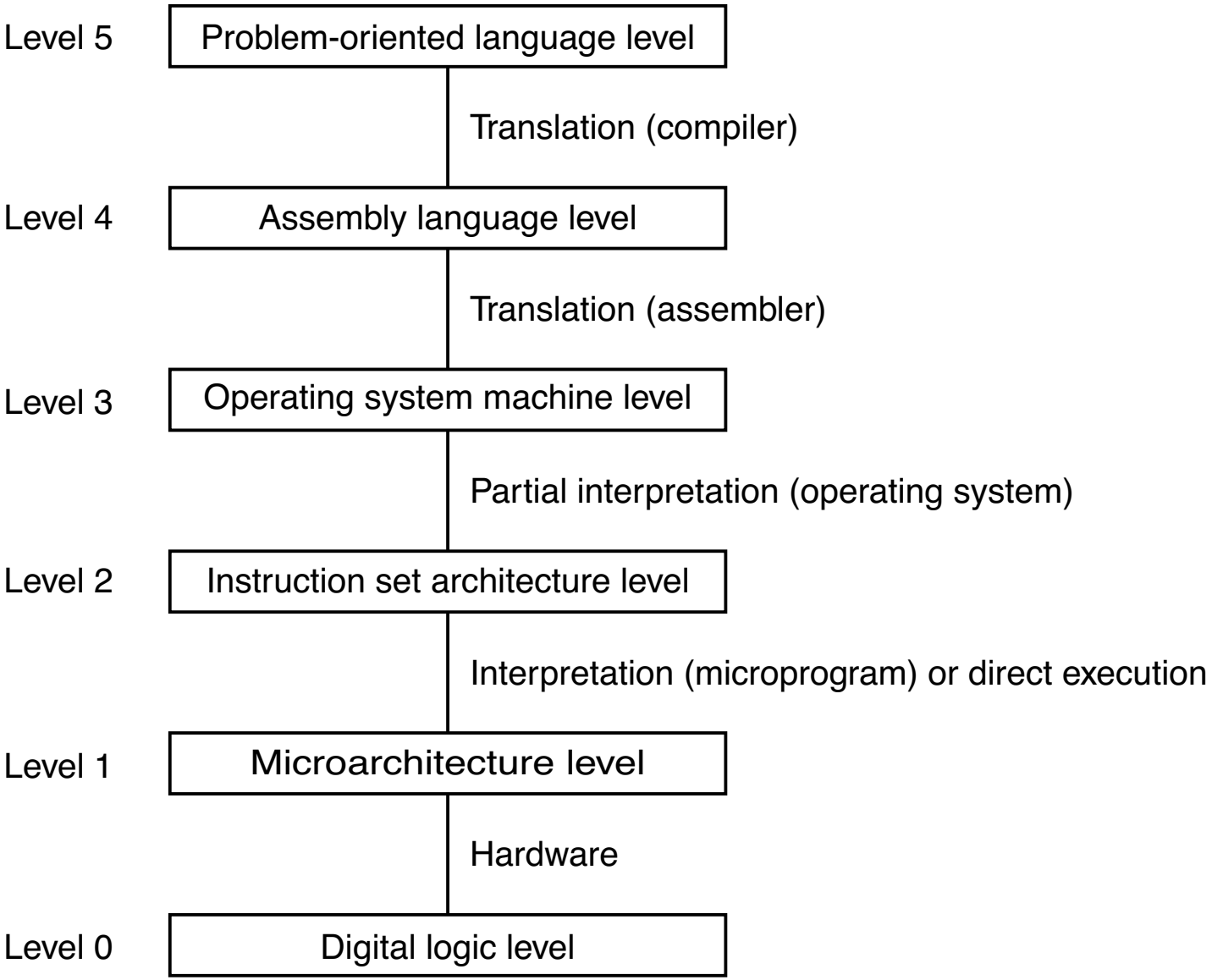
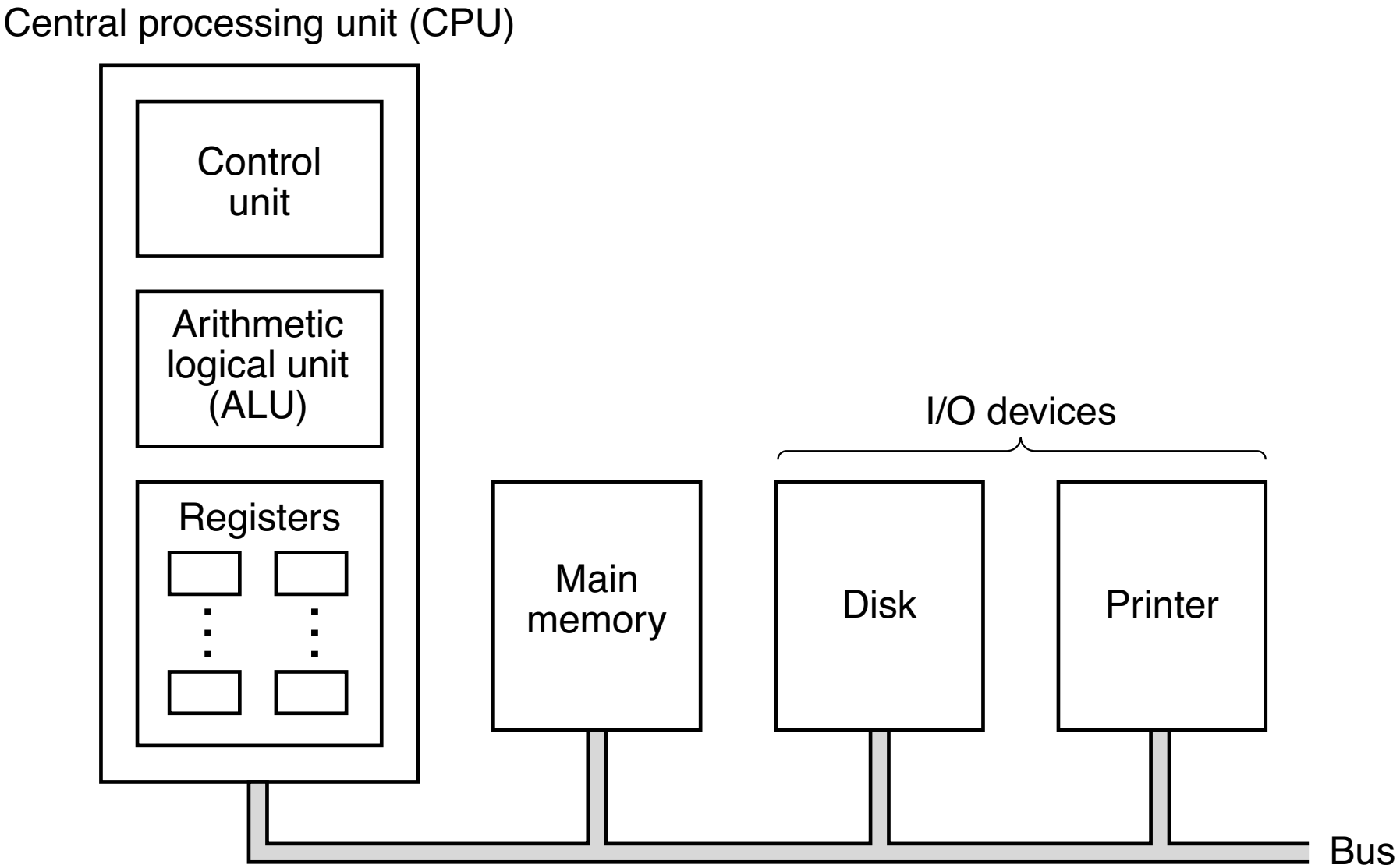
# Livello 1: processore



# Schema base di un calcolatore



# Fetch-decode-execute

Compito del processore è quello di eseguire il ciclo fetch-decode-execute il più rapidamente possibile.

La CPU, dopo essere stata inizialmente avviata,

- fetch: carica una ben determinata istruzione macchina dalla memoria

# Fetch-decode-execute

Compito del processore è quello di eseguire il ciclo fetch-decode-execute il più rapidamente possibile.

La CPU, dopo essere stata inizialmente avviata,

- fetch: carica una ben determinata istruzione macchina dalla memoria
- decode: determina il tipo di istruzione e i suoi argomenti

# Fetch-decode-execute

Compito del processore è quello di eseguire il ciclo fetch-decode-execute il più rapidamente possibile.

La CPU, dopo essere stata inizialmente avviata,

- fetch: carica una ben determinata istruzione macchina dalla memoria
- decode: determina il tipo di istruzione e i suoi argomenti
- execute: carica gli argomenti, svolge le operazioni necessarie a eseguire l'istruzione, memorizza i risultati e si predispone per il ciclo successivo.

Il ciclo quindi si ripete fino a quando non si incontra una particolare istruzione di arresto.

# Il data path nel processore

Il percorso fisico dei dati (data path) comprende

- alcuni registri (memoria interna alla CPU)

# Il data path nel processore

Il percorso fisico dei dati (data path) comprende

- alcuni registri (memoria interna alla CPU)
- un'unità aritmetica e logica (ALU)

# Il data path nel processore

Il percorso fisico dei dati (data path) comprende

- alcuni registri (memoria interna alla CPU)
- un'unità aritmetica e logica (ALU)
- dei bus che collegano ALU e registri.

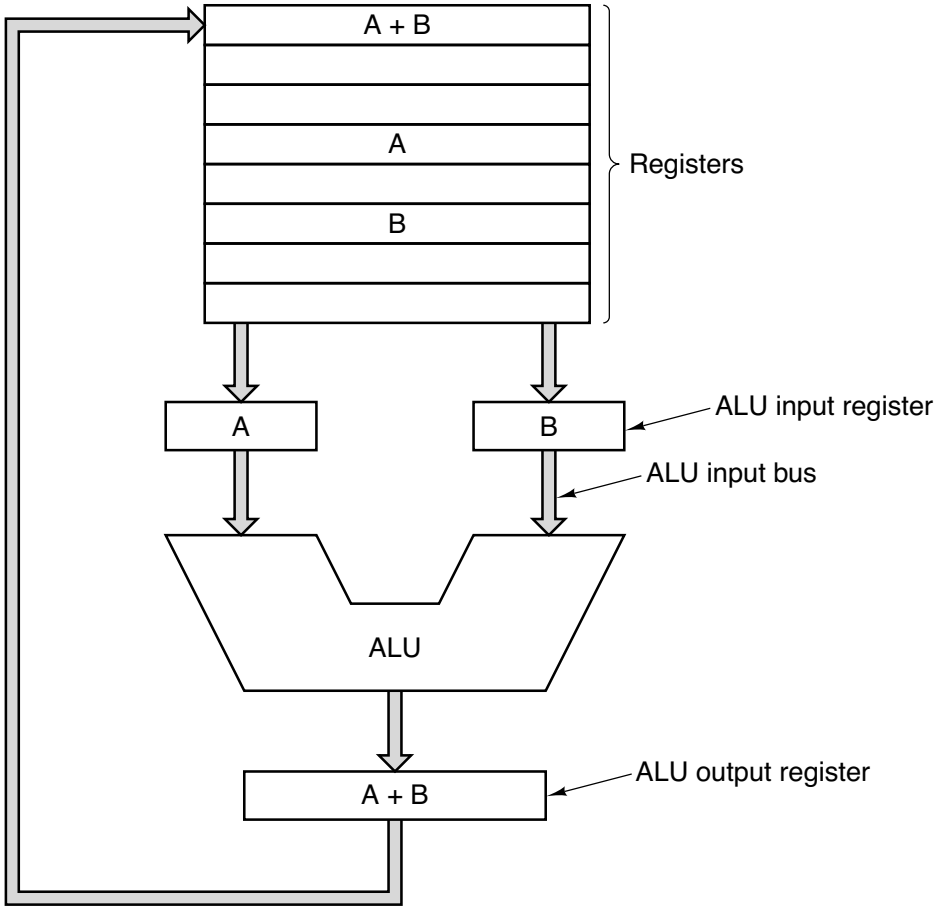
La ALU esegue le sequenze di micro-operazioni aritmetico-logiche necessarie al completamento dell'istruzione.

N.B.: attenzione alle omonimie! Si parla convenzionalmente di registro (interno alla CPU) oppure di locazione di memoria (principale)!

N.B.: un registro della CPU non ha necessariamente (di più: quasi mai) la dimensione di una locazione nella memoria principale!

# Es.: data path della somma

Es.: pseudo-istruzione di somma `sum(A,B)`.
L'istruzione in particolare dirà da dove caricare gli addendi e dove memorizzare il risultato.



# Micro-operazioni della CPU

Una micro-operazione è eseguita nel data path in un singolo ciclo di clock.

Esempi di micro-operazione nelle attuali CPU:

- calcolo di un'operazione aritmetica o logica con accesso limitato ai registri della CPU

# Micro-operazioni della CPU

Una micro-operazione è eseguita nel data path in un singolo ciclo di clock.

Esempi di micro-operazione nelle attuali CPU:

- calcolo di un'operazione aritmetica o logica con accesso limitato ai registri della CPU
- singolo accesso alla memoria in lettura o scrittura.

# Micro-operazioni della CPU

Una micro-operazione è eseguita nel data path in un singolo ciclo di clock.

Esempi di micro-operazione nelle attuali CPU:

- calcolo di un'operazione aritmetica o logica con accesso limitato ai registri della CPU
- singolo accesso alla memoria in lettura o scrittura.

Raramente un'istruzione macchina si compone di una sola micro-operazione.

# Unità di controllo della CPU

Il funzionamento del data path viene regolato mediante opportuni segnali dall'unità di controllo.

L'unità di controllo è un circuito sequenziale che coordina più in generale il funzionamento dell'intero processore. In particolare,

- viene configurata dall'istruzione in esecuzione una volta che essa è stata caricata nell'Instruction Register (IR)

# Unità di controllo della CPU

Il funzionamento del data path viene regolato mediante opportuni segnali dall'unità di controllo.

L'unità di controllo è un circuito sequenziale che coordina più in generale il funzionamento dell'intero processore. In particolare,

- viene configurata dall'istruzione in esecuzione una volta che essa è stata caricata nell'Instruction Register (IR)
- abilita la lettura e scrittura nei registri coinvolti

# Unità di controllo della CPU

Il funzionamento del data path viene regolato mediante opportuni segnali dall'unità di controllo.

L'unità di controllo è un circuito sequenziale che coordina più in generale il funzionamento dell'intero processore. In particolare,

- viene configurata dall'istruzione in esecuzione una volta che essa è stata caricata nell'Instruction Register (IR)
- abilita la lettura e scrittura nei registri coinvolti
- invia segnali di controllo ai circuiti della ALU

# Unità di controllo della CPU

Il funzionamento del data path viene regolato mediante opportuni segnali dall'unità di controllo.

L'unità di controllo è un circuito sequenziale che coordina più in generale il funzionamento dell'intero processore. In particolare,

- viene configurata dall'istruzione in esecuzione una volta che essa è stata caricata nell'Instruction Register (IR)
- abilita la lettura e scrittura nei registri coinvolti
- invia segnali di controllo ai circuiti della ALU
- sincronizza la comunicazione tra CPU e memoria principale.

# Logica cablata o programmata

Il set di micro-operazioni può essere realizzato in due modi.

- Logica cablata: si realizza l'hardware di un circuito sequenziale tradizionale. Più complicato e costoso da realizzare, meno flessibile, è però più veloce e offre prestazioni migliori.

# Logica cablata o programmata

Il set di micro-operazioni può essere realizzato in due modi.

- Logica cablata: si realizza l'hardware di un circuito sequenziale tradizionale. Più complicato e costoso da realizzare, meno flessibile, è però più veloce e offre prestazioni migliori.
- Logica programmata: l'unità di controllo è a sua volta una micro-architettura capace di eseguire un micro-programma. Più semplice, economico e flessibile, ma più lento.

Diversi esempi di questa dicotomia: schede grafiche, processori di segnale, . . .

# Logica cablata o programmata?

Anni ’40: primi processori in logica cablata.

Anni ’50: primi processori con un con un insieme più ricco di istruzioni in logica micro-programmata.

Es.: IBM 360 realizzava un unico insieme di istruzioni in versioni molto diverse per prestazioni e costi a seconda del tipo di logica.

Anni ’70: apice della micro-programmazione a causa del *gap* tra velocità della CPU e delle memorie. Istruzioni macchina di una complessità vicina ai linguaggi di programmazione di livello alto.

Es.: VAX (Digital EC).

# Processori RISC e CISC

Anni ’80: la riduzione del *gap* tra prestazioni della CPU e delle memorie riporta l’interesse sulla logica cablata. La famiglia di architetture che ne risulta prende il nome di RISC: Reduced Instruction Set Computer. Molto veloci e performanti.

Es.: IBM 801, SUN Sparc (Patterson, Berkeley), MIPS (Hennessy, Stanford), Alpha (Digital), PowerPC (Motorola) IBM, ARM.

Anni 90: affermazione della filosofia RISC con l’importante eccezione delle CPU Intel x86 CISC: Complex Instruction Set Computer (Es.: IA-32), motivata da compatibilità con hardware preesistente.

# I processori attuali

La contrapposizione RISC – CISC si è sfumata con l'avvento di processori RISC in grado di eseguire ampi insiemi di micro-operazioni sempre più complesse. RISC è più efficiente del 20-30%.

Es.: le istruzioni CISC nei processori Intel x86 IA-32 si appoggiano su un core RISC:

- istruzioni semplici, ricorrenti: realizzate direttamente da istruzioni RISC
- istruzioni complesse: scomposte in più istruzioni RISC
- istruzioni sofisticate: eseguite mediante un programma macchina.

# Chi scrive le istruzioni macchina?

Data la difficoltà di tradurre un problema in istruzioni macchina, il programmatore sfrutta la compilazione o l'interpretazione di un programma scritto in un linguaggio ad alto livello: Java, C, C++, Scheme, Pascal, Shell, ...

- il compilatore traduce un programma sorgente ad alto livello nel programma in codice macchina che realizza la procedura da eseguire. Es.: C, C++

# Chi scrive le istruzioni macchina?

Data la difficoltà di tradurre un problema in istruzioni macchina, il programmatore sfrutta la compilazione o l'interpretazione di un programma scritto in un linguaggio ad alto livello: Java, C, C++, Scheme, Pascal, Shell, ...

- il compilatore traduce un programma sorgente ad alto livello nel programma in codice macchina che realizza la procedura da eseguire. Es.: C, C++
- l'interprete traduce una sequenza di istruzioni di alto livello nella corrispondente sequenza di istruzioni macchina. Es.: Java bytecode, linguaggio di shell.

# Un esempio: l'hardware Mic

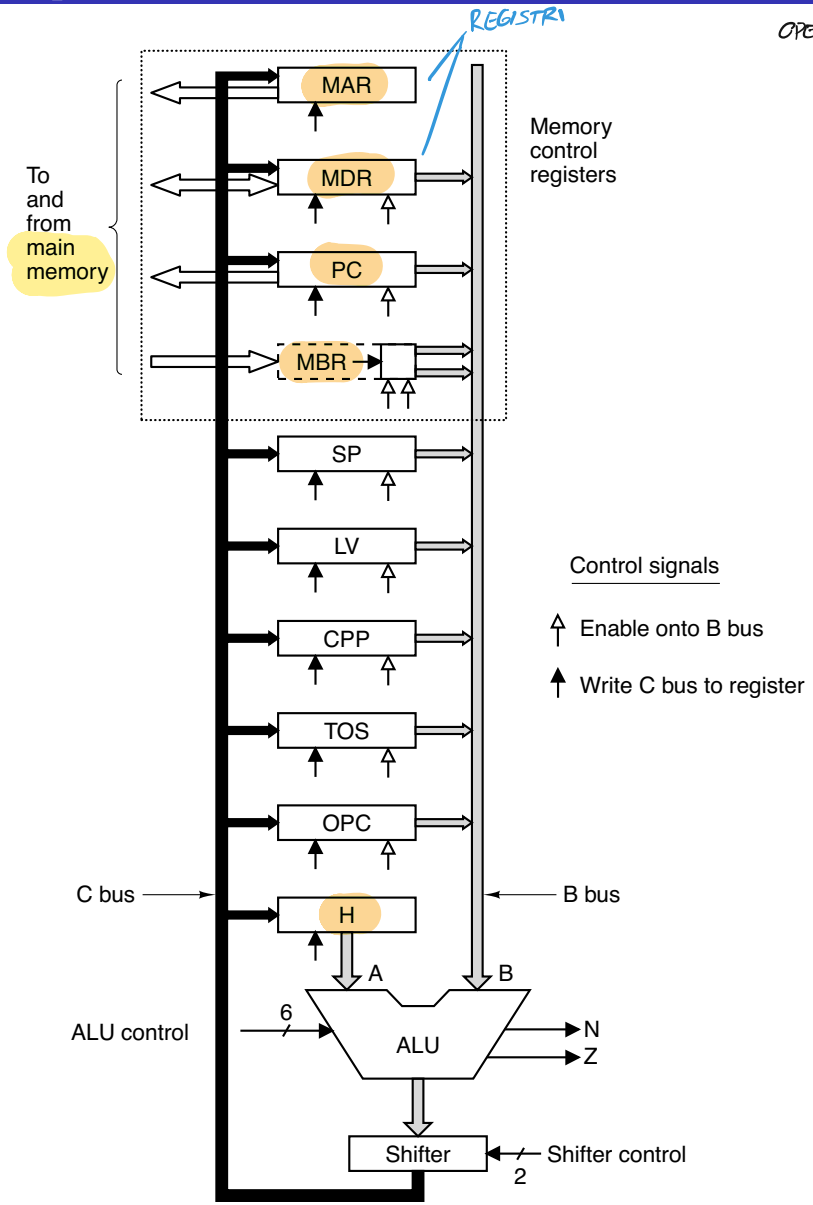
Affrontiamo i problemi generali di progettazione di un processore guardando molto sinteticamente all'hardware Mic, che esegue le istruzioni macchina generate dal compilatore Java (bytecode).

Mic è implementato dall'architettura ARM estesa ARM926EJ-S. Più frequentemente il bytecode è interpretato dalla Java Virtual Machine (JVM).

La porzione di memoria principale dedicata a un programma eseguito da Mic si compone di tre parti:

- area del codice (istruzioni macchina)
- area delle costanti (dati costanti usati nel programma)
- area stack (dati variabili usati nel programma).

# Mic-1: data path




"Circolazione" di dati e risultati lungo registri e ALU.

INVERTIRE UN NUMERO IN COMPL. 2

10101101 INVERSIONE

01010010 + 1 = 01010010

INCREMENTO

ESISTE SOLO INVA, NON INVB

B-11: 10101100

11111111 +

ENA=0 AND INVA=1

A=1

# Mic-1: ALU

La ALU svolge somme e sottrazioni su interi in complemento a 2 e operazioni logiche bit-wise.

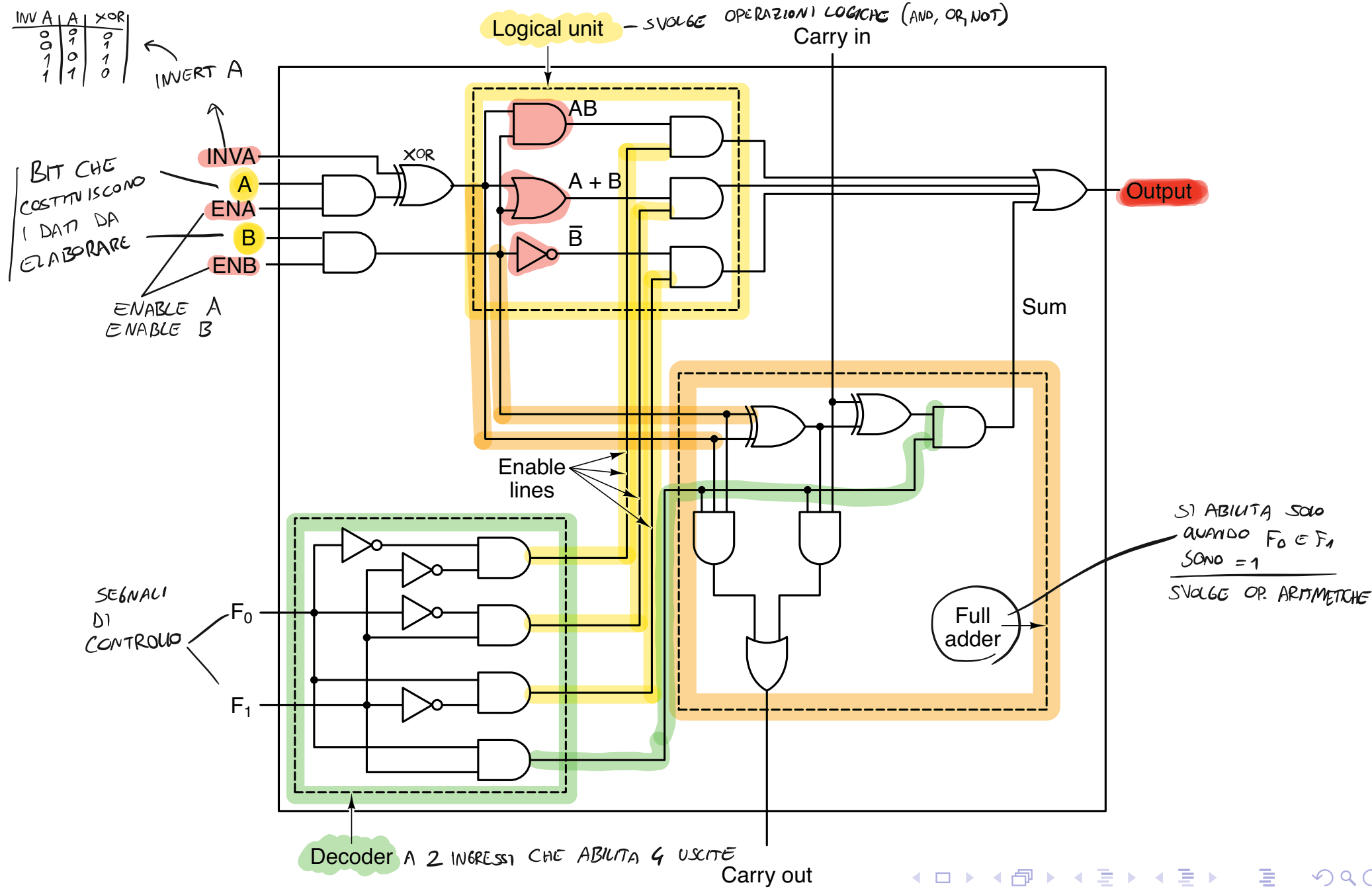
I segnali di controllo $F_0$ e $F_1$ determinano il tipo e l'operazione logica; ENA, ENB, INVA, Carry In sul bit più a destra l'operazione aritmetica da eseguire.

Esempi di aritmetica realizzata:

RIPORTO IN ENTRATA

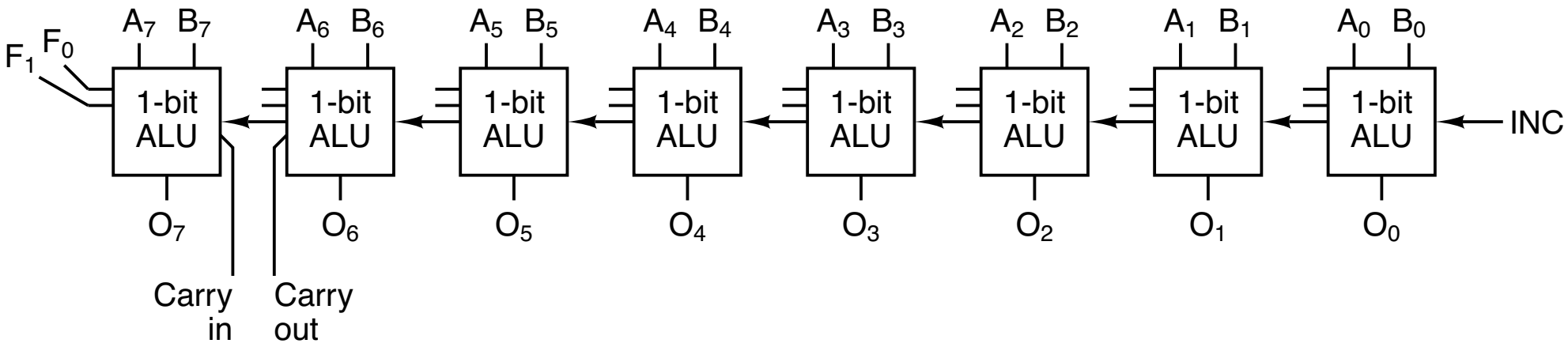
- $A + B + 1$ con ENA=1, ENB=1, INVA=0, Carry=1
- $A + 1$ con ENA=1, ENB=0, INVA=0, Carry=1
- $B + 1$ con ENA=0, ENB=1, INVA=0, Carry=1
- $-A$ con ENA=1, ENB=0, INVA=1, Carry=1 (INVERSIONE, INCREMENTO)
- $B - A$ con ENA=1, ENB=1, INVA=1, Carry=1
- $B - 1$ con ENA=0, ENB=1, INVA=1, Carry=0.

# Mic-1: modulo 1-bit della ALU



# Mic-1: ALU a *N* bit

Es.: ALU completa a 8 bit formata da una cascata di ALU a 1 bit




Mic-1 contiene una ALU a 32 bit.

# Mic-1: registri

Registri dedicati al dato

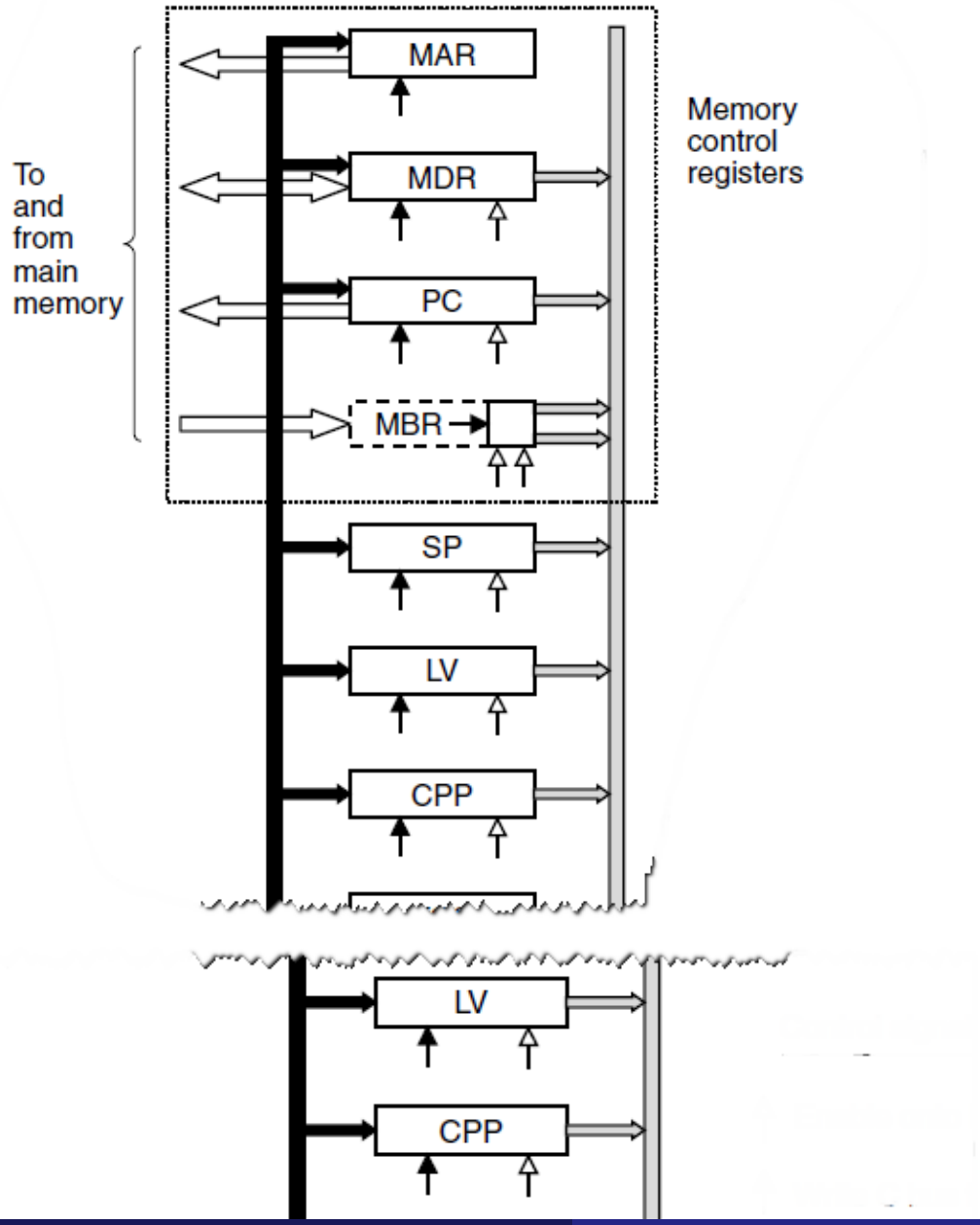
- MAR (Memory Address Register): indirizzo del dato
- MDR (Memory Data Register): dato

Registri dedicati all'istruzione

- PC (Program Counter): indirizzo della prossima istruzione da eseguire
- MBR (Memory Bytecode Register): indirizzo dell'istruzione in esecuzione

Altri registri (non approfondiamo). H contiene sempre il secondo argomento dell'istruzione, spesso il risultato dell'operazione.

# Mic-1: Registri



# Circuito di controllo

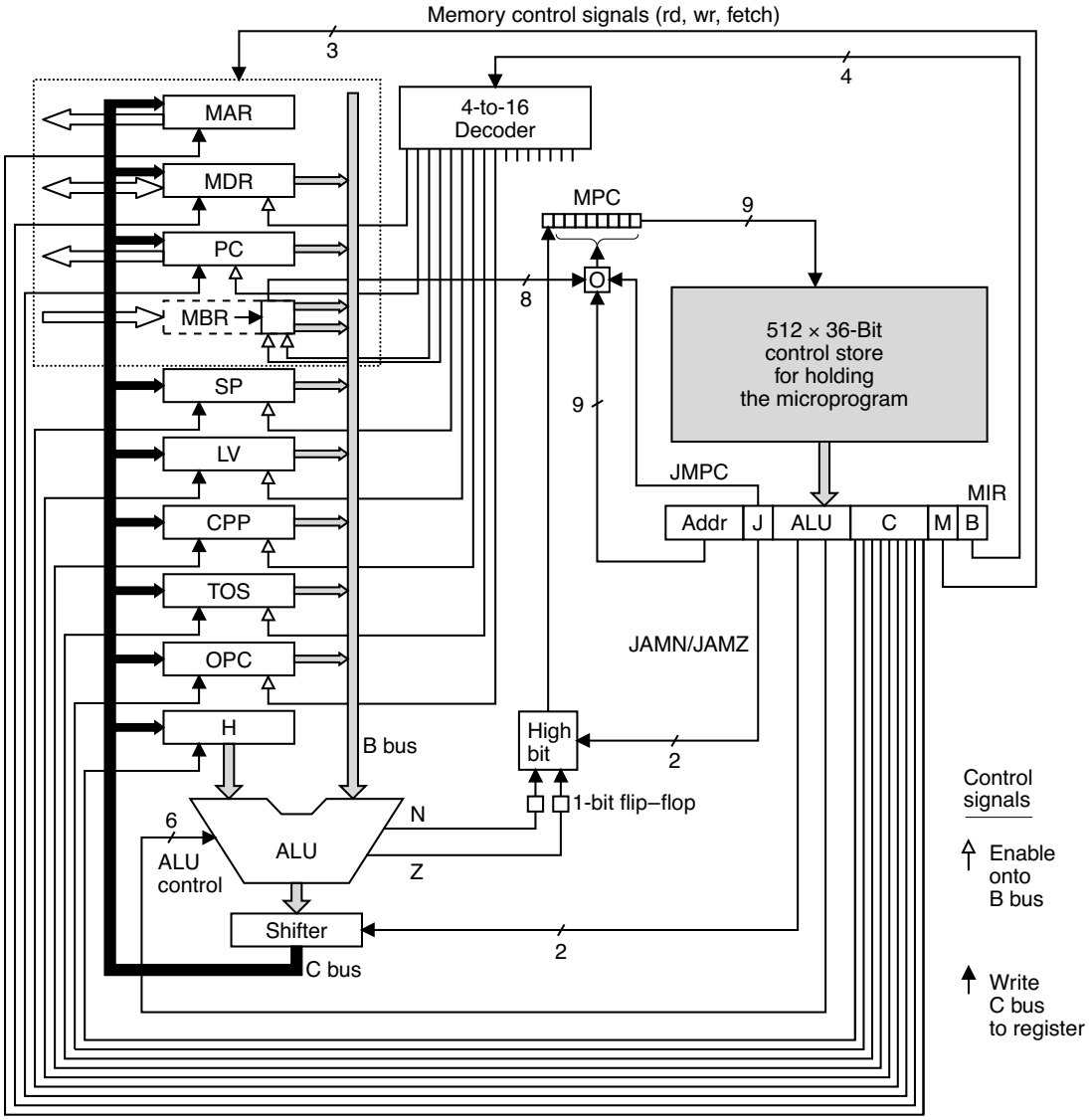
Mic-1 è in logica programmata. Il micro-codice è contenuto in una memoria ROM

2 registri e un multiplexer sono sufficienti per realizzare le micro-operazioni causate dalla micro-istruzione.

Micro-istruzioni di 36 bit:

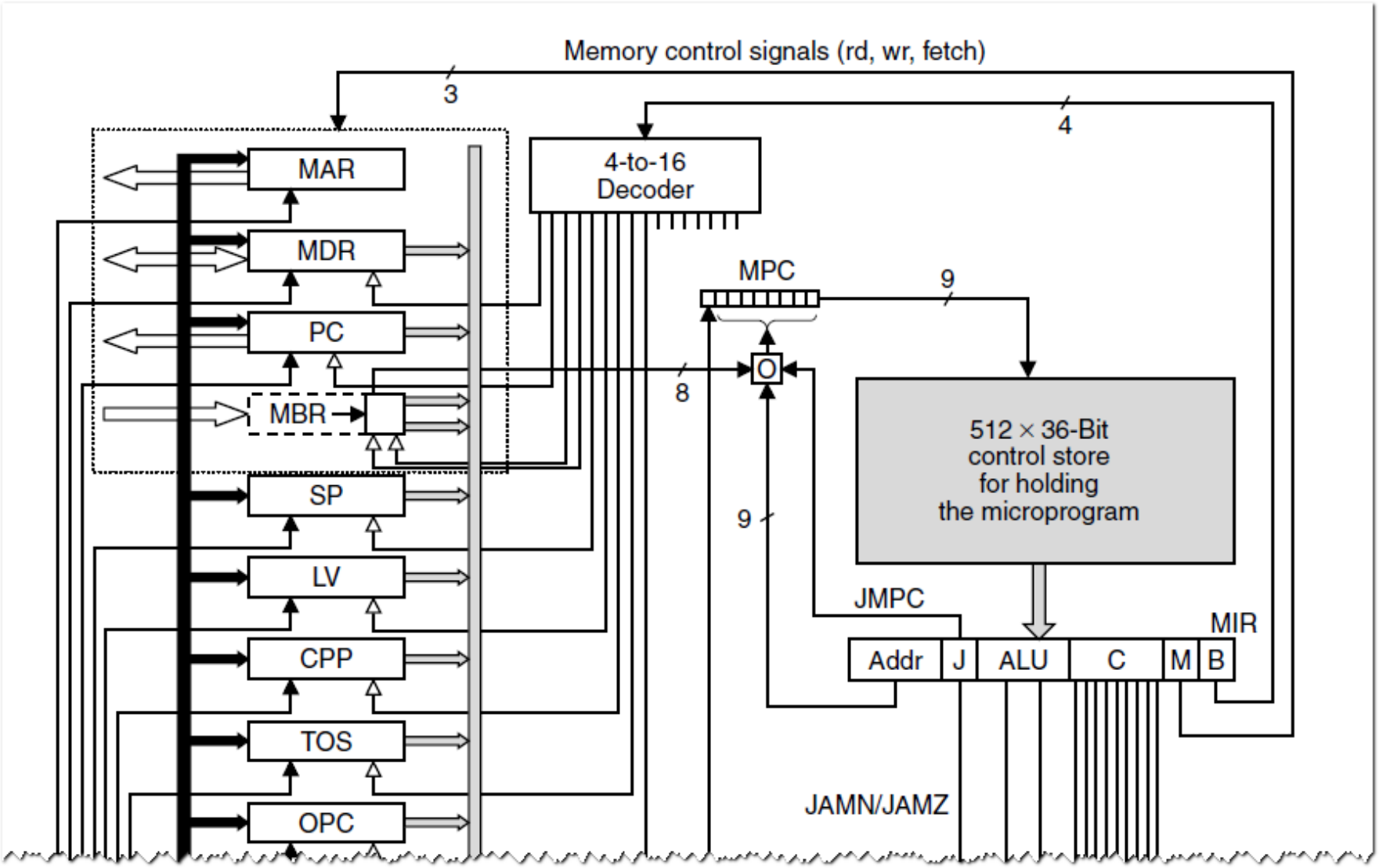
- in parte segnali di controllo
- in parte individuano la micro-istruzione successiva.

# Circuito di controllo




Da ricordare solo indicativamente!

# Circuito di controllo: Control store



# Circuito di controllo

Invia più tipi di segnali:

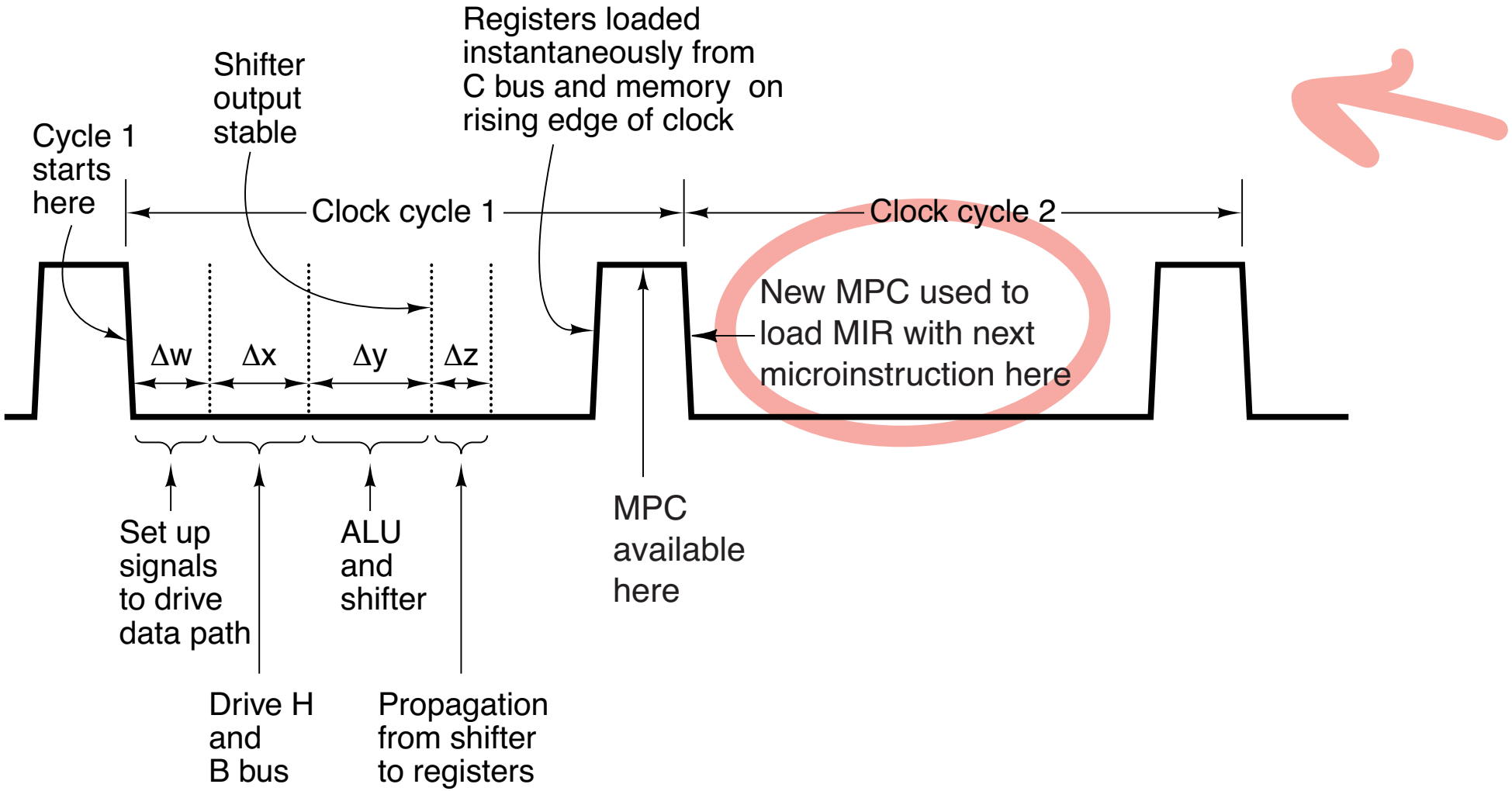
- ai registri l'accesso in lettura e scrittura
- all'area stack della memoria l'accesso in lettura e scrittura
- all'area codice della memoria l'accesso in lettura per il fetch
- alla ALU la micro-istruzione da eseguire.

Determina l'istruzione successiva via:

- l'istruzione corrente nel registro MBR
- la micro-istruzione, se l'istruzione corrente prevede un salto altrove nell'area codice
- l'uscita della ALU, se l'istruzione corrente prevede un salto condizionato dal risultato.

# Utilizzo del ciclo di clock

Il periodo di clock dev'essere maggiore della somma dei tempi necessari a svolgere una micro-istruzione.



# Aumento della velocità di calcolo

La velocità di calcolo può essere aumentata riducendo il ciclo di clock:

- migliorare la tecnologia dei circuiti integrati (transistor più veloci)
- ottimizzare l'organizzazione spaziale dei circuiti integrati minimizzando i tempi di trasferimento dei segnali.

# Aumento della velocità di calcolo

La velocità di calcolo può essere aumentata riducendo il ciclo di clock:

- migliorare la tecnologia dei circuiti integrati (transistor più veloci)
- ottimizzare l'organizzazione spaziale dei circuiti integrati minimizzando i tempi di trasferimento dei segnali.

Oppure, ottimizzando il numero di micro-istruzioni nell'unità di tempo:

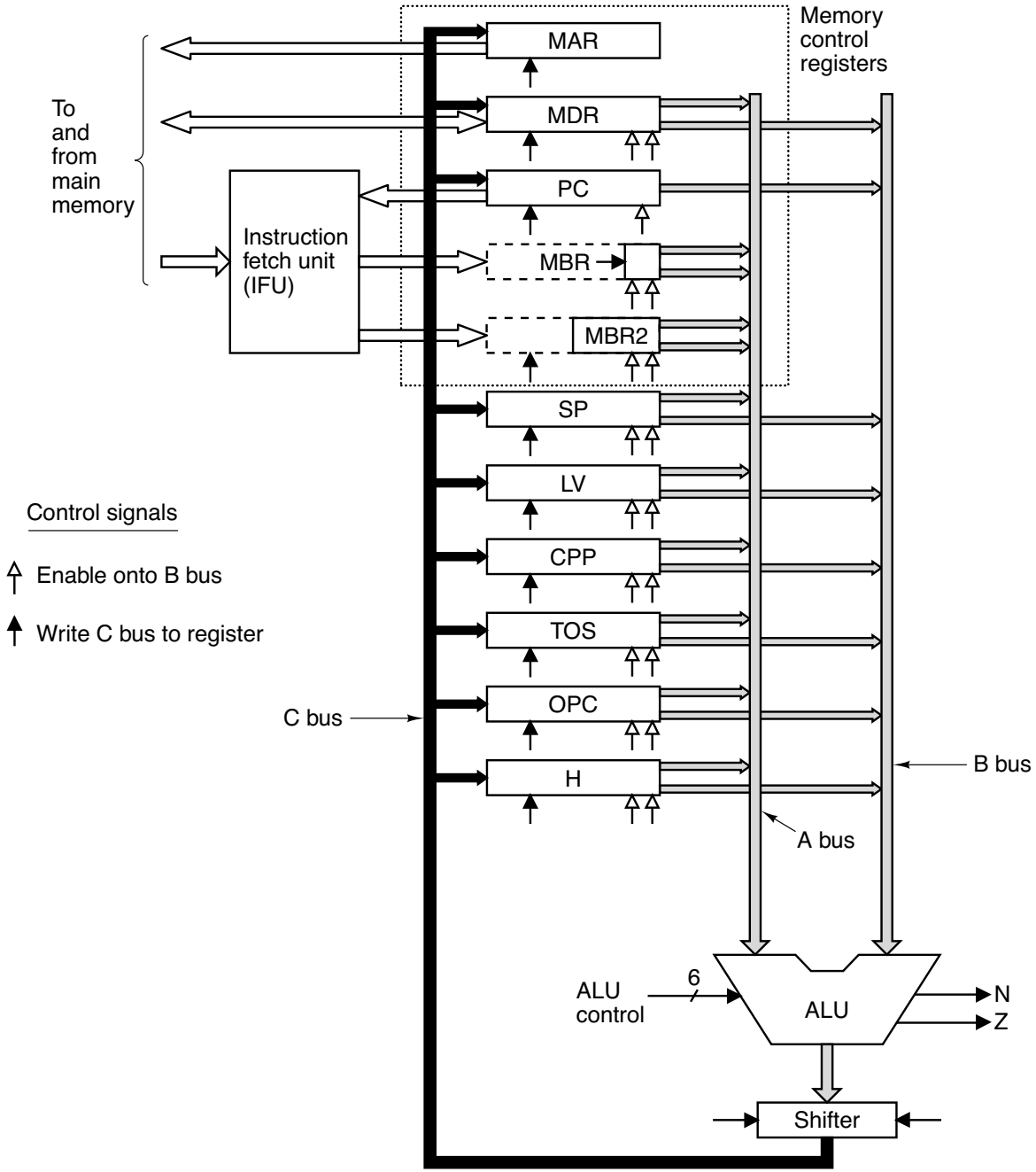
- diminuire il numero di micro-istruzioni necessarie per eseguire un'istruzione
- parallelizzare le micro-operazioni.

# Aumento delle micro-operazioni

Come si realizzano micro-istruzioni più potenti?

- sviluppando ALU con più funzioni di calcolo disponibili
- aumentando il numero di registri
- aumentando il numero di bus
- bufferizzando l'accesso ai dati e alle istruzioni con l'ausilio di unità di memorizzazione intermedie. Es. (Mic-2): Instruction Fetch Unit (IFU).

# Mic-2


Memory
control
registers
MAR
MDR
PC
MBR
MBR2
SP
LV
CPP
TOS
OPC
H
To
and
from
main
memory
Instruction
fetch unit
(IFU)
Control signals
Enable onto B bus
Write C bus to register
C bus
B bus
A bus
ALU
control
6
ALU
N
Z
Shifter

# Parallelizzazione micro-operazioni

Pipelining:

# Parallelizzazione micro-operazioni

Pipelining:

- l'esecuzione dell'istruzione macchina è divisa in più stadi, ciascuno realizzato da una o più micro-operazioni
- le micro-operazioni di ogni stadio possono essere eseguite in parallelo a quelle di altri stadi
- in tal modo, a ogni ciclo di clock sono eseguiti contemporaneamente più stadi.

Tecnica utilizzata in tutti i processori. In ambito Intel: dal 486 in poi (1989).

# Parallelizzazione micro-operazioni

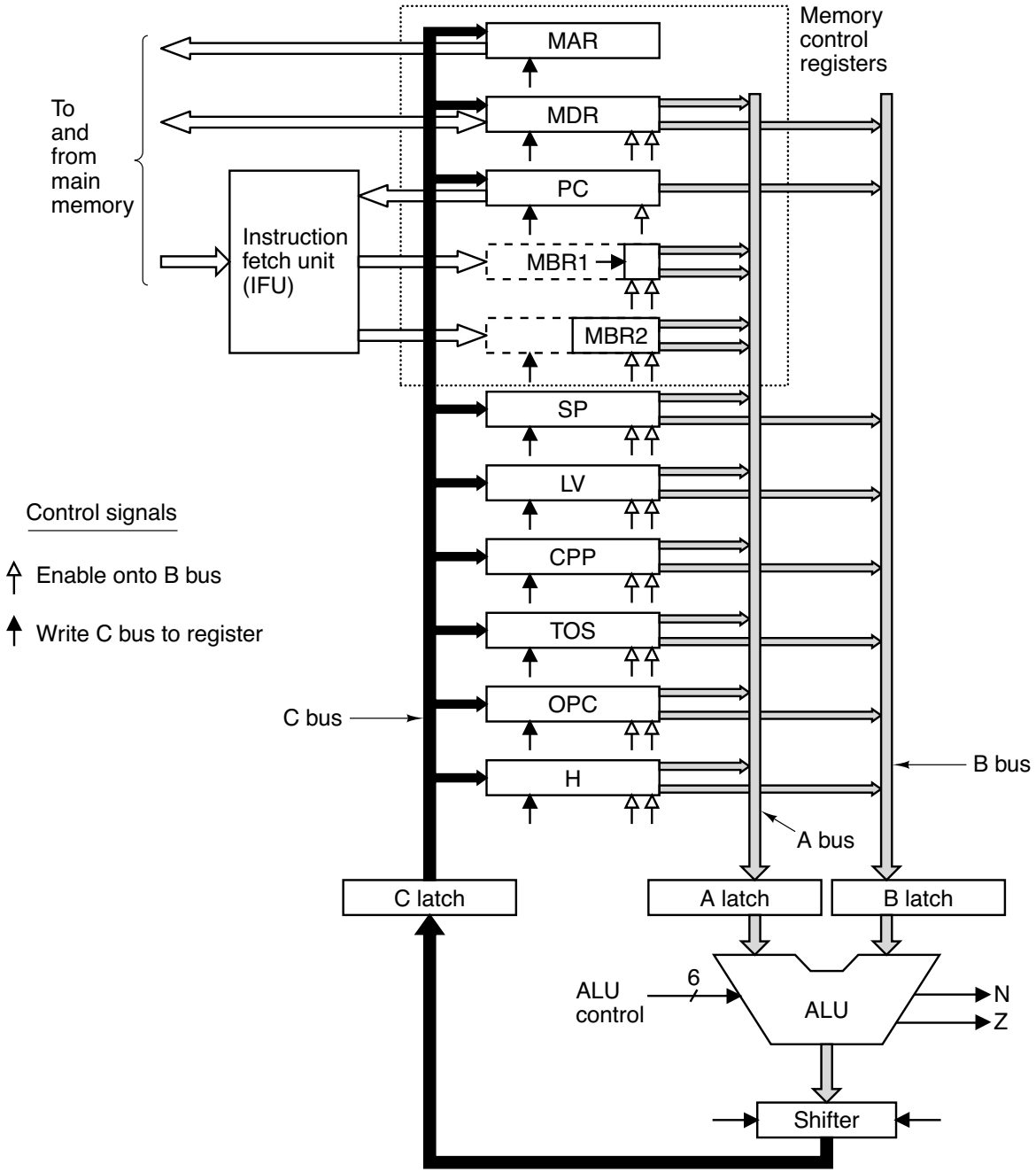
Pipelining:

- l'esecuzione dell'istruzione macchina è divisa in più stadi, ciascuno realizzato da una o più micro-operazioni
- le micro-operazioni di ogni stadio possono essere eseguite in parallelo a quelle di altri stadi
- in tal modo, a ogni ciclo di clock sono eseguiti contemporaneamente più stadi.

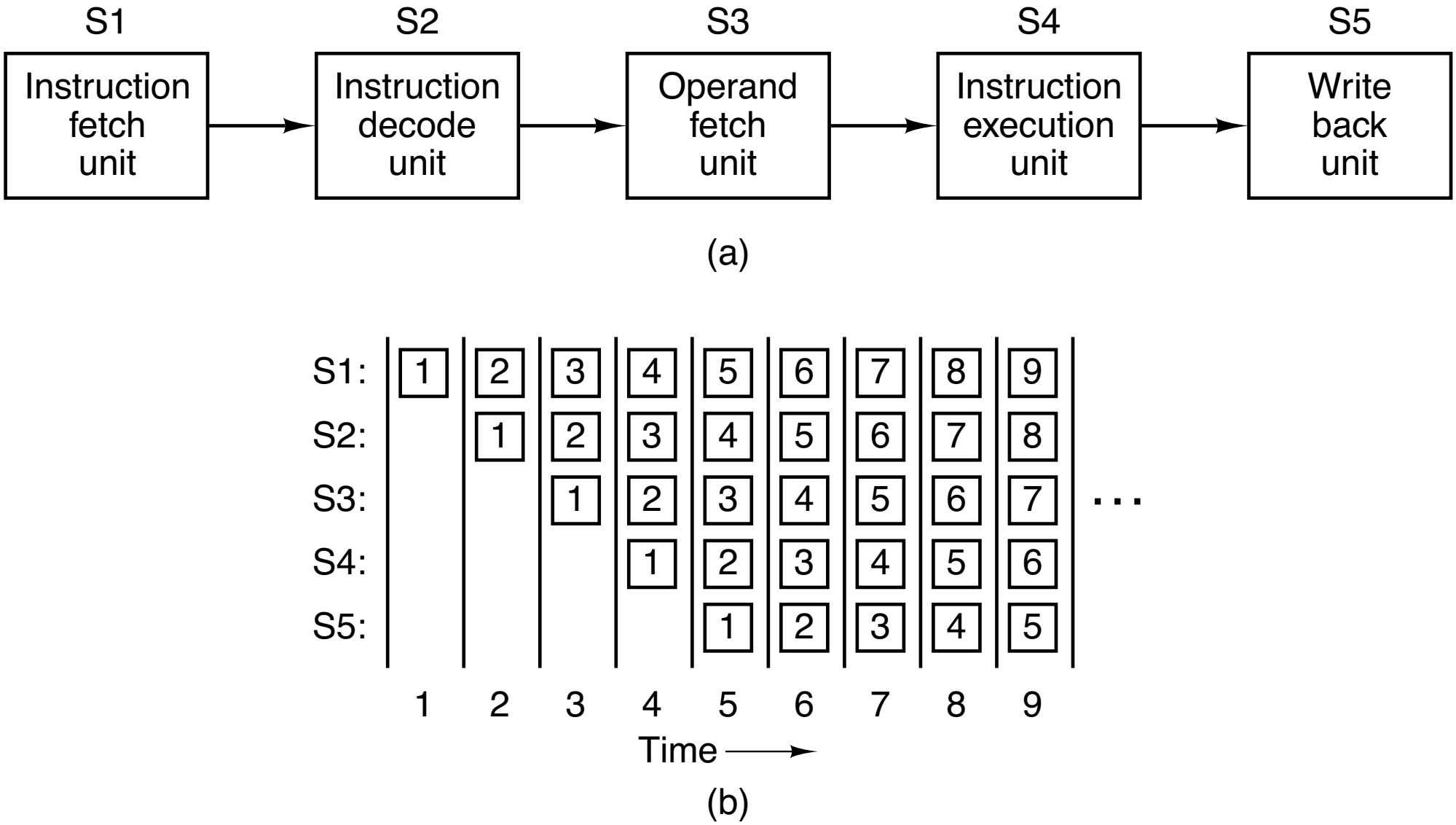
Tecnica utilizzata in tutti i processori. In ambito Intel: dal 486 in poi (1989).

Il pipelining velocizza il flusso dei calcolo ma non i tempi di risposta.

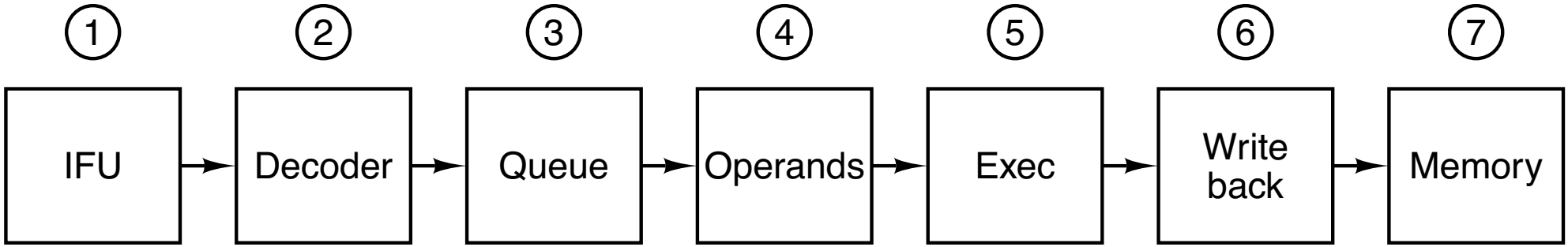
# Mic-3: pipelining a tre stadi.



# Esempio di pipelining



# Esempio di pipelining




Processori diversi usano scomposizioni e strutture della pipeline diverse.

Lunghezza tipica di una pipeline: 7–14 stadi.

Caso limite (Pentium IV): 20 stadi (guerra dei GHz).

# Processori superscalari

I processori superscalari se possibile eseguono più istruzioni macchina contemporaneamente.

Quindi, possiedono più pipeline operanti in parallelo.

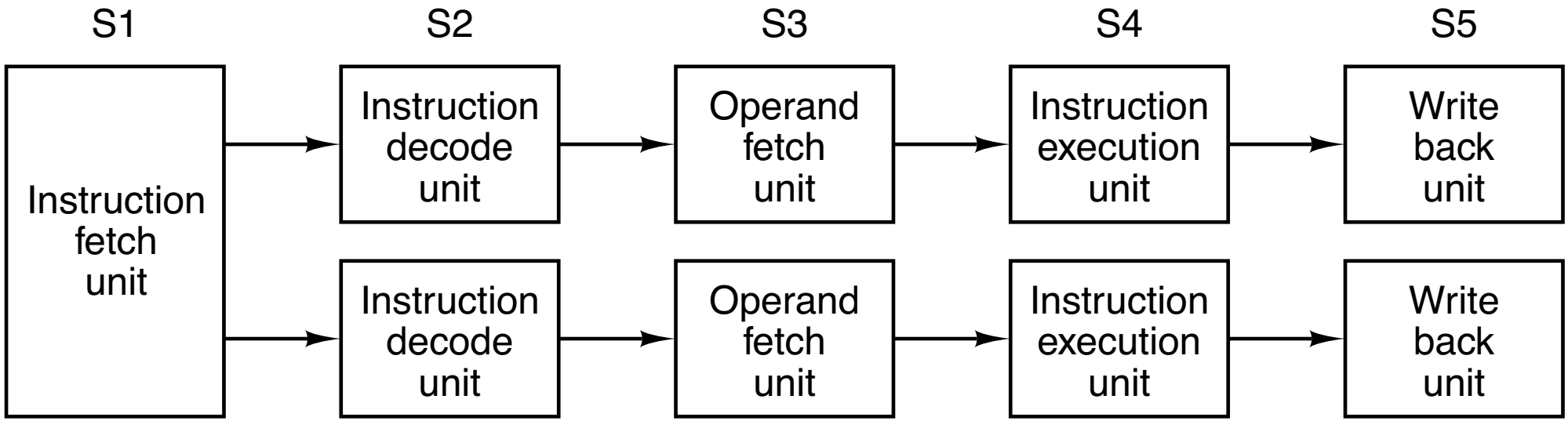
La superscalarità migliora ulteriormente il rapporto istruzioni / cicli di clock.

# Processori superscalari

I processori superscalari se possibile eseguono più istruzioni macchina contemporaneamente.

Quindi, possiedono più pipeline operanti in parallelo.

La superscalarità migliora ulteriormente il rapporto istruzioni / cicli di clock.



# Superscalarità: funzionamento

- Il primo stadio preleva più istruzioni dalla memoria e le instrada su pipeline contemporaneamente attive nel processore
- le istruzioni sono eseguite lungo stadi identici operanti in parallelo
- la conclusione di ogni istruzione non può avvenire senza che uno stadio finale singolo oppure un sistema di controllo termini ordinatamente quelle contemporaneamente eseguite. Occorre infatti gestire la dipendenza tra istruzioni.

# Dipendenza tra istruzioni:

In un programma le istruzioni devono essere logicamente eseguite nell'ordine in cui sono scritte.

Un'esecuzione parallela non controllata può portare a risultati scorretti, classificabili come

- RAW: Read After Write

```
R0 = R1
R2 = R0 + 1
```

# Dipendenza tra istruzioni:

In un programma le istruzioni devono essere logicamente eseguite nell'ordine in cui sono scritte.

Un'esecuzione parallela non controllata può portare a risultati scorretti, classificabili come

- RAW: Read After Write

```
R0 = R1
R2 = R0 + 1
```

- WAR: Write After Read

```
R1 = R0 +1
R0 = R2
```

# Dipendenza tra istruzioni:

In un programma le istruzioni devono essere logicamente eseguite nell'ordine in cui sono scritte.

Un'esecuzione parallela non controllata può portare a risultati scorretti, classificabili come

- RAW: Read After Write

```
R0 = R1
R2 = R0 + 1
```

- WAR: Write After Read

```
R1 = R0 +1
R0 = R2
```

- WAW: Write After Write

```
R0 = R1
R0 = R2 + 1.
```

# Rilevamento delle dipendenze

Le dipendenze vengono rilevate mediante una tabella delle dipendenze (scoreboard), realizzata in una memoria riservata allo scopo: per ogni registro il processore conta gli accessi in lettura e scrittura rimasti in sospeso, in quanto dipendenti da un'istruzione non terminata.

Le istruzioni dipendenti da istruzioni non ancora terminate dunque devono essere tenute in sospeso.

Si creano bolle: stadi in una pipeline bloccati dal completamento di stadi in altre pipeline.

# Mitigare le bolle

La perdita di prestazioni causata dalla dipendenza tra istruzioni può essere mitigata con diversi accorgimenti.

- Esecuzione fuori ordine: si eseguono istruzioni non dipendenti, ancorchè non consequenziali

# Mitigare le bolle

La perdita di prestazioni causata dalla dipendenza tra istruzioni può essere mitigata con diversi accorgimenti.

- Esecuzione fuori ordine: si eseguono istruzioni non dipendenti, ancorchè non consequenziali
- Registri ombra: si memorizzano dati dipendenti su copie dei registri specificati dall'istruzione

# Mitigare le bolle

La perdita di prestazioni causata dalla dipendenza tra istruzioni può essere mitigata con diversi accorgimenti.

- Esecuzione fuori ordine: si eseguono istruzioni non dipendenti, ancorchè non consequenziali
- Registri ombra: si memorizzano dati dipendenti su copie dei registri specificati dall'istruzione
- Register renaming: il processore alloca registri diversi da quelli specificati dall'istruzione

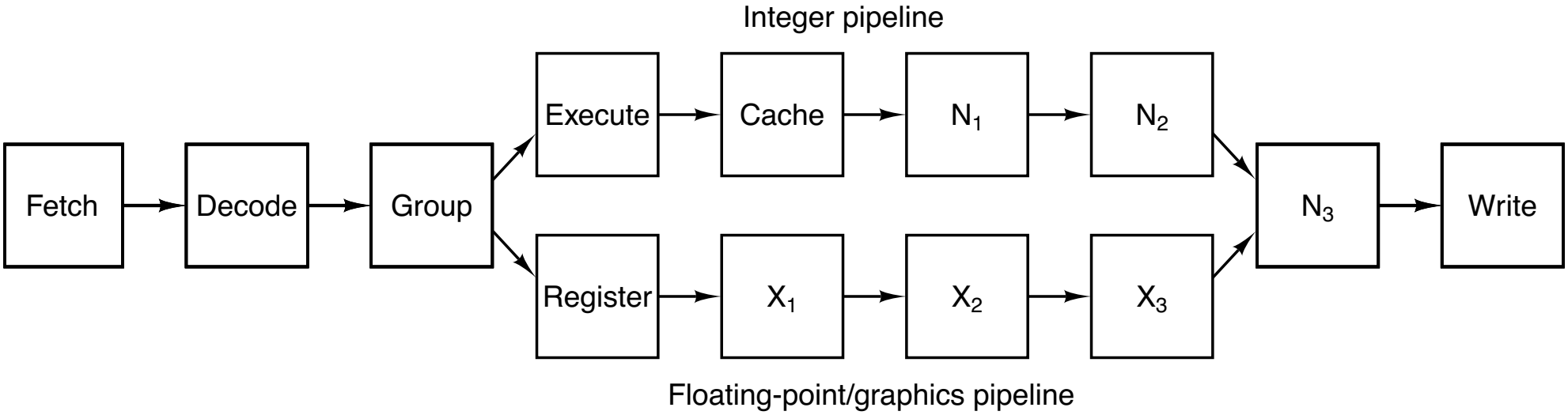
# Mitigare le bolle

La perdita di prestazioni causata dalla dipendenza tra istruzioni può essere mitigata con diversi accorgimenti.

- Esecuzione fuori ordine: si eseguono istruzioni non dipendenti, ancorchè non consequenziali
- Registri ombra: si memorizzano dati dipendenti su copie dei registri specificati dall'istruzione
- Register renaming: il processore alloca registri diversi da quelli specificati dall'istruzione
- Multi-threading (hyper-threading): si eseguono più programmi contemporaneamente. É necessario duplicare i registri. Primo passo verso i processori multicore.

# Es.: processore superscalare SPARC

Negi anni ’90 il processore Sun SPARC fu il primo a fare uso di pipeline specializzate.




Attualmente i processori possono eseguire decine di micro-istruzioni simultaneamente su 4-15 pipeline.

Oltre alla dipendenza tra istruzioni, l’esecuzione superscalare deve gestire efficacemente le istruzioni di salto condizionato. Es.: se $x > 0$ salta.

# Esecuzione di salti condizionati

Il processore impiega alcuni cicli di clock per valutare una condizione. Nel frattempo non sa se eseguire oppure saltare le istruzioni successive.
Strategie adottabili

- stall: la CPU non prosegue fino al completamento della valutazione della condizione; logicamente corretto ma inefficiente

# Esecuzione di salti condizionati

Il processore impiega alcuni cicli di clock per valutare una condizione. Nel frattempo non sa se eseguire oppure saltare le istruzioni successive.
Strategie adottabili

- stall: la CPU non prosegue fino al completamento della valutazione della condizione; logicamente corretto ma inefficiente
- predizione e speculazione: la CPU esegue istruzioni sotto condizione. L'esecuzione è annullata se la previsione si rivela errata.

In un programma ci sono numerosi salti condizionati. La predizione e speculazione di salto assicurano migliori prestazioni.

# Predizione del salto condizionato

Si adoperano due tecniche:

- predizione statica (decisa nel codice una volta per tutte)
  - semplice: le istruzioni che seguono un salto all'indietro sono sempre eseguite sotto condizione
  - ogni salto condizionato è accompagnato da un suggerimento generato da un profiler durante la compilazione o dal programmatore

# Predizione del salto condizionato

Si adoperano due tecniche:

- predizione statica (decisa nel codice una volta per tutte)
  - semplice: le istruzioni che seguono un salto all'indietro sono sempre eseguite sotto condizione
  - ogni salto condizionato è accompagnato da un suggerimento generato da un profiler durante la compilazione o dal programmatore
- predizione dinamica, tramite accesso a una history table mantenuta aggiornata con gli esiti dei precedenti salti condizionati.

# Speculazione sul salto condizionato

Invece della predizione, nell'esecuzione speculativa si eseguono entrambi i rami del codice che segue il salto condizionato.

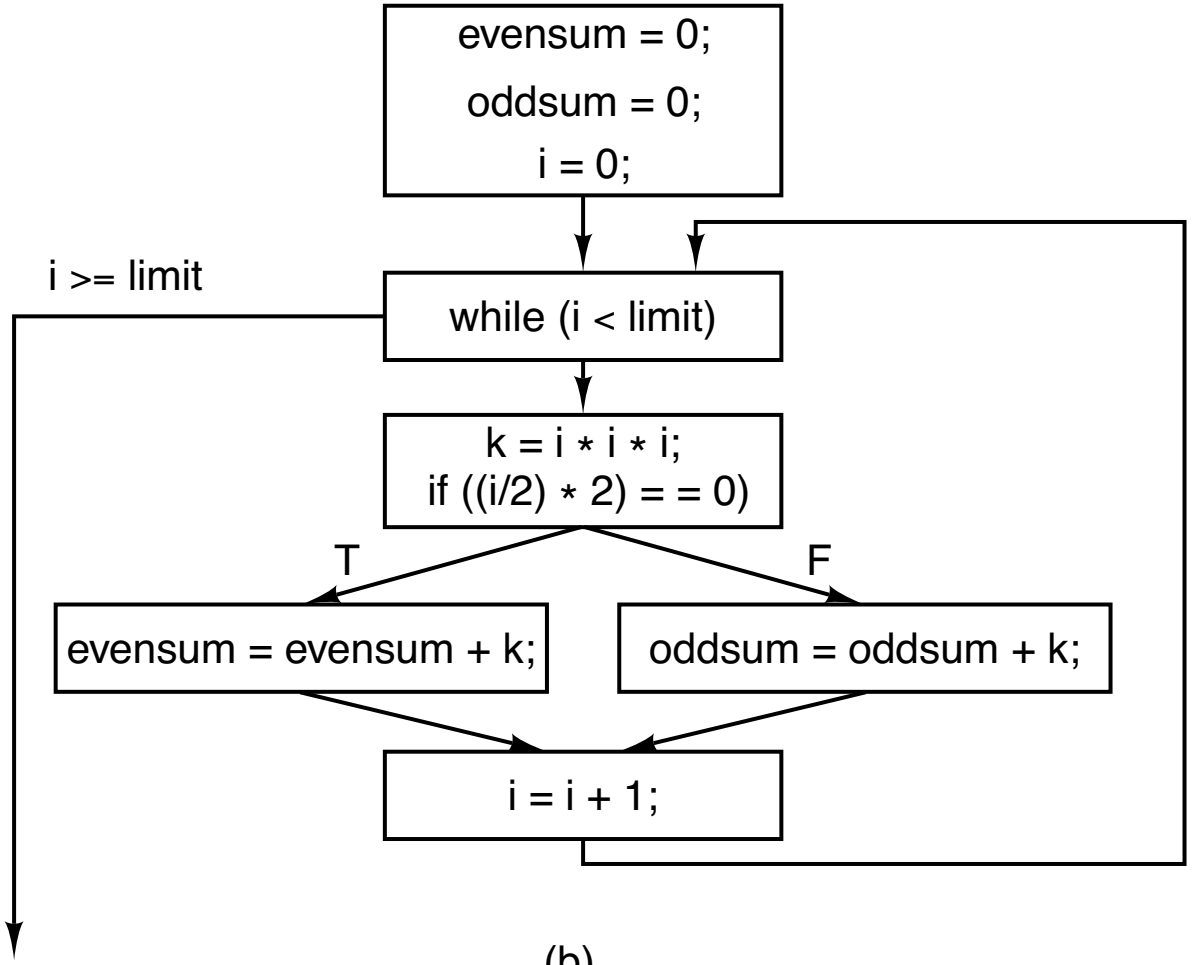
In questo caso si eseguono anche istruzioni che sicuramente verranno scartate. Perciò l'esecuzione

- dev'essere reversibile: utilizzo di registri ombra e marcatura (poison bit) di istruzioni che se eseguite generano effetti irreversibili
- deve evitare il *fetch* di istruzioni troppo onerose da eseguire: istruzione macchina *ad hoc* SPECULATIVE-LOAD.

# Esempio di esecuzione speculativa

```
evensum = 0;
oddsum = 0;
i = 0;
while (i < limit) {
    k = i * i * i;
    if ((i/2) * 2) == 0)
        evensum = evensum + k;
    else
        oddsum = oddsum + k;
    i = i + 1;
}
```

(a)




(b)

Il pre-caricamento delle variabili è suggerito dal compilatore. Il pre-salvataggio è sui registri-ombra.

Errore `if((i/2)*2 == 0)` (corretto nel libro di testo).

# Accesso alla memoria e caching

L'accesso alla memoria principale resta un'operazione troppo lenta: il processore deve attendere la conclusione del servizio anche per più di una decina di cicli di clock.

Per mitigare il problema si equipaggia l'architettura con uno o più livelli di memoria più veloce e costosa intermedi al processore e la memoria principale. Essi costituiscono la memoria cache.

L'efficacia della cache è strettamente legata a come sono selezionati dati e istruzioni che la occupano dinamicamente durante l'esecuzione. Trattenere in cache informazione non in uso è peggio che non disporre della cache!

# Cache multilivello

L'ottimizzazione del rapporto costo/prestazioni si ottiene prevedendo più livelli di cache.

Man mano che si sale di livello, la tecnologia scelta implicherà

- un'estensione di memoria maggiore
- una velocità più bassa
- un costo minore.

# Cache multilivello

L'ottimizzazione del rapporto costo/prestazioni si ottiene prevedendo più livelli di cache.

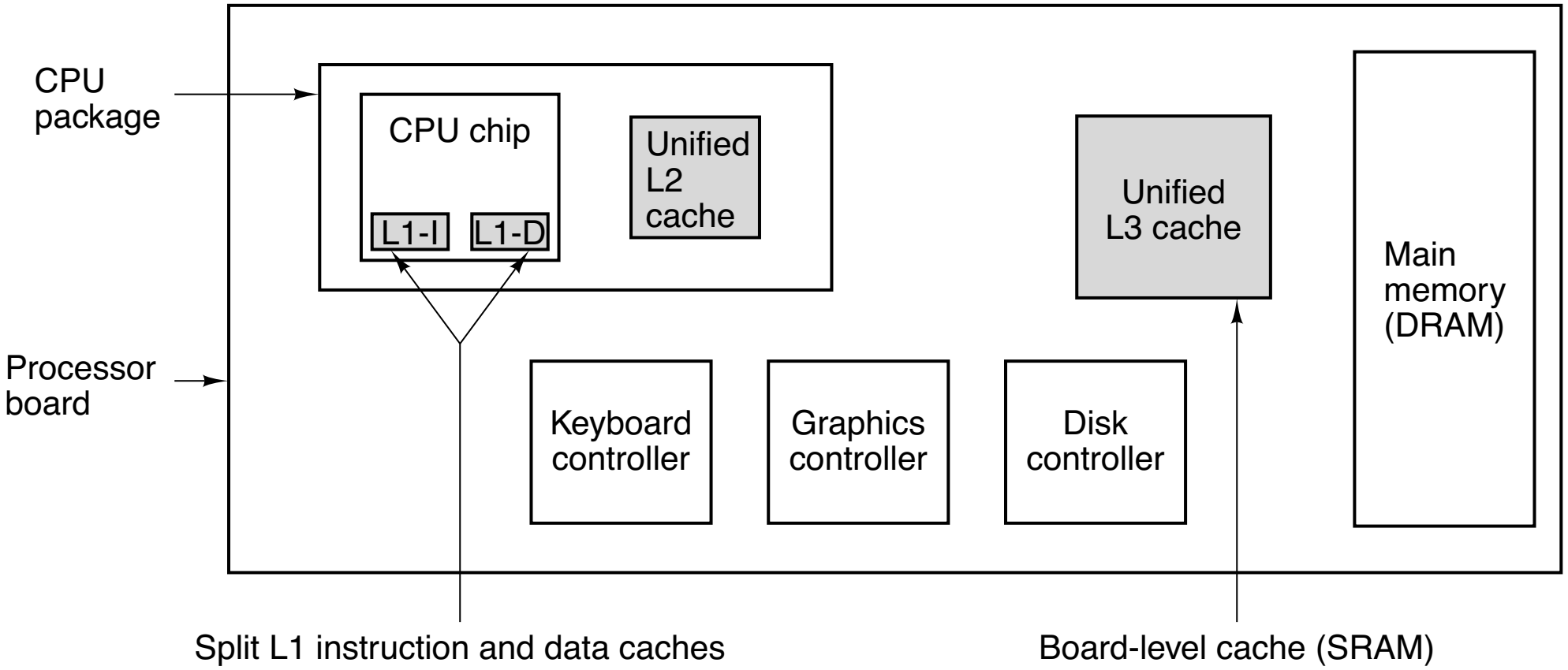
Man mano che si sale di livello, la tecnologia scelta implicherà

- un'estensione di memoria maggiore
- una velocità più bassa
- un costo minore.

Tipicamente sono presenti 3 livelli di cache.
Es.: Intel Core i7 (Sandy Bridge) ha 32 KB di livello 1, 256 KB di livello 2, 1-20 MB di livello 3.

# Cache split

In più, le moderne cache di primo livello sono divise in due parti dedicate rispettivamente a contenere dati e istruzioni.



# Prestazioni processori superscalari

L'efficienza di un processore superscalare dipende principalmente dai seguenti fattori:

- percentuale delle istruzioni che non vengono bloccate causa dipendenze
- percentuale di predizioni di salto corrette
- percentuale di accessi alla memoria senza uscire dalla cache.

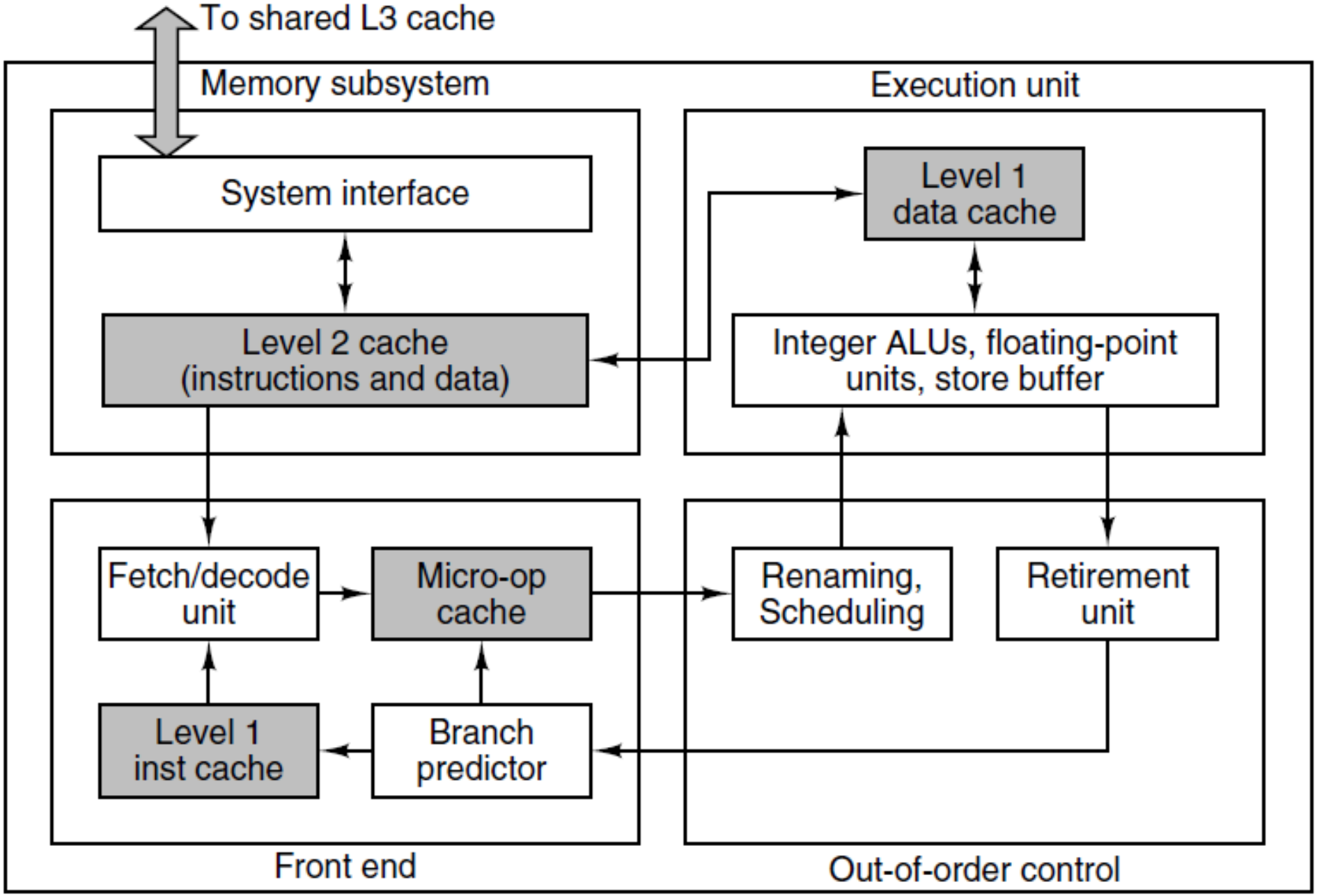
Queste percentuali sono difficilmente valutabili a priori in quanto dipendono dal programma eseguito. Esse sono quindi valutate attraverso test condotti su opportuni programmi di test (benchmarking).

# Il chip Intel Core i7 (Sandy Bridge)

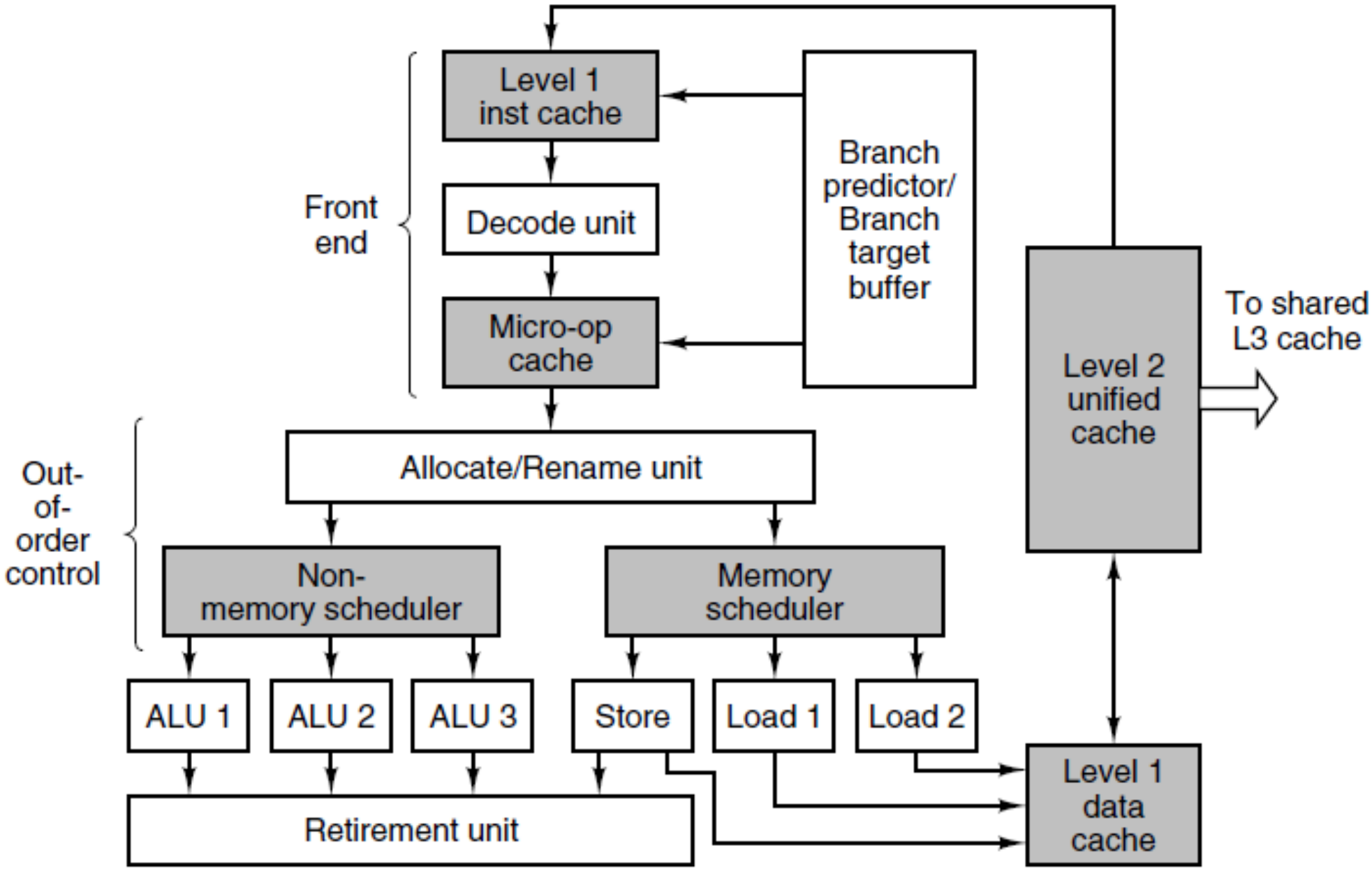
Intel i7 è un processore multicore (più CPU nello stesso chip) in grado di distribuire l'esecuzione di uno o più programmi simultaneamente sulle CPU (hyper-threading):

- i primi stadi della pipeline traducono codice CISC in istruzioni RISC
- le istruzioni RISC sono depositate nella cache di livello 0 (L0)
- predizione di salto sofisticata (algoritmo segreto)
- la cache L3 è condivisa dai core
- il processore grafico, integrato nel chip, implementa istruzioni vettoriali AVX (Advanced Vector Extensions).

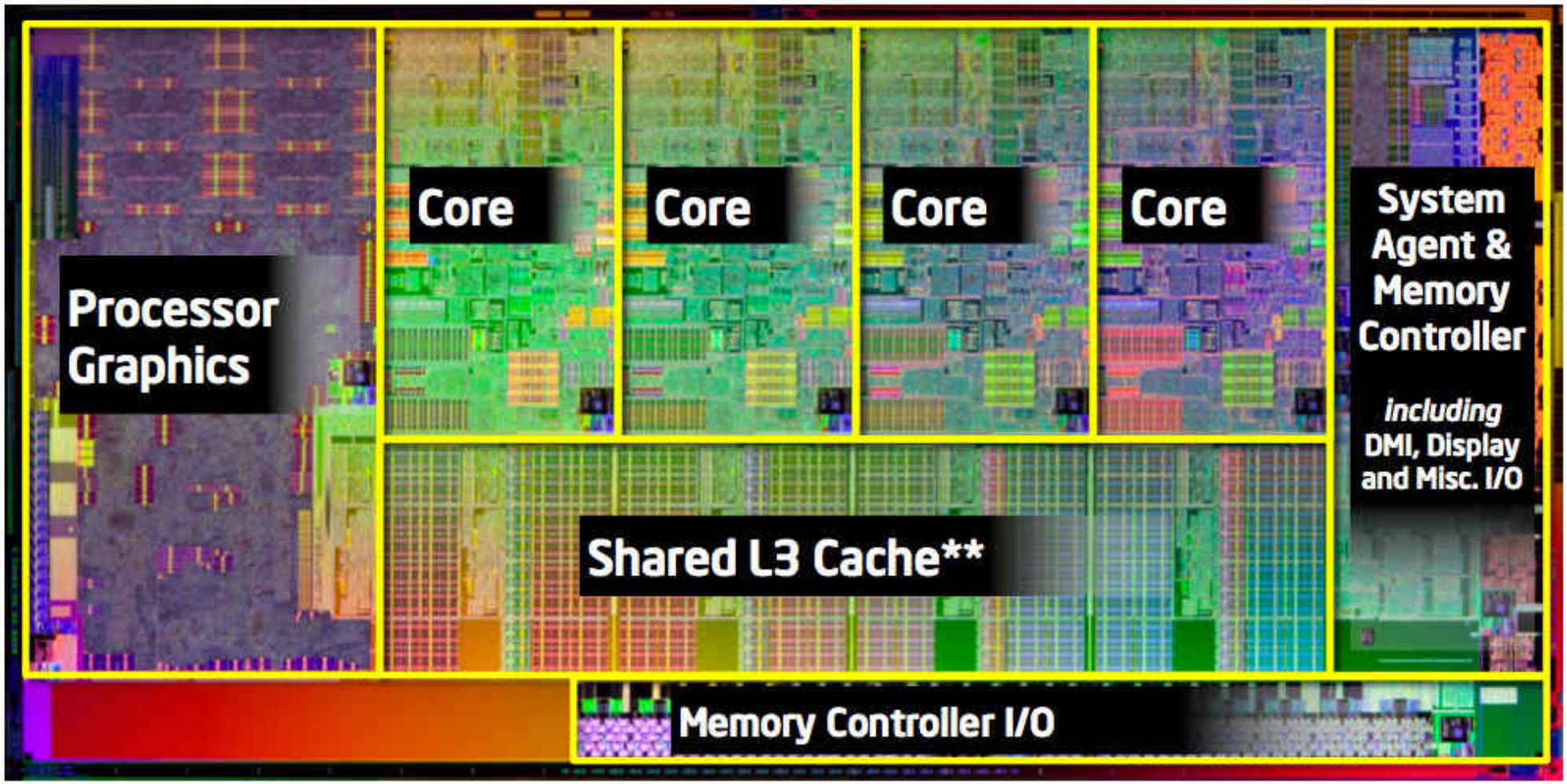
# Architettura core Sandy Bridge



# Pipeline core Sandy Bridge



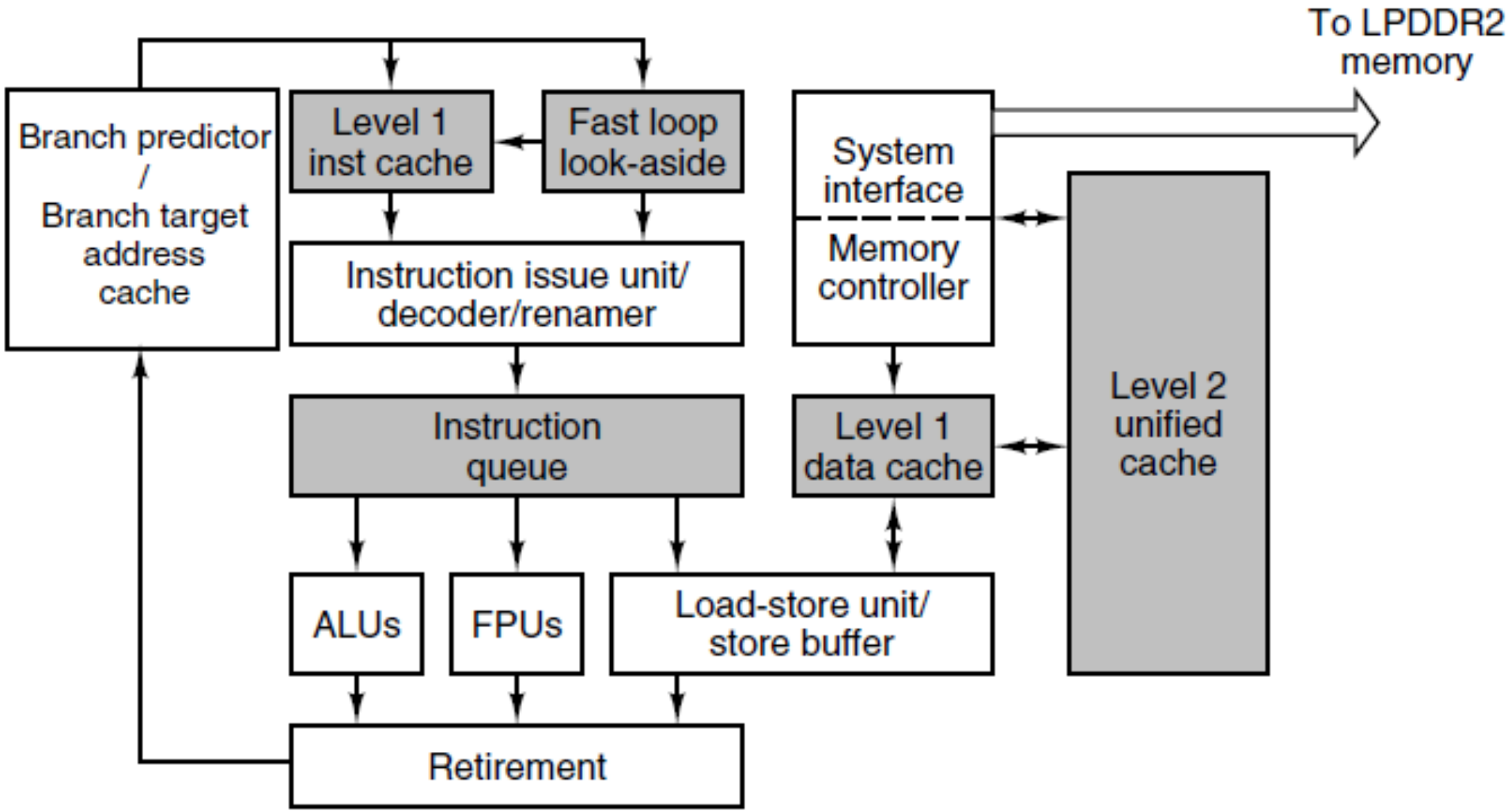
# Chip Sandy Bridge



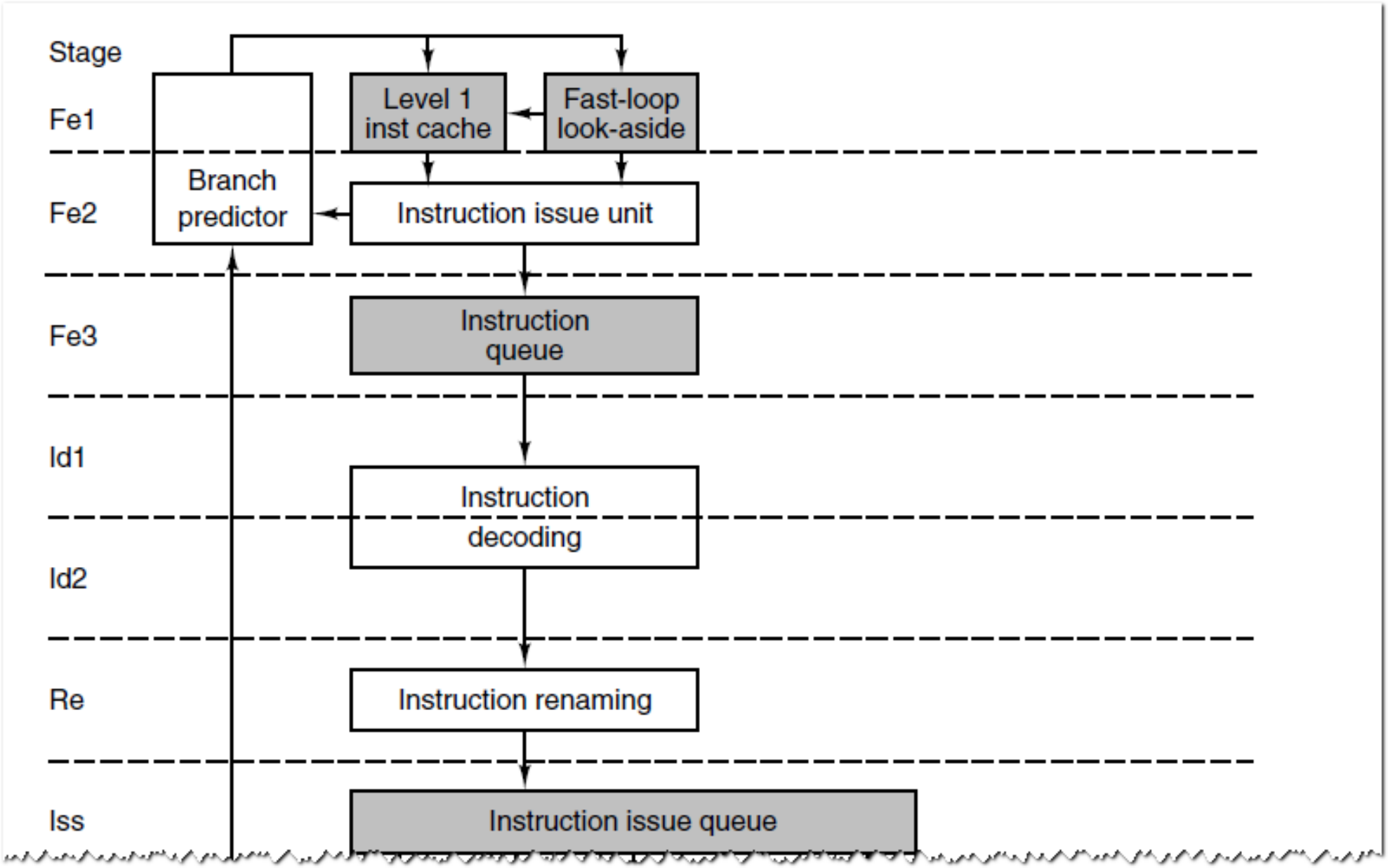
# Il chip ARM Cortex A9

- Progetto Acorn realizzato messo su chip da vari costruttori (Motorola).
- Core integrato in System on Chip (SoC, più componenti su un singolo chip).
- Implementa le istruzioni macchina ARM v7.
- Strutturalmente abbastanza simile al Core i7.
- Manca lo stadio iniziale di traduzione, in quanto nasce come macchina RISC.

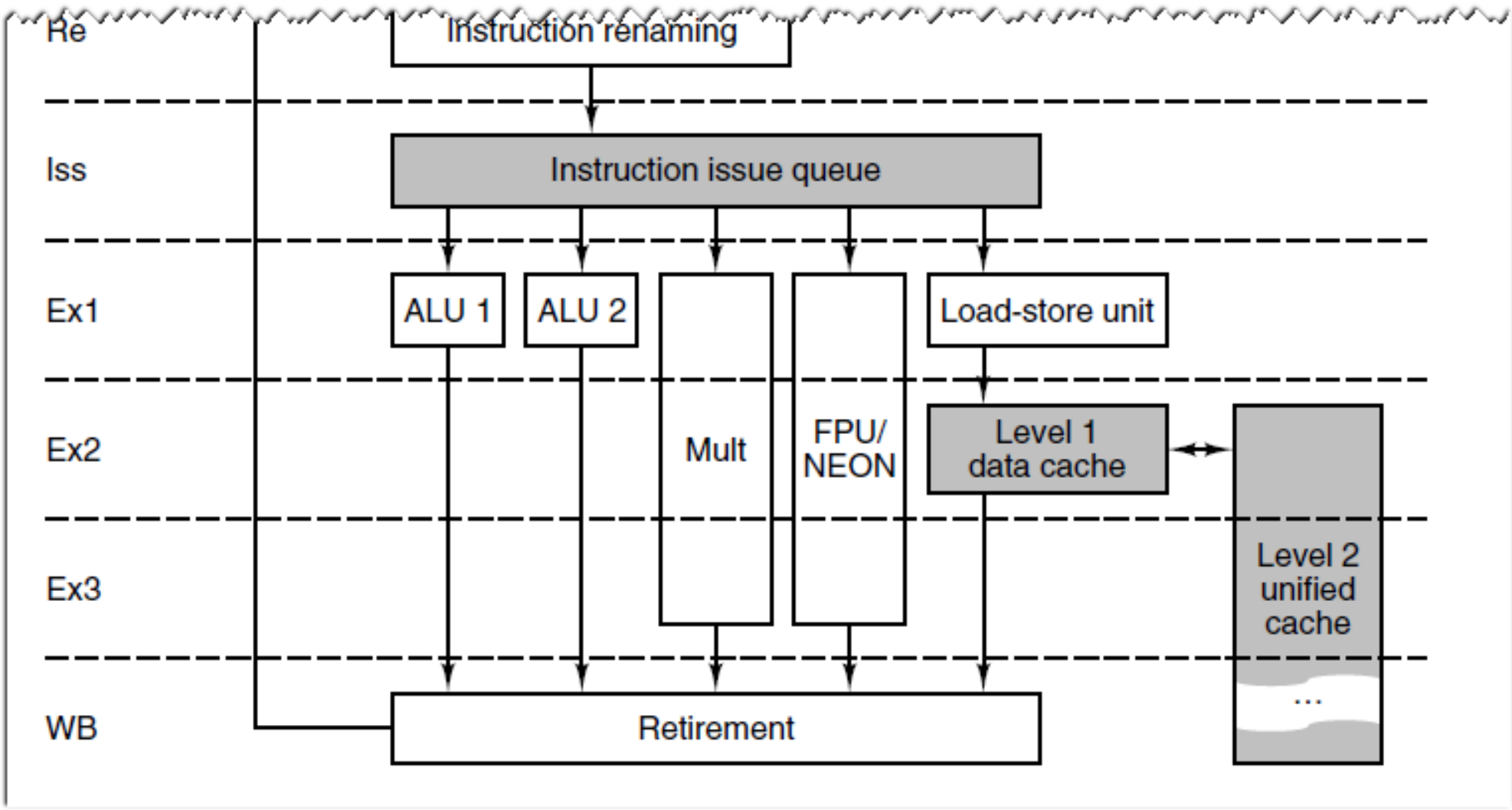
# Architettura Cortex A9



# Pipeline Cortex A9



# Pipeline Cortex A9



# Il chip ATmega168