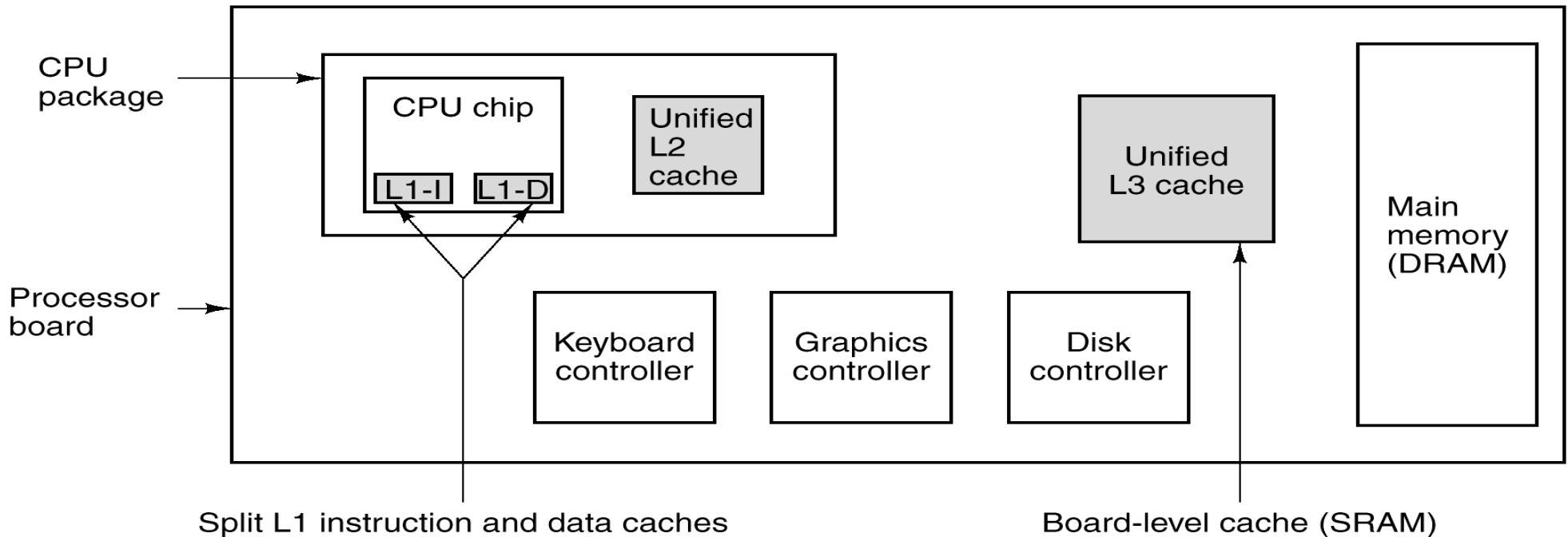


# Un'altra cache..





# Memorie Cache



- Scopo della cache: disaccoppiare le velocità di CPU e RAM
- Località spaziale: alta probabilità di accedere in tempi successivi a indirizzi molto vicini
- Località temporale: alta probabilità di accedere più volte agli stessi indirizzi in tempi molto vicini
- Gerarchie di cache: a 2 o 3 livelli
- Cache *inclusive*: ciascuna contiene quella del livello superiore

# Ci accorgiamo della presenza della cache?

Matrice 30.000x30.000, Intel Core i7 @ 3.6 GHz, 16GB RAM

```
int sum1(int** m, int n) {  
    int i, j, sum = 0;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            sum += m[i][j];  
    return sum;  
}
```

**1,84 secondi**

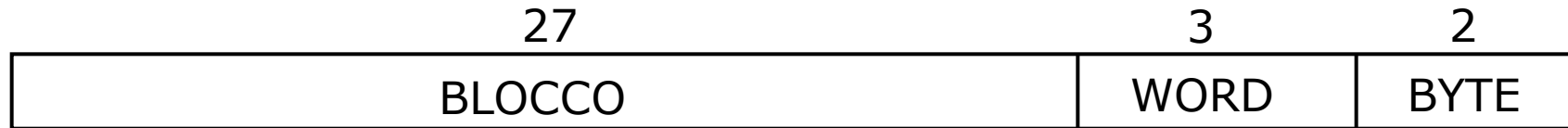
```
int sum1(int** m, int n) {  
    int i, j, sum = 0;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            sum += m[j][i];  
    return sum;  
}
```

**18,63 secondi**  
(circa 10 volte più lento)

# Organizzazione della Memoria in presenza di cache

- Lo spazio di memoria è organizzato in blocchi (da 4 a 64 byte), chiamati anche linee di cache
- Ciascuna linea contiene più word
- Ciascuna word contiene più byte
- Le cache sono organizzate in righe (o slot): ciascuna contiene una linea di cache, cioè un blocco di memoria
- Tutti i trasferimenti avvengono a livello di blocco
- Quando una word non viene trovata in cache, si trasferisce l'intera linea dalla memoria, o dalla cache di livello più basso

# Organizzazione della Memoria (esempio)



Struttura degli indirizzi

- Indirizzi a 32 bit (spazio di indirizzamento di  $2^{32}$  byte)
- Linee di cache (blocchi) di 32 byte
- Word di 4 byte
- Struttura dell'indirizzo:
  - I 27 bit più significativi rappresentano il numero di blocco
  - I successivi 3 bit il numero della word all'interno del blocco
  - Gli ultimi due bit il numero del byte all'interno della word

# Esempi di indirizzi

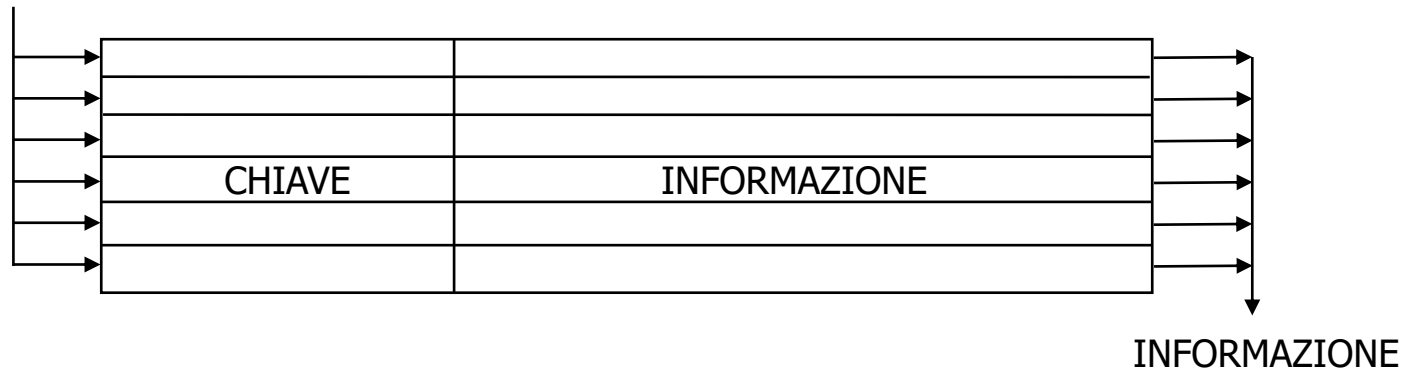
- 00000000000000000000000000000000 000 00
  - 1° blocco – 1° word – 1° byte
- 00000000000000000000000000000000 000 01
  - 1° blocco – 1° word – 2° byte
- 00000000000000000000000000000000 001 00
  - 1° blocco – 2° word – 1° byte (5° byte del blocco)
- 00000000000000000000000000000000 011 10
  - 1° blocco – 4° word – 3° byte (15° byte del blocco)
- 00000000000000000000000000000001 010 11
  - 2° blocco – 3° word – 4° byte
- 00000000000000000000000000000101 110 10
  - 6° blocco – 7° word – 3° byte
- 000000000000000000000000010110 101 00
  - 23° blocco – 6° word – 1° byte

# Ricerca di un blocco in cache

- Una cache contiene un sottoinsieme di blocchi di memoria di indirizzo non contiguo
- Quando la CPU cerca una word, non sa in quale posizione essa si possa trovare nella cache (se effettivamente c'è)
- Non c'è modo di risalire dall'indirizzo di un blocco di memoria alla sua posizione in cache
- Non è possibile utilizzare il normale meccanismo di indirizzamento delle RAM:
  - Si fornisce un indirizzo
  - Viene letto il dato che si trova allo indirizzo specificato
- Si usano allora Memorie Associative

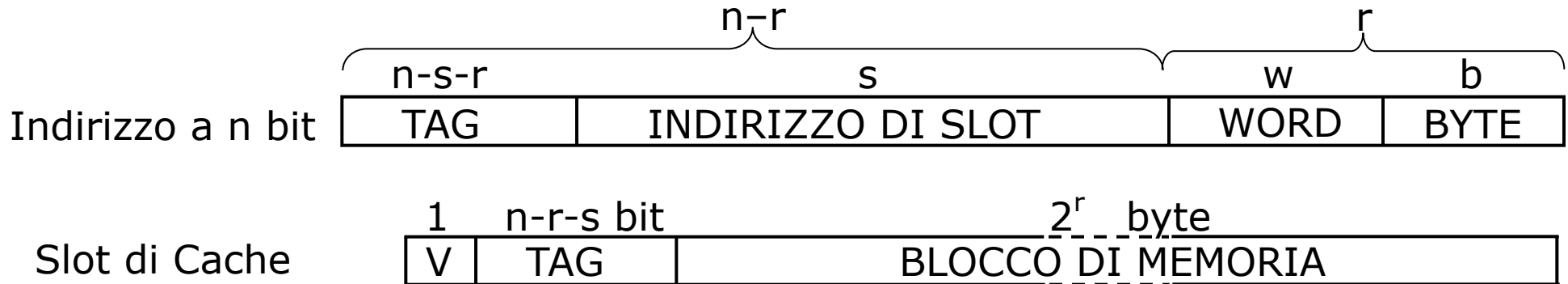
# Memorie Associative

CHIAVE DELL'INFORMAZIONE



- Ciascun elemento (riga) è costituito da due parti: la *chiave* e il *dato*
- L'accesso ad un elemento viene effettuato non solo in base all'indirizzo ma anche in base a parte del suo contenuto (*chiave*)
- L'accesso *associativo* avviene in un unico ciclo
- Nel caso di una cache:
  - Un elemento viene chiamato *slot* di cache
  - La chiave viene chiamata *tag* (etichetta)
  - L'informazione è una *cache line* (o blocco)

# Cache a Mappatura Diretta

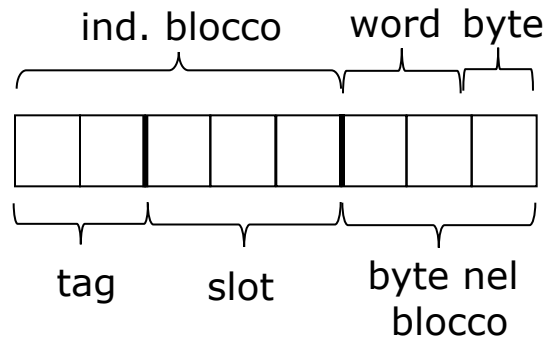


- Spazio di memoria di  $2^n$  byte, diviso in blocchi da  $2^r$  byte
- Gli  $n-r$  bit più significativi dell'indirizzo specificano il blocco
- In una cache con  $2^s$  slot si associa ad ogni blocco la slot di cache indicata dagli  $s$  bit meno significativi del suo indirizzo
- Se il blocco è in cache deve essere in quella slot, e lì bisogna cercarlo
- Il TAG sono gli  $n-s-r$  bit più significativi dell'indirizzo
- Il TAG è contenuto nella slot
- Il TAG permette di distinguere tra tutti i blocchi che condividono la stessa slot (collidono)

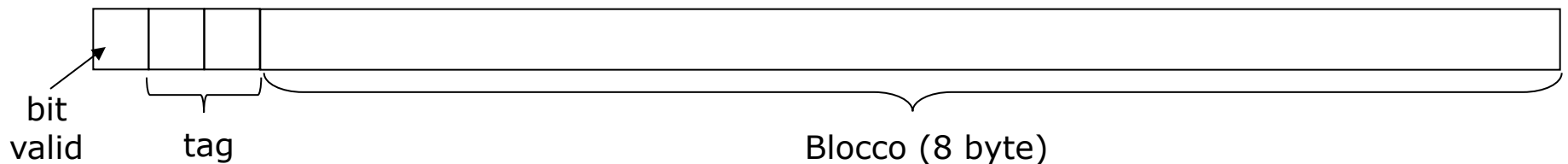


# Esempio

- Indirizzi a 8 bit ( $n = 8$ )
- Linee di cache a 8 byte ( $r = 3$ )
- Word di 2 byte
- Cache di 8 slot
- Struttura indirizzo:



- Struttura slot:



# Esempio – continua (è un’animazione che si può scaricare dal sito)

|           |   |
|-----------|---|
| 11111111  |   |
| 11111110  |   |
| .         |   |
| .         |   |
| .         |   |
| .         |   |
| 00001 111 | r |
| 00001 110 | q |
| 00001 101 | p |
| 00001 100 | o |
| 00001 011 | n |
| 00001 010 | m |
| 00001 001 | l |
| 00001 000 | i |
| 00000 111 | h |
| 00000 110 | g |
| 00000 101 | f |
| 00000 100 | e |
| 00000 011 | d |
| 00000 010 | c |
| 00000 001 | b |
| 00000 000 | a |

Memoria

|     |   |   |   |   |    |    |   |   |   |    |    |  |  |
|-----|---|---|---|---|----|----|---|---|---|----|----|--|--|
| 111 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 110 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 101 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 100 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 011 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 010 | 0 | 0 | 0 |   |    |    |   |   |   |    |    |  |  |
| 001 | 0 | 0 | 0 | w | lx | ny | α | δ | ϕ | θ% | r& |  |  |
| 000 | 0 | 0 | 0 | a | b  | c  | d | e | f | g  | h  |  |  |

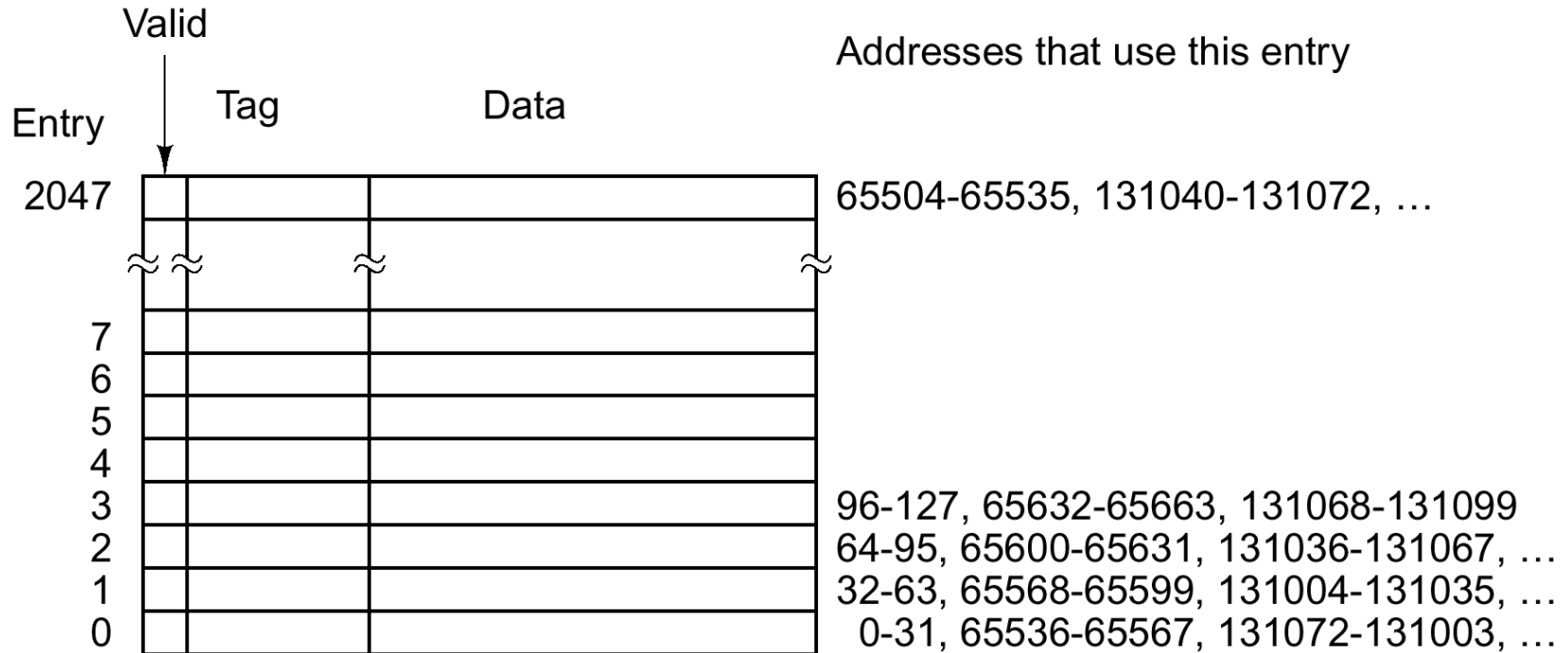
Bit valid      Tag

Cache

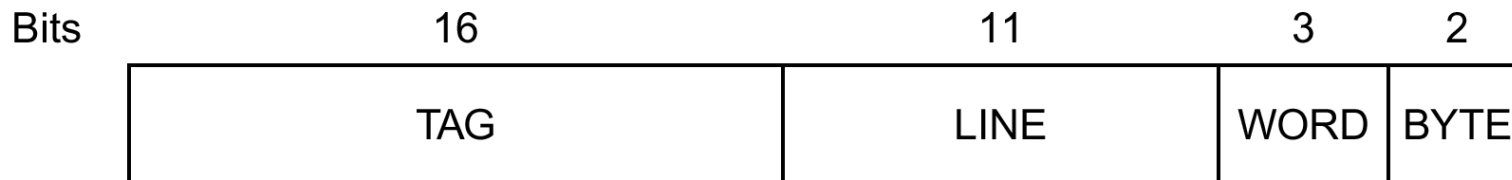
Accessi:

0000 01001 101  
 0000 10001 1001  
 0000 00010 011  
 0100 10010 110

# Cache a Mappa Diretta (esempio)

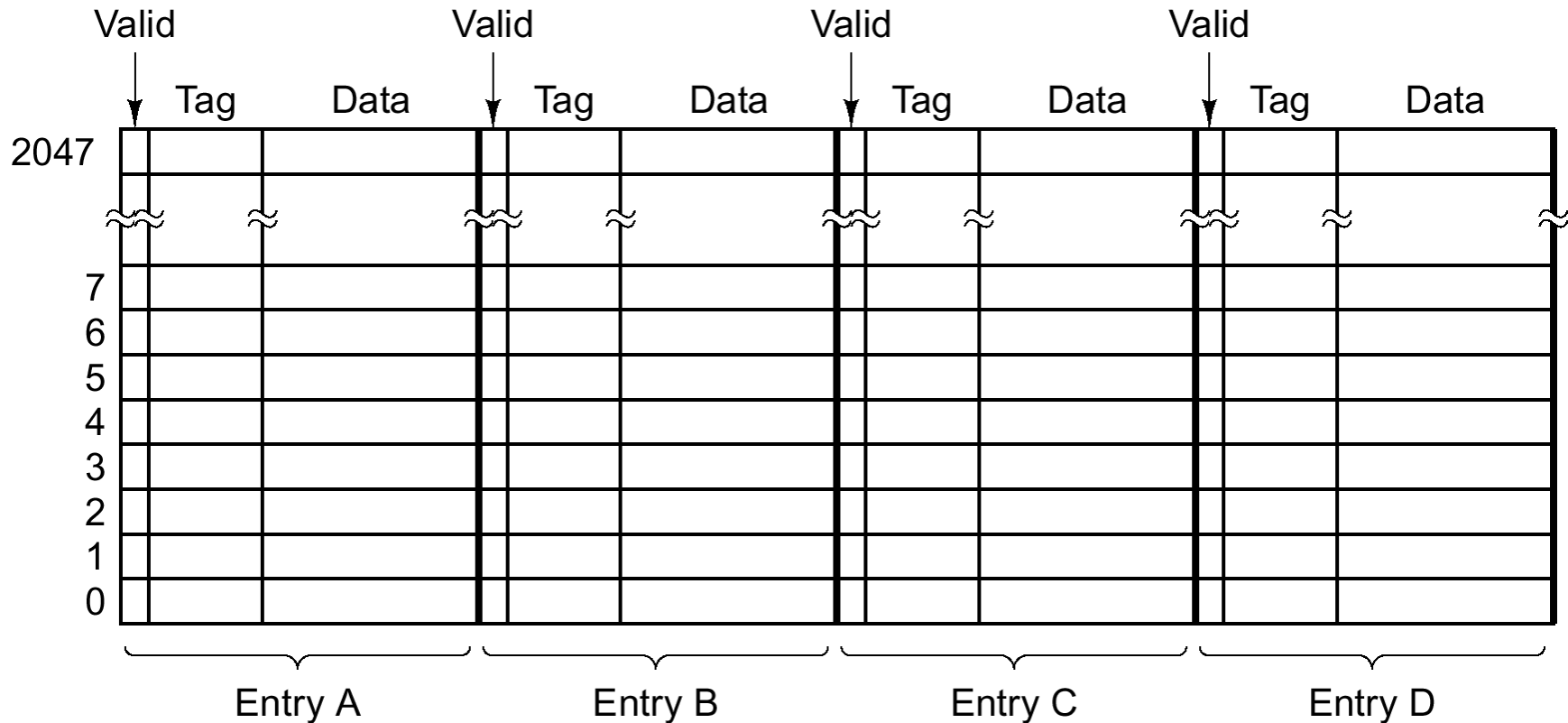


(a)



(b)

# Cache Associative ad Insiemi



- Ogni slot è costituita da  $n$  elementi, ciascuno composto di bit valid, tag e blocco
- Un blocco può stare in un elemento qualsiasi della slot che gli corrisponde
- Attenua il problema della collisione di più blocchi sulla stessa slot

# Gestione della Cache

- La CPU deduce numero di slot e TAG del blocco a partire dall'indirizzo
- Viene fatto un ciclo e confrontato il TAG nella slot con quello del blocco
- Cache Hit in lettura: tutto ok
- Cache Hit in scrittura:
  - *write through*: scrive anche in memoria
  - *write back*: unica scrittura finale, quando il blocco è rimosso dalla cache
- Cache Miss in lettura: il blocco viene trasferito in cache e sovrascrive quello presente (questo va copiato se modificato)
- Cache Miss in scrittura:
  - *write allocation*: porta il blocco in cache (conviene per scritture ripetute)
  - *write to memory*: si effettua la scrittura in memoria

# Esercizio su memorie cache I

Una cache a mappa diretta con 16K slot e cache line di 64 byte, è installata in un sistema con indirizzi a 32 bit:

- specificare la struttura di ciascuna slot, indicando esplicitamente la dimensione complessiva della slot e quella di ciascun campo;
- calcolare il numero di slot e la posizione nella slot del byte con indirizzo esadecimale 7B80034A;
- verificare se i due byte di indirizzo esadecimale 32353793 e 3F5537BC collidono sulla stessa slot.

## Esercizio su memorie cache II

Si consideri una memoria cache associativa a 4 vie composta da 4K slot in un sistema con indirizzi a 24 bit e cache line da 16 byte. Indicando con X la cifra meno significativa non nulla del proprio numero di matricola, specificare:

- la struttura dell'indirizzo di memoria, specificando la dimensione dei vari campi in bit;
- la struttura della slot di cache, specificando la dimensione dei vari campi in bit;
- la dimensione totale della cache (ordine di grandezza decimale);
- i passi necessari alla ricerca nella cache del byte di indirizzo BXAXF2.

## Memoria cache III

Si vuole progettare una cache a mappatura diretta per un sistema con indirizzi a 32 bit e linee di cache di 32 byte.

Calcolare:

- il numero minimo di slot necessario a garantire che non più di  $2^{13}$  blocchi collidano sulla stessa slot;
- la relativa struttura dell'indirizzo di memoria e della slot di cache, specificando la dimensione dei campi in bit;
- quanto varia il numero di slot necessari nel caso di cache associativa a due vie;
- i passi necessari alla scrittura del byte di indirizzo 7CA3F37D con riferimento a situazioni di cache hit e cache miss.



## Memoria cache IV

Si vuole progettare una cache unificata a mappatura diretta per una CPU con indirizzi a 32 bit e linee di cache di 32 byte. Supponendo di avere a disposizione una memoria di 4MB e 40KB di spazio disponibile massimo sul chip della CPU determinare:

- la struttura di una possibile slot di cache che soddisfi questi requisiti e la relativa struttura dell'indirizzo di memoria;
- le dimensioni totali della cache progettata;
- se e come sia possibile modificare la struttura determinata al punto A per ridurre le collisioni sulle slot di cache;
- cosa può succedere se la CPU vuole leggere il byte 260 della memoria principale.

# Domande cache

Con riferimento ad una cache a mappatura diretta con 16K slot e cache line di 64 byte installata in un'architettura a 32 bit, indicare se le seguenti affermazioni sono vere o false.

- Il campo TAG della cache è di 14 bit.
- I primi 6 bit dell'indirizzo non vengono usati per indirizzare una slot di cache.
- Il numero di collisioni su una slot di cache aumenta se aumentiamo le dimensioni della cache fino a 32K.
- I byte di indirizzo F4B6A598 e 3CE6A5B3 collidono sulla stessa slot della cache.
- I byte di indirizzo 4F3B7318 e 4F3B733A collidono sulla stessa slot della cache.
- Una slot della cache è grande 525 bit.
- Su una slot della cache collidono 4K cache line di memoria.
- L'accesso a un byte di memoria contiguo a un byte presente nella cache non genera mai cache miss.

## Soluzioni esercizio precedente

Con riferimento ad una cache a mappatura diretta con 16K slot e cache line di 64 byte installata in un'architettura a 32 bit, indicare se le seguenti affermazioni sono vere o false.

- @NO Il campo TAG della cache è di 14 bit.
- @SI I primi 6 bit dell'indirizzo non vengono usati per indirizzare una slot di cache.
- @NO Il numero di collisioni su una slot di cache aumenta se aumentiamo le dimensioni della cache fino a 32K.
- @SI I byte di indirizzo F4B6A598 e 3CE6A5B3 collidono sulla stessa slot della cache.
- @NO I byte di indirizzo 4F3B7318 e 4F3B733A collidono sulla stessa slot della cache.
- @SI Una slot della cache è grande 525 bit.
- @SI Su una slot della cache collidono 4K cache line di memoria.
- @NO L'accesso a un byte di memoria contiguo a un byte presente nella cache non genera mai cache miss.

# CPU Core i7

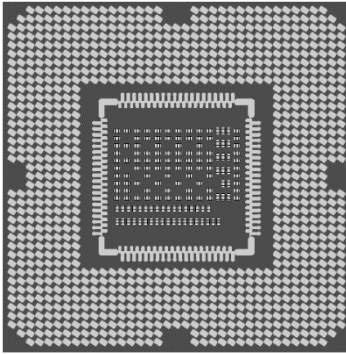
## Esternamente:

- Macchina CISC tradizionale
  - Operazioni su interi a 8/16/32 bit
  - Operazioni FP a 32/64 bit (IEEE 754)
- Set di istruzioni esteso e molto disordinato
  - Lunghezza variabile da 1 a 17 bytes
- 8 registri

## Internamente:

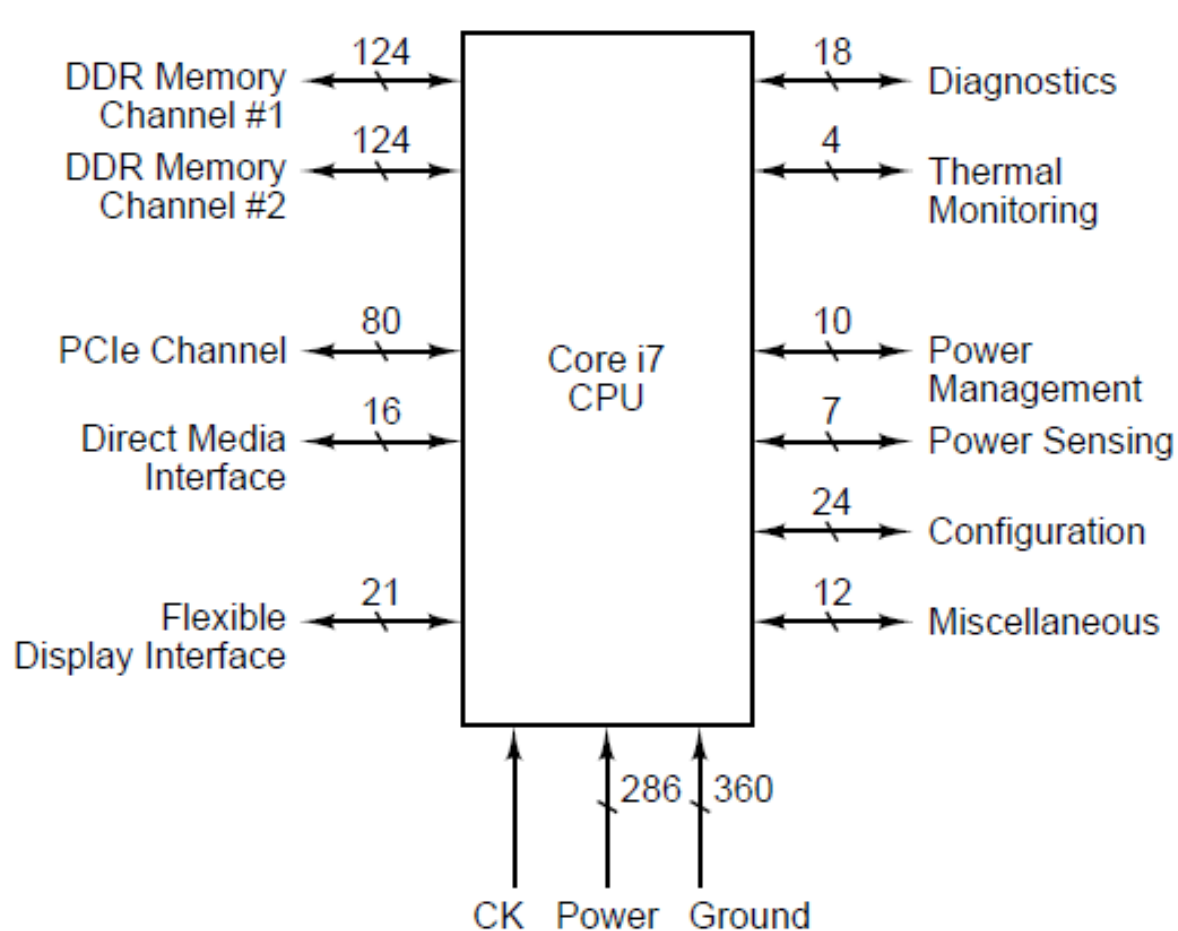
- Architettura "Sandy Bridge" (32 nm)
  - Successivi:
    - Ivy Bridge e Haswell (22nm)
    - Broadwell, Skylake, Kaby Lake e Coffee Lake (14nm)
    - Ice/Comet Lake (10nm) (10th generation)
- Nucleo RISC
- Lunga pipeline
- Multi-core (4/6)

# Intel Core i7



- Nahalem (2008): 731 milioni di transistor, 45 nm, 3.2 Ghz
- Sandy Bridge (2011): 1,16 miliardi di transistor, 32nm, 3.5 Ghz
- Architettura a 64 bit compatibile con i predecessori
- Aritmetica Floating-point IEEE 754
- Architettura multicore (2-6)
- Hyperthreaded, superscalare (fattore 4), pipelined
- Bus di memoria sincrono a 64 bit + Bus PCIe
- QPI (Quick Path Interconnect): comunicazione con altri processori
- Cache 1° livello 32KB dati + 32KB istruzioni
- Cache 2° livello 256 KB per core (snooping)
- Cache 3° livello condivisa da 4 a 15 MB
- Scheda con 1155 pin (diversa dai predecessori)
- Consuma da 17 a 150W (stati differenti per ridurre il consumo)

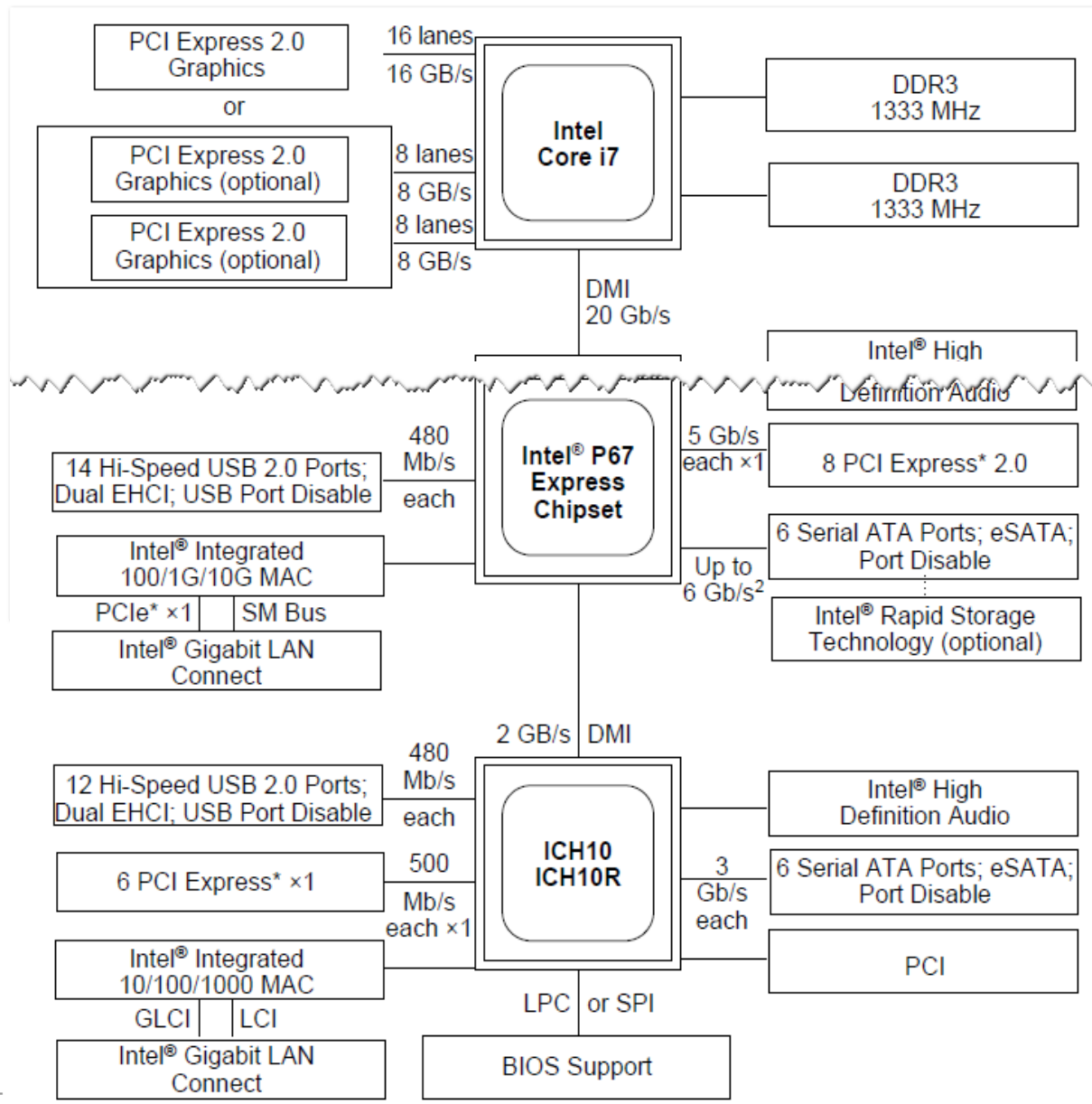
# Intel Core i7: Pinout Logico



# Intel Core i7: Pinout Logico (2)

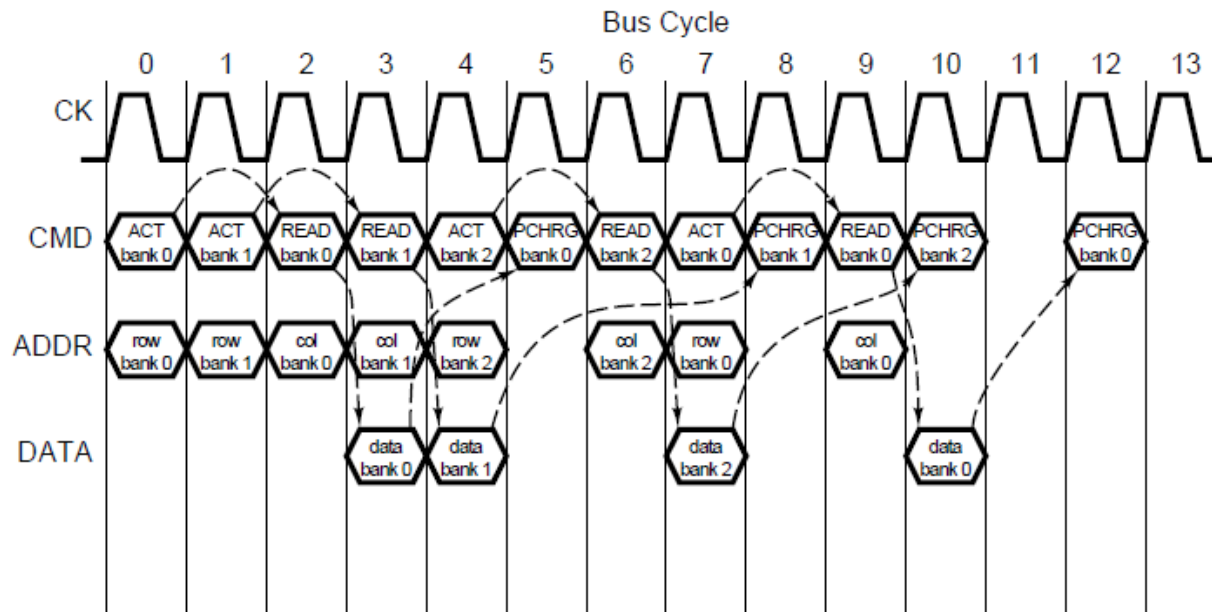
- 1155 piedini
  - 447 per i segnali (alcuni duplicati, 131 in tutto)
  - 286 connessioni di alimentazione
  - 360 connessioni di massa
  - 62 per "uso futuro"
- Due gruppi indipendenti per l'interfaccia con una DRAM
  - 64bit, 666Mhz, 1,333 MTPS, 20 GB/sec complessivi
- Un gruppo per l'interfaccia con linee PCIe
  - 16 linee (lane), 16GB/sec complessivi
- Un gruppo per la comunicazione con il chipset (DMI)
  - 4 linee, 2.5GB/sec complessivi
  - P67: SATA, USB, Audio, PCIe, Flash;
  - ICH10: PCI, 8259A, clock, timer, controllori DMA
- Gestione delle interruzioni sia come l'8088 che con APIC (Advanced Programmable Interrupt Controller)
- Gestione della tensione: (possibili diversi valori di Voltaggio)
- Thermal monitoring: sensori di calore per il "thermal throttling"
- 11 linee di diagnosi secondo lo standard IEEE 1149.1 JTAG

# Struttura di un sistema moderno basato su i7



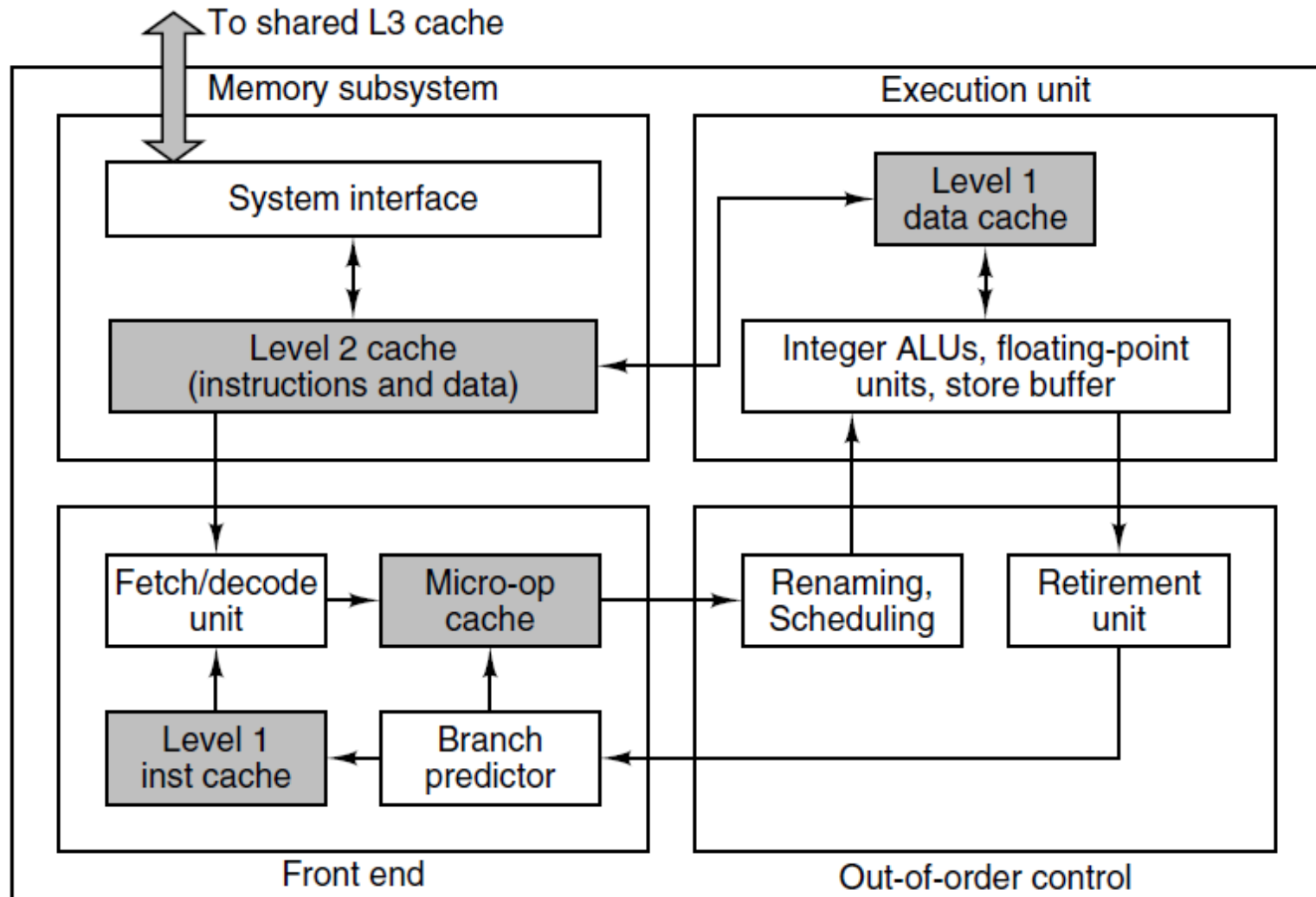


# Intel Core i7: Memory Bus



- Bus gestito con pipelining: è possibile sovrapporre 4 transazioni
- Ogni interfaccia DRAM ha 3 gruppi di linee
- Fasi di una transazione (usano gruppi di linee indipendenti):
  - Attivazione e invio indirizzi
  - Comando di Read/Write di parole contigue della memoria (bank)
    - Richiede due passi: comando e trasferimento dati
  - Chiusura e preparazione per la prossima transazione
- Funziona solo con memorie sincrone

# Microarchitettura Sandy Bridge di un core i7



# Microarchitettura Sandy Bridge i7

## Sottosistema di memoria:

- Contiene una cache L2 unificata
  - 8-way, 256KB, cache line 64B, write-back
- Interfaccia verso L3 condivisa che contiene una unità di prefetching
  - 12-way, da 8 a 20MB, cache line 64B, interfaccia con RAM

## Front end:

- Preleva istruzioni dalla cache L2 e le decodifica
- Istruzioni in L1 (8-way, 64KB, cache line 64B)
- Scompone istruzioni in micro-op RISC e le appoggia in una cache L0

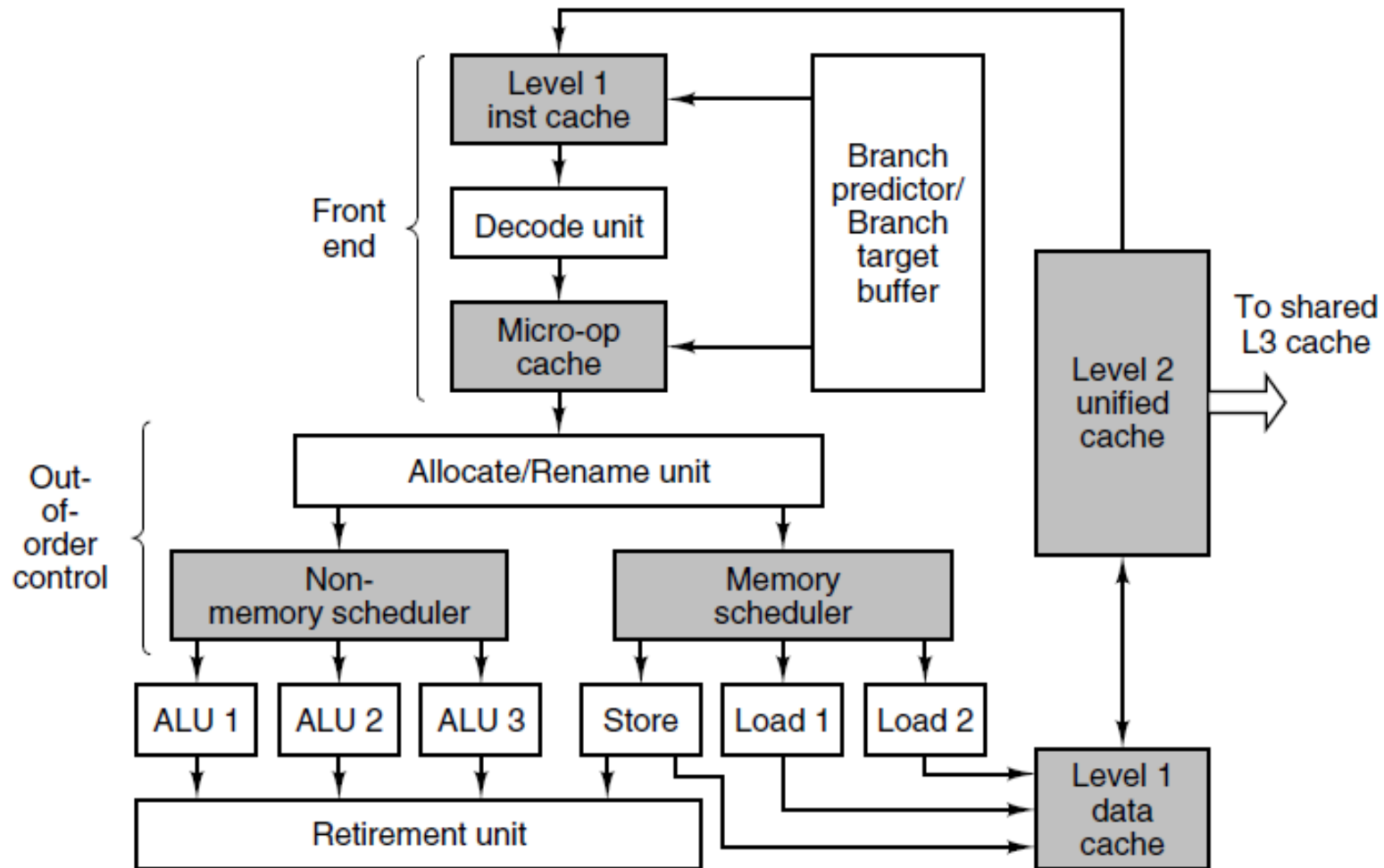
## Controllo dell'esecuzione:

- Sceglie le microistruzioni che possono andare in esecuzione
- Ritira in ordine le microistruzioni

## Unità di esecuzione:

- Esegue le microistruzioni su unità funzionali multiple
- Accede a dati nei registri e nella cache dati L1
- Invia informazioni al perditore di salti

# Pipeline dell'architettura Sandy Bridge



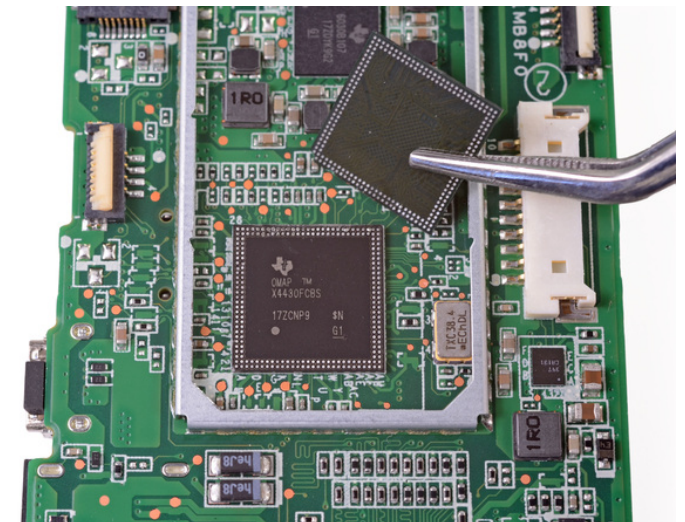
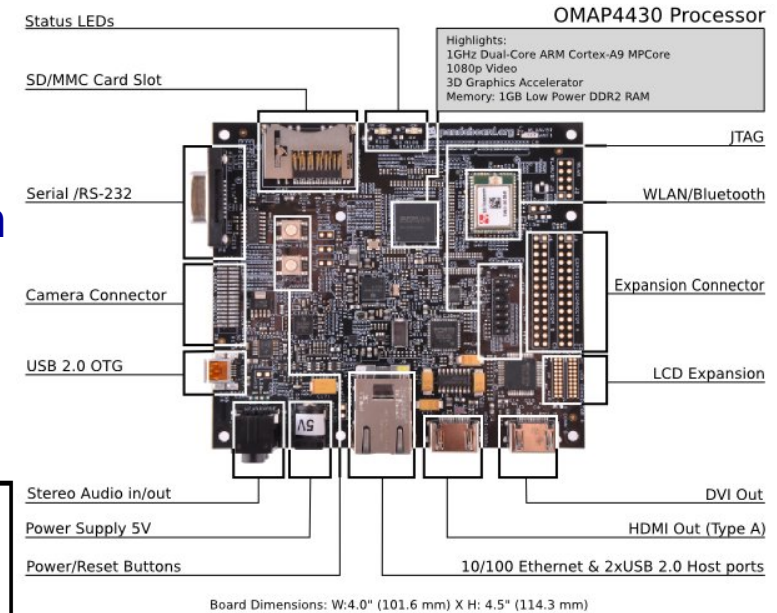
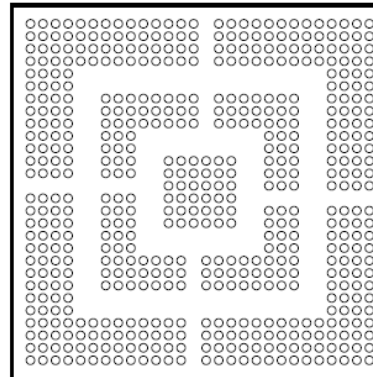
# CPU OMAP4430

SOC con 2 microprocessori **ARM Cortex A9**

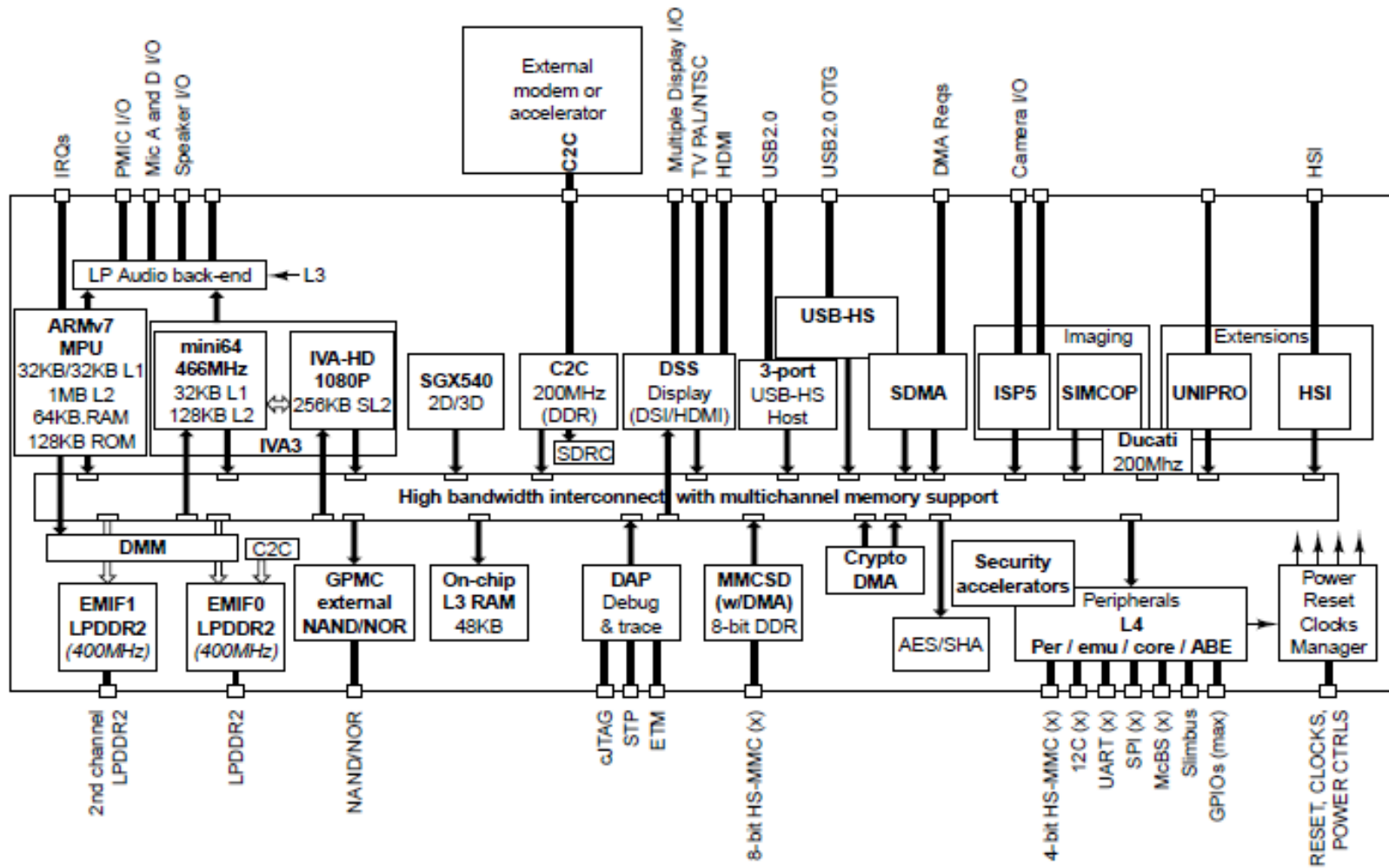
- Implementazione Texas Instruments dell'architettura ARM
- A 32 bit, bus di memoria a 32 bit
- RISC pura
- 2 livelli di cache
- Set di istruzioni ridotto e ordinato
  - Lunghezza fissa (4 bytes)
  - Hardware dedicato per istruzioni multimediali
- 16 registri generali
- 32 registri opzionali per operazioni in virgola mobile
- Organizzazione piuttosto semplice
- Multicore (fino a 4 core)
- Pipeline a 11 stadi

# OMAP4430 della Texas Instruments

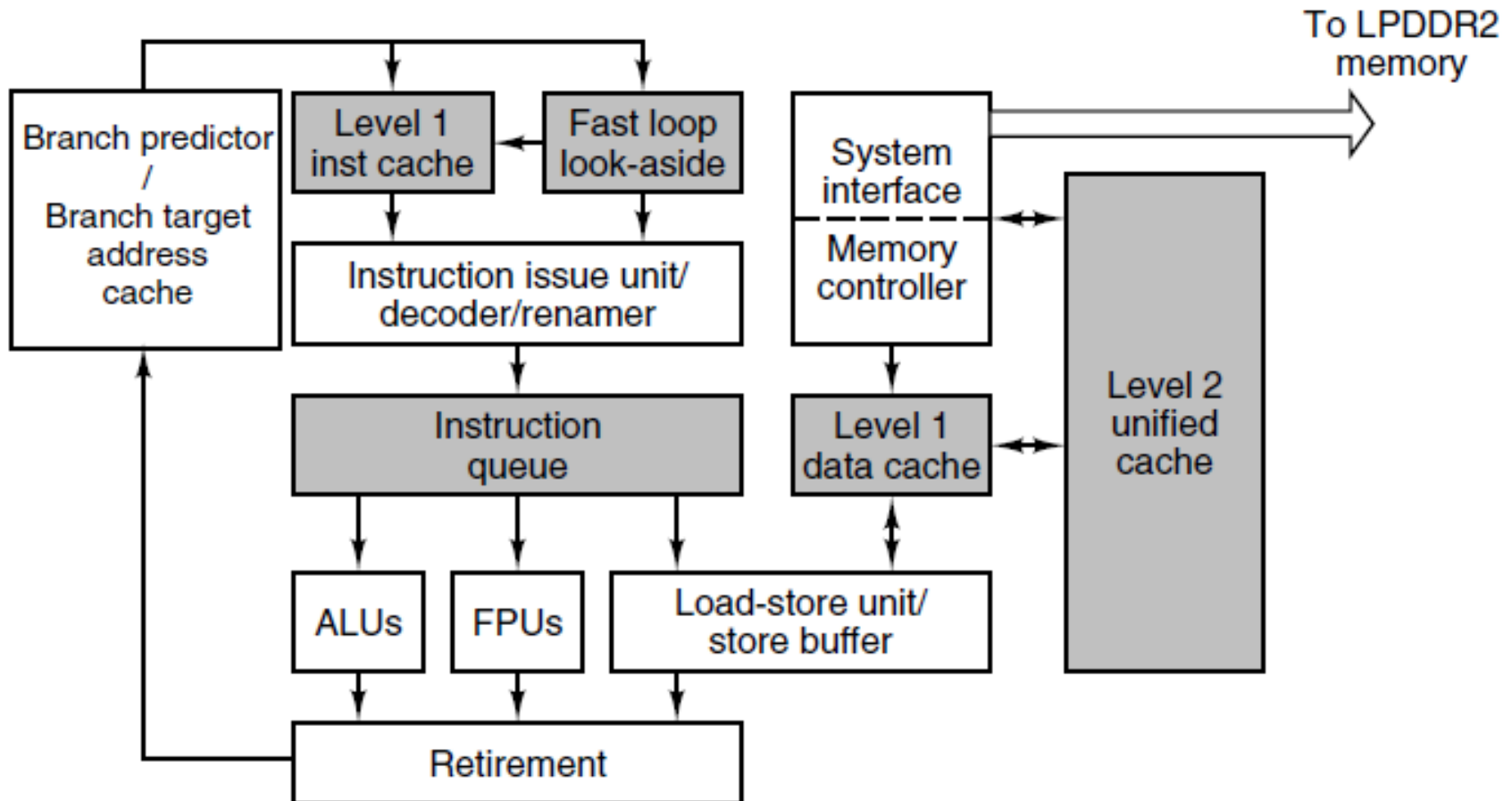
- SoC basato su ISA ARM
- Target: sistemi mobile o embedded
- Equipaggiamento:
  - 2 CPU ARM RISC Cortex-A9 a 1Ghz, 45nm
  - 1 GPU POWERV SGX540 (rendering 3D)
  - 1 ISP (manipolazione immagini)
  - 1 VPU IVA3 (video enc/dec)
- Interfacce I/O:
  - Touchscreen, keypad
  - DRAM, Flash
  - USB, HDMI
- Basso consumo di potenza
  - 660mW/100μW
  - dynamic voltage scaling
  - power gating
- Superscalare (2 istr+branch pred.+OoO)
- 2 L1: 32KB+32KB, 1 L2: 1MB
- Interfaccia DRAM (LPDDR2)
- Scheda a 547 pin



# Architettura OMAP4430



# Microarchitettura ARM Cortex A9

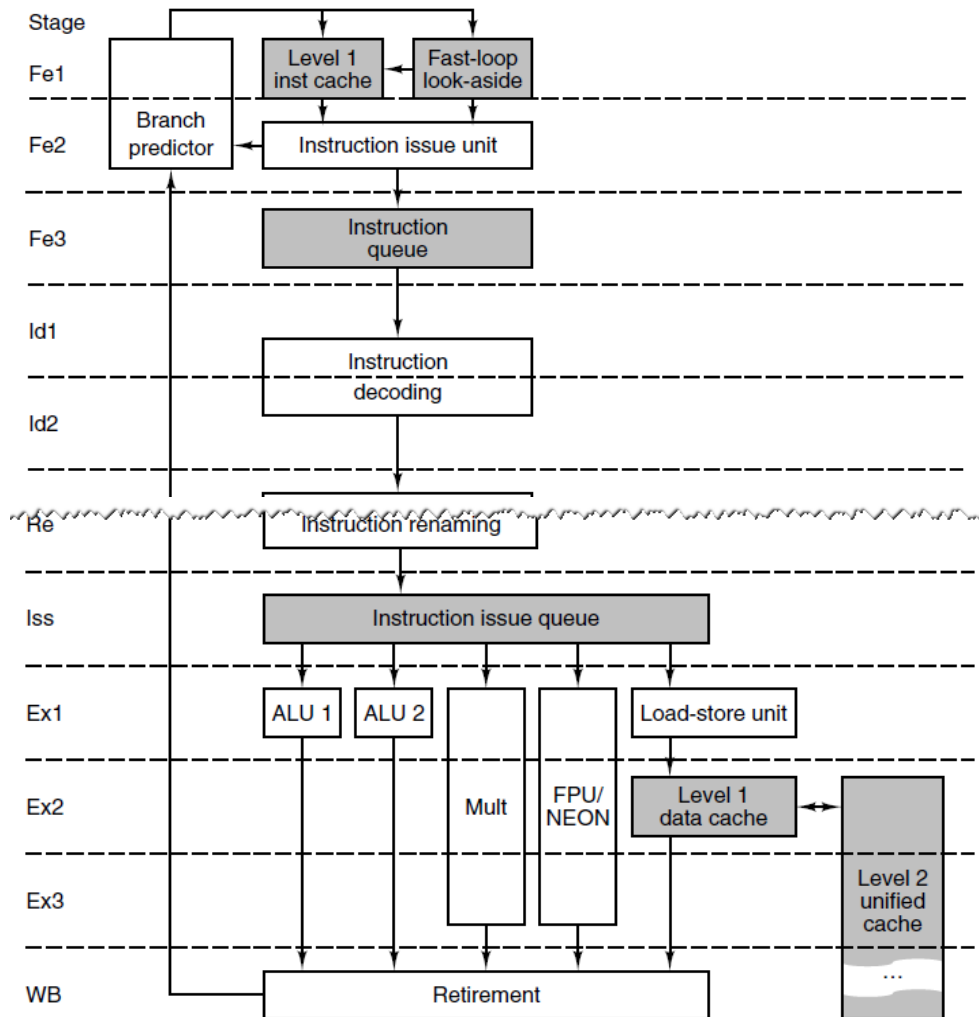




# Microarchitettura ARM Cortex A9

- Cache L2:
  - Unificata da 1MB
- I-cache L1:
  - 32KB 4-way
  - cache line di 32 byte: 8K istruzioni in tutto
- Unità di lancio:
  - prepara fino a 4 istruzioni per ciclo e le mette in un buffer
- Unità di esecuzione
  - Decodifica e sequenzia le istruzioni
  - Produce una coda di istruzioni da eseguire
- Almeno 2 Unità di esecuzione
  - 1 con 2 I-ALU per somme + 1 I-ALU per prodotti con registri dedicati
  - 1 di load/store con:
    - 4-way D-L1 da 32KB con line di cache a 32B
    - Prefetching di dati
  - 1 FP-ALU (VFP) e 1 SIMD vettoriale (NEON) opzionali
- Interfaccia con la memoria:
  - Architettura a 32 bit, word da 4B
  - 4GB di memoria indirizzabile su 2 canali indipendenti (8GB totali)

# Pipeline ARM Cortex A9



# CPU ATmega168

Chip semplificato con  $<1M$  transistor

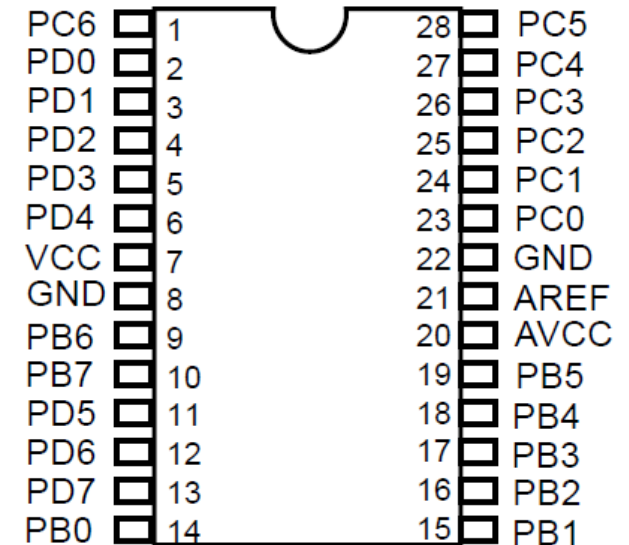
- Economicità prevale sulle prestazioni
- Macchina RISC a 8 bit
- 32 registri eterogenei
- Istruzioni eseguite in un ciclo
- Pipeline a due stadi: fetch+esecuzione

Internamente:

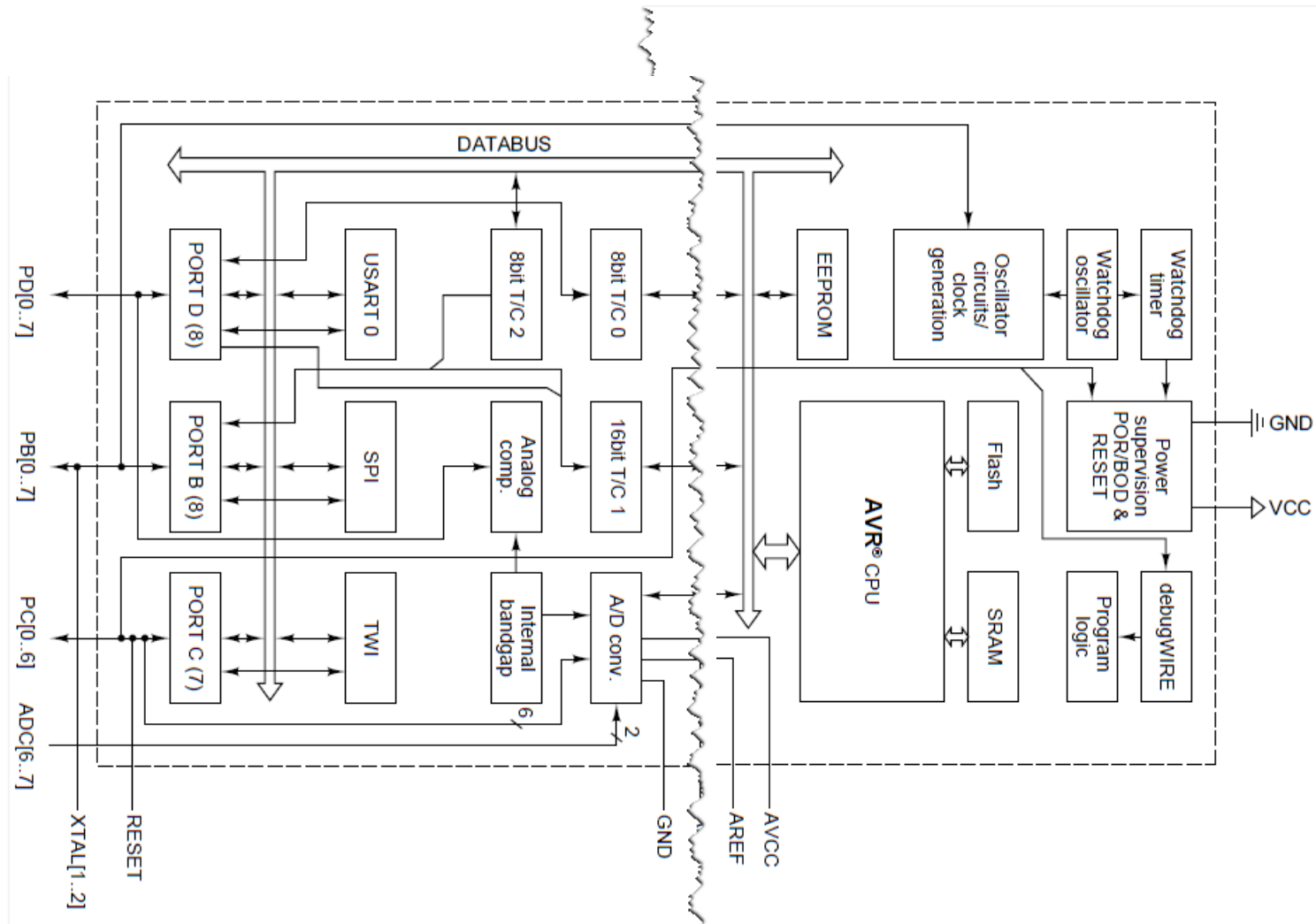
- Organizzazione semplice
- Basata su un Bus principale
- 1 SRAM da 1KB per i dati volatili
- 1 EEPROM da 1KB per dati statici
- Esecuzioni e ritiri in ordine

# Atmel ATmega168

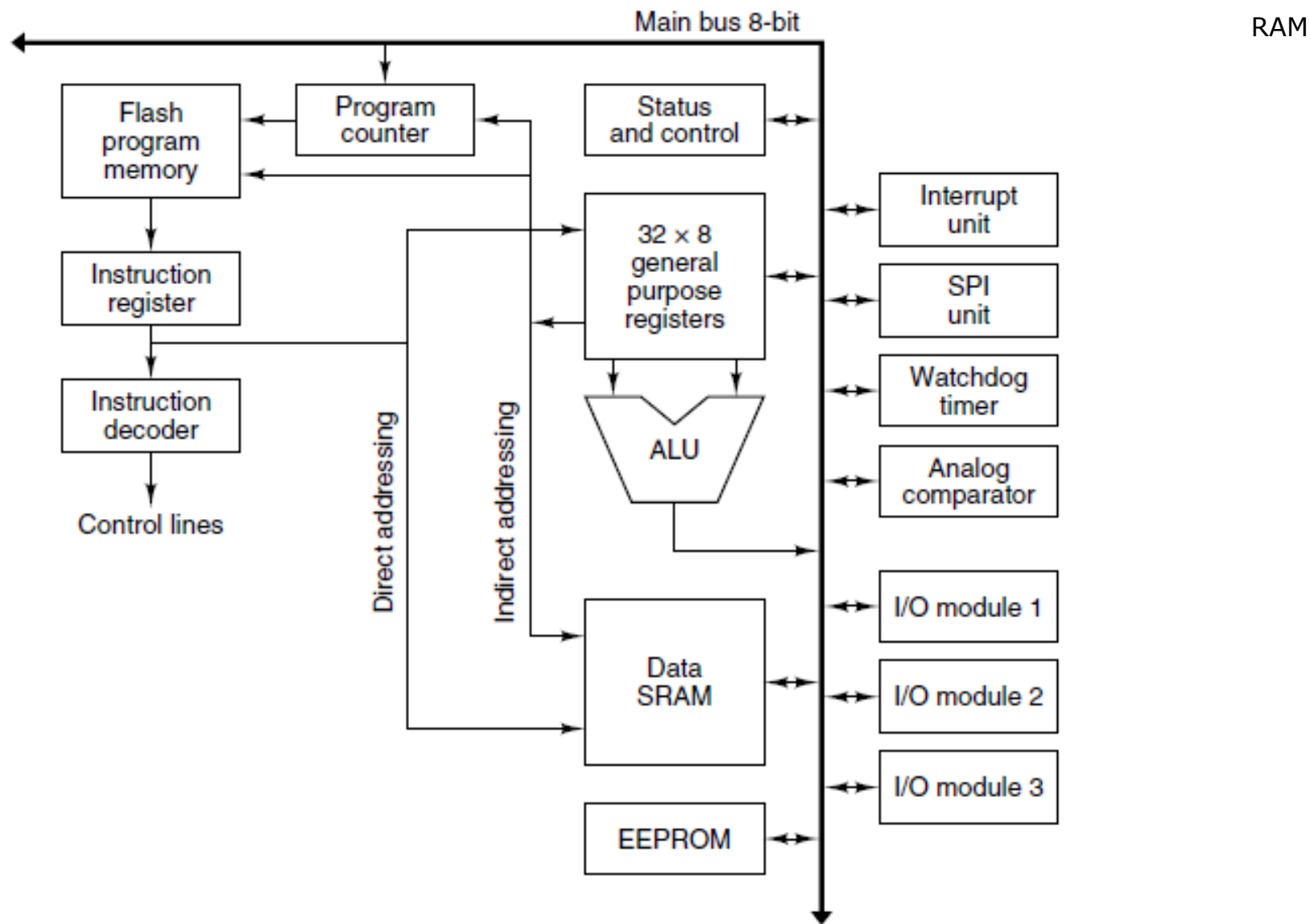
- Microcontrollore per applicazioni embedded (~1\$)
- CPU a 8 bit basata su ISA AVR
- Scheda a 28 pin
  - 23 porte di I/O
    - 8 per porte B e D
    - 7 per porta C (analogica)
  - 1 Vcc+2GND
  - 2 per configurare circuiti analogici
- Memorie incorporate
  - 16KB Flash
  - 1KB EEPROM
  - 1KB SRAM
- No RAM esterna
- Clock real time
- Interfaccia seriale



# Architettura ATmega168



# Microarchitettura ATmega168



RAM

# Microarchitettura ATmega168

- Registri collegati al Bus principale a 8 bit
  - Register file: contiene 32 registri a 8 bit per dati temporanei
  - Status e control: registro di stato
  - Program counter: indirizzo istruzione da eseguire
  - Registro delle istruzioni: istruzione corrente
- Ciclo macchina attraverso il main bus
- Indirizzamento a memoria
  - dati: 2 registri (64KB max)
  - istruzioni: 3 registri (16MB max)
- Unità di controllo delle interruzioni
- Interfaccia seriale
- Timer
- Comparatore analogico
- 3 porte digitali di I/O (fino a 24 dispositivi)

# Esecuzione di una istruzione nell'ATmega168

- Semplice pipeline
- Due stadi
  1. Fetch dell'istruzione nel registro delle istruzioni
  2. Esecuzione dell'istruzione:
    - a) Lettura dei registri sorgente
    - b) Elaborazione della ALU
    - c) Memorizzazione del risultato nel registro target
- Tutto in 2 cicli di clock a 10-20Mhz

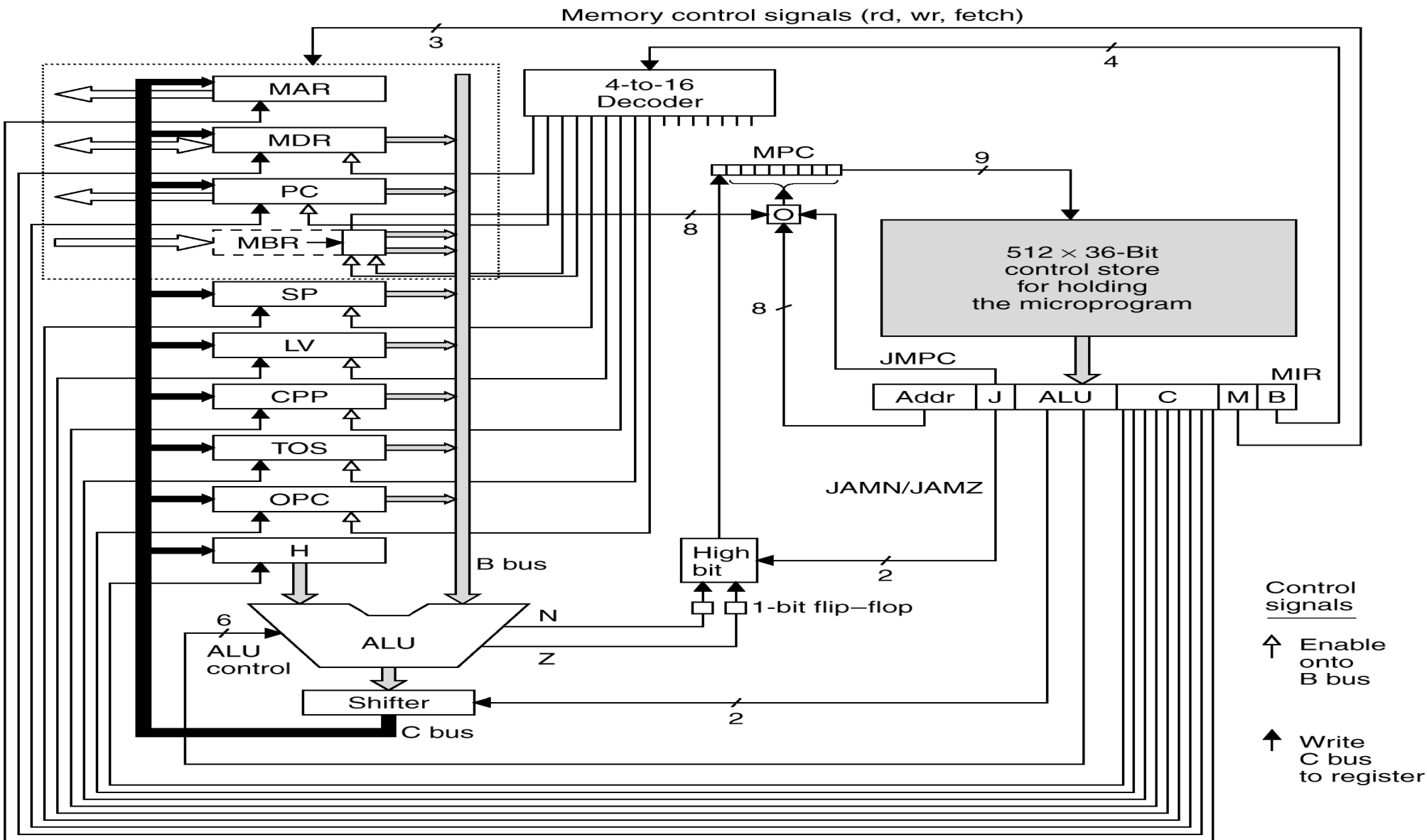


# Esercizio sulle architetture di CPU I

Si vuole realizzare una semplice CPU con architettura CISC a 8 bit dotata di due registri general purpose, due registri per il fetch delle istruzioni (il Program Counter e il registro istruzione corrente) e due registri per il trasferimento dei dati da/per la memoria (uno per gli indirizzi e l'altro per i dati). La CPU deve essere in grado di svolgere 8 operazioni aritmetiche a numeri interi. Tutte le altre specifiche possono essere liberamente scelte.

- Disegnare l'architettura generale (in particolare il data path) di tale CPU (comprensiva dei segnali di controllo) e illustrare concisamente il suo funzionamento.
- Definire il formato di una microistruzione per tale architettura cercando di minimizzare la sua lunghezza.
- Indicare possibili modifiche dell'architettura proposta in grado di migliorare le prestazioni.

# Architettura di riferimento



## Esercizio sulle architetture di CPU II

Si vuole realizzare una semplice CPU con architettura RISC a 8 bit dotata di un registro general purpose, un registro accumulatore, due registri per il fetch delle istruzioni (il Program Counter e il Registro Istruzione Corrente) e due registri per il trasferimento dei dati da/per la memoria (uno per gli indirizzi e l'altro per i dati). La CPU deve essere in grado di svolgere 16 operazioni aritmetiche a numeri interi. Tutte le altre specifiche possono essere liberamente scelte.

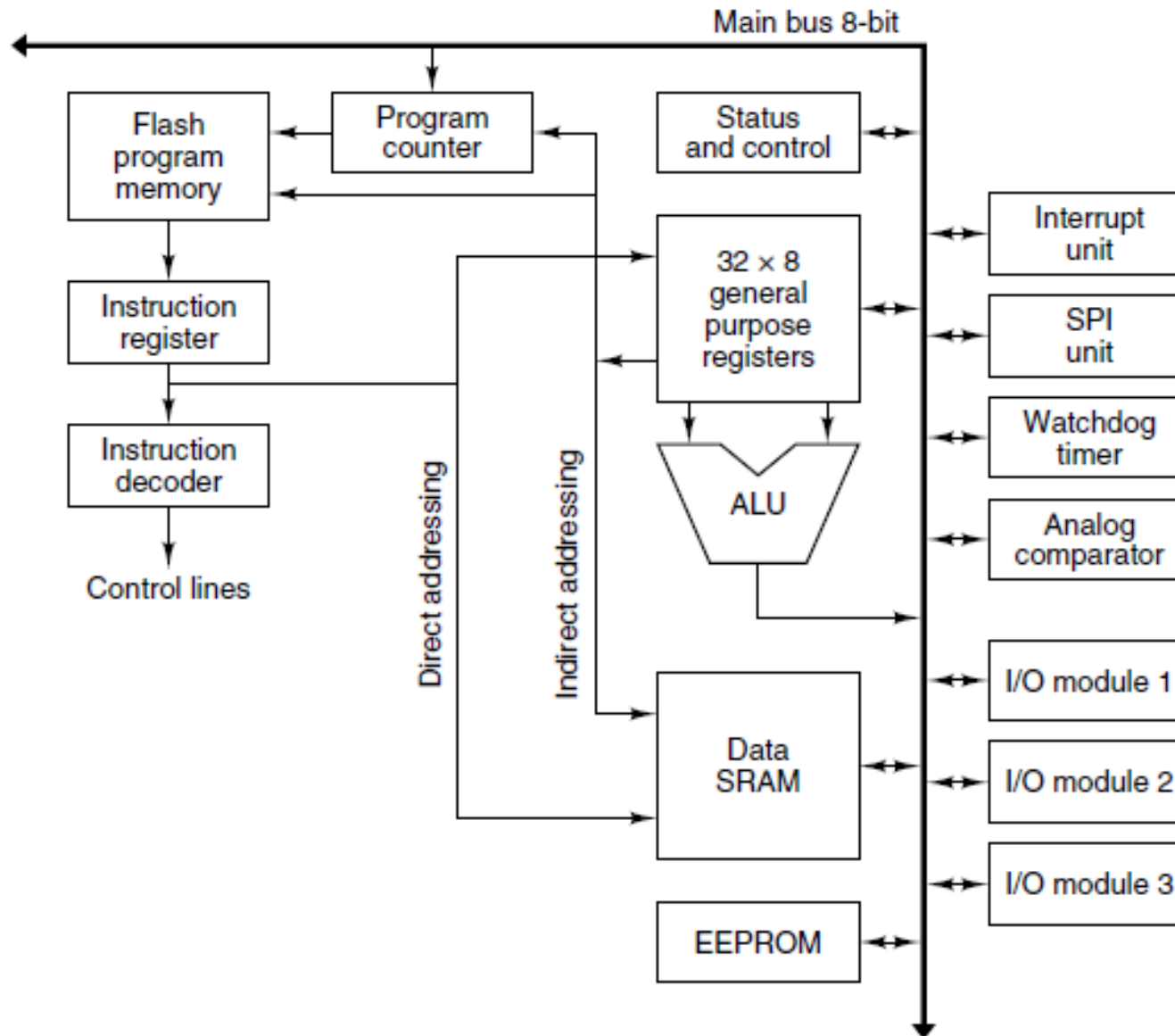
- Disegnare l'architettura generale (in particolare il data path) di tale CPU (comprensiva dei segnali di controllo) e illustrare coincisamente il suo funzionamento
- Definire il formato di una istruzione macchina per tale architettura cercando di minimizzare la sua lunghezza.
- Indicare possibili modifiche dell'architettura proposta per trasformarla in un'architettura CISC.

# Architetture di CPU III

Si vuole realizzare una CPU per applicazioni embedded che non possiede RAM e nella quale tutte le istruzioni macchina da eseguire sono memorizzate in una ROM. Tale CPU è dotata di due registri *general purpose*, un registro accumulatore, una porta di I/O e due registri per il caricamento delle istruzioni dalla ROM. La CPU deve essere in grado di eseguire 8 operazioni aritmetiche a numeri interi. L'esecuzione delle istruzioni macchina è strettamente sequenziale. Tutte le altre specifiche possono essere liberamente scelte.

- Disegnare l'architettura generale (in particolare il data path) di tale CPU (comprensiva dei segnali di controllo) secondo i principi RISC e illustrare coincisamente il suo funzionamento.
- Definire il formato di una istruzione macchina per tale architettura fissando la dimensione dei registri.
- Indicare possibili modifiche dell'architettura proposta per poter leggere e scrivere dati memorizzati su una memoria RAM.

# Architettura di riferimento



# Architetture di CPU IV

Si vuole realizzare una CPU con architettura CISC dotata di tre registri general purpose, un registro accumulatore e due coppie di registri per il trasferimento di dati e istruzioni da/per la memoria. La CPU deve essere in grado di eseguire 16 operazioni aritmetiche a numeri interi e deve essere dotata di 4 stadi di pipeline, il primo dei quali è costituito da una unità IFU. Tutte le altre specifiche possono essere liberamente scelte.

- Disegnare l'architettura generale (in particolare il data path) di tale CPU (comprensiva dei segnali di controllo) e illustrare coincisamente il suo funzionamento.
- Definire il formato di una istruzione macchina per tale architettura fissando la dimensione dei registri.
- Indicare possibili modifiche dell'architettura proposta per diminuire il fenomeno delle collisioni tra istruzioni macchina.

# Architettura di riferimento

