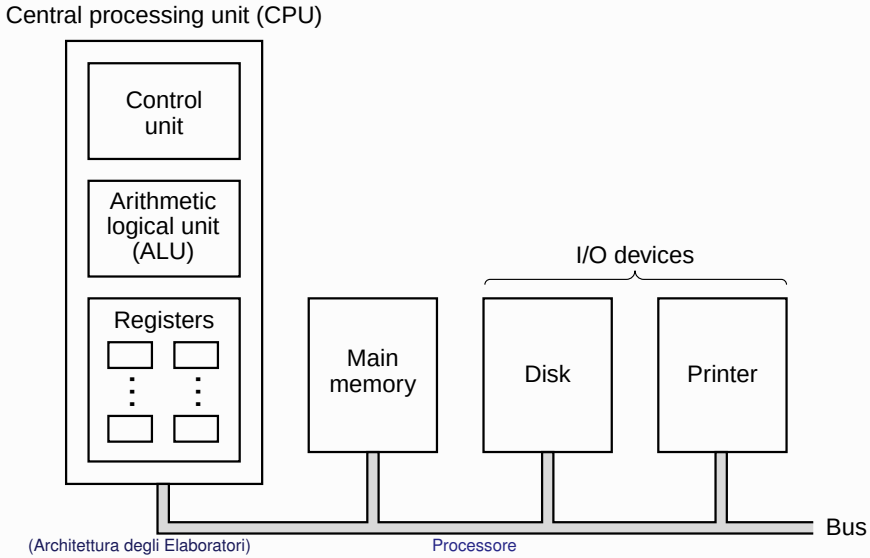
Processore

Presentazione a livelli del calcolatore:
dopo porte logiche, circuiti base e memorie, esaminiamo il:

Processore (CPU - Central Processing Unit): cuore del calcolatore, l'unità che esegue le istruzioni macchina (cap. 2 e 4).

Nel seguito le altre componenti del calcolatore: bus, I/O, memoria di massa.

Struttura schematica di un calcolatore

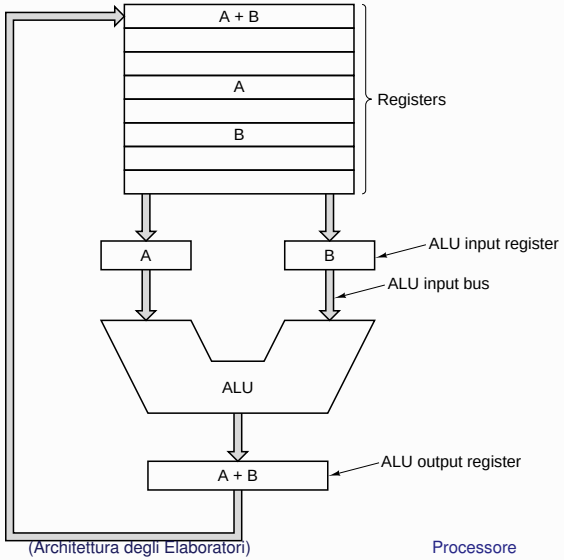


Il processore (CPU)

Compito del processore: eseguire il ciclo fetch-decode-execute;

- fetch: preleva un'istruzione dalla memoria istruzione macchina;
- decode: determina il tipo di istruzione e i suoi argomenti;
- execute: esegui l'istruzione:
  recupera gli argomenti, esegui un operazione, memorizza i risultati;
- ripete il ciclo.

Componenti processore: Data Path



Data Path

Formata da:

- una serie di registri (memorie),
- un'unità aritmetica e logica (ALU),
- dei bus di collegamento.

Può eseguire micro-operazioni.

Micro-operazione

L'operazione eseguibile dal data path in un singolo ciclo di clock.

Azioni eseguibili da una micro-operazione:

- una singola operazione aritmetica-logica (i cui argomenti e risultato risiedono nei registri).
- una comunicare con la memoria (richiesta di lettura o scrittura di una locazione)

Un'istruzione macchina viene eseguita mediante una o più micro-operazioni.

Unità di controllo

Il funzionamento del data path viene regolato, mediante segnali, dall'unità di controllo.
Circuito sequenziale che regola il funzionamento del processore.

- Esamina l'istruzione corrente, contenuta Instruction Register (IR).
- Invia segnali di lettura e scrittura ai registri.
- Invia segnali di controllo alla ALU.
- Gestisce la comunicazione con la memoria principale.

Realizzazione

Due alternative:

- cablata: si realizza un circuito sequenziale classico;
- micro-programmata: il controllo è un piccolo calcolatore capace di eseguire un micro-programma.

Dicotomia: cablata - programmata

Le stessa funzionalità possono essere implementate mediante:

- hardware (logica cablata): più complicata e costosa da realizzare, ma offre prestazioni migliori;
- software (logica programmata): più semplice e flessibile, ma più lenta.

Diversi esempi di questa dicotomia: le istruzioni grafiche, l'elaborazione dei segnali, ...

Un po' di storia

I primi processori: poche istruzioni, logica cablata.

Anni 50:

- prime unità di controllo micro-programmato, permette la costruzione di calcolatori economici ma con un ricco insieme di istruzioni.
- calcolatori con maggiori presazioni usano la logica cablata.
- Si costruiscono calcolatori molto diversi per prestazioni e costi con lo stesso insieme di istruzioni. (IBM 360)

Anni 70

Le potenzialità della micro-programmazione vengo sfruttate al massimo.

- Linguaggi macchina sempre più sofisticati, vicini ai linguaggi di programmazione standard.
- Il calcolatore apice di questa filosofia: VAX (Digital DEC).
- Motivazioni tecnologiche: all'epoca i control-store (la memoria micro-programma) sono molti più veloci della memoria principale (RAM).

Anni 80, processori RISC

Si cambia rotta: poche istruzioni, logica cablata

- le istruzioni complesse non sono così utili
- istruzioni semplici molto più veloci;
- si possono utilizzare controlli cablati;
- la velocità della RAM si avvicina a quella del control-store

I primi RISC

Reduced Instruction Set Computer

- 801 (IBM),
- CPU-RISC (Patterson - Berkley) $\Rightarrow$ Sparc,
- MIPS (Hennesey - Stanford),
- Alpha (Digital), PowerPC (Motorola, IBM), ARM,

Anni 90, diatriba: RISC – CISC

Complex Instruction Set Computer.

- Tutti i processori di nuova concezione sono RISC.
- Ma i processori usati nei PC: architettura Intel x86, (IA-32), sono CISC.
- Motivi: compatibilità con il software preesistente, non si vogliono riscrivere i programmi (il codice).

Attualmente

La contrapposizione RISC – CISC è più sfumata.

- La legge di Moore ha portato alla creazione di processori RISC con insiemi di istruzioni sempre più ampi.
- Il processori CISC (IA-32) usano al loro interno un cuore RISC,
  - istruzioni semplici, più comuni: eseguite direttamente,
  - istruzioni più complesse: scomposte in più istruzioni semplici,
  - istruzioni sofisticate: eseguite mediante micro-programma.

RISC – CISC

- In linea di principio, i processori RISC sono preferibili,
- le architetture CISC hanno uno svantaggio del (20-30%),
- per i processori x86, lo svantaggio è compensato dalle economie di scala (vengono prodotti in gran numero).

I processori nel libro di testo

Per illustrare con maggior dettaglio il funzionamento dei processori,

si mostra un esempio concreto di progettazione di un processore.

Più precisamente: si mostra come costruire processore (Mic)
che eseguire istruzioni del Java bytecode:
si realizza la Java Virtual Machine (JVM)

Approccio per esempi e bottom-up: prima descrizione delle componenti, poi quadro di insieme.

I processori nel libro di testo

- Pro: un esempio pratico, consistente con il resto.
- Contro: presentazione lunga, molti dettagli tecnici, difficile comprensione, linguaggio macchina non standard.
- A lezione presentazione più superficiale: descrizione dei principi, pochi dettagli. Argomenti affrontati in un ordine diverso.
- Problema: meno correlazione col libro di testo, non esiste un insieme di pagine che sia completo, autosufficiente e non sovrabbondante (molti più argomenti di quelli presentati a

Premessa: i compilatori

Un programma ad alto livello (Java, C, C++, Scheme, Pascal, . . .) deve essere tradotto in linguaggio macchina (istruzioni eseguibili dal calcolatore).

Due alternative:

- compilatore: programma traduttore, dal programma *sorgente* genere un programma macchina equivalente.
- interprete: programma interprete, legge il programma sorgente e lo esegue direttamente (non genera codice intermedio).

Java

Java si propone come il linguaggio per le applicazioni in rete.
I programmi Java devono poter migrare nella rete:

- codice compilato non ha questa possibilità, è specifico ad una particolare architettura (processore – sistema operativo),
  inoltre: problemi di sicurezza;
- il codice sorgente non può essere spostato in maniera efficiente.

Java Virtual Machine, JVM

Soluzione:

- viene definito un codice intermedio (tra Java e il linguaggio macchina): Java bytecode,
- viene definita un macchina virtuale la Java Virtual Machine (JVM),

  capace di eseguire programmi scritti in Java bytecode,

  macchina virtuale: implementata, via software, su diverse piattaforme (processore, sistema operativo),

  livello Java bytecode

Java bytecode

Programmi Java compilati in Java bytecode, eseguibile da ogni calcolatore mediante la Java Runtime Environment (JRE), composto da:

- JVM (programma interpretare il Java bytecode);
- librerie.

Vantaggi, svantaggi,

- vantaggi: codice universale, compatto, con meccanismi di protezione;
- svantaggi: minore efficienza rispetto alla compilazione diretta (mitigata dai compilatori just-in-time).

Processori Mic (1,2,3,4)

Implementazione hardware, cablata (non virtuale o programmata) della JVM.
Mic è un processore capace di eseguire un sottoinsieme del Java bytecode.
Progetto didattico, non esistono implementazioni.

Esistono delle vere implementazioni, de

- picoJava (Sun Microsystems),
- ARM926EJ-S, (architettura ARM estesa con il Java bytecode),

Java bytecode

Simili, per molti aspetti, ai linguaggi macchina.

Alcune differenze importanti:

- meccanismi di protezione contro codice malevolo;
- compatto, istruzioni lunghe un byte (+ un eventuale argomento di 1-2 byte);
- primitive object-oriented (chiamate dei metodi).

Per semplicità, i processori Mic implementano un piccolo sottoinsieme del Java bytecode.

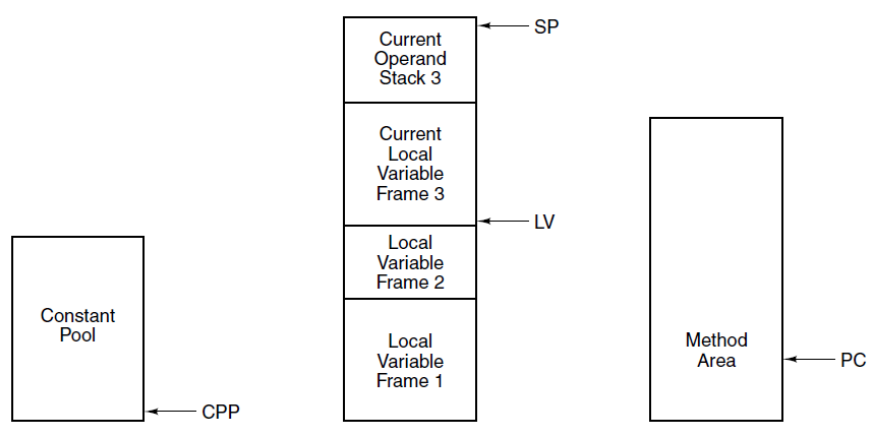
Il modello di memoria

Memoria standard processori, monolitica, uno stesso spazio per programmi e dati.

Memoria della JVM è divisa in quattro parti.

- area del codice (programmi)
- area delle costanti (dati utilizza dai programmi)
- stack delle variabili locali
- stack degli operandi (dati su cui eseguire operazioni)

Il modello di memoria



Stack delle variabili

- Usato per gestire le chiamate di procedura (metodi).
- Ogni procedura ha le sue variabili, spazio di memoria.
- Questi spazi gestiti come una pila.
  Ad ogni chiamata di procedura si alloca uno spazio.
  Spazio recuperato quando la procedura termina.

Stack degli operandi

- JVM non è una macchina a registri (come quasi tutti i processori).
  Le operazioni aritmetiche logiche non fanno riferimento ai registri interni, ma ad uno:
- Stack degli operandi. Pila su cui:
  - inserire o prelevare word, (sequenze di 32 bit)
  - eseguire operazioni sugli ultimi dati inseriti.

Esempio la JVM valuta l'espressione
$(3 + 5) \times (4 + 2)$ in ...

I-Java bytecode

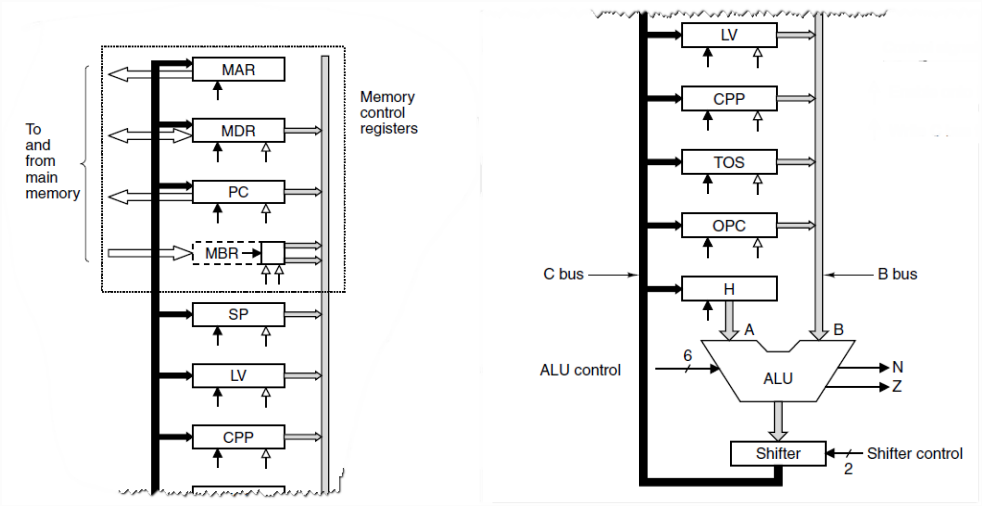
- Operazioni aritmetiche e logiche: IADD, IAND, IOR, ISUB, IINC vn con.
- Operazioni trasferimento dati, da e per la memoria: ILOAD vn, ISTORE vn, LDC_W i, (DUP, POP, SWAP, BIPUSH b).
- Operazioni per il controllo del flusso dell'esecuzione:
  GOTO, IFEQ os, IFLT os, IF_ICMEQ os,
  Chiamate di metodi: INVOKEVIRTUAL d, IRETURN,
- Altro: NOP, WIDE

Il processore Mic-1

Un semplice processore per il Java bytecode.
Funzionamento:

- ad ogni ciclo di clock il data path può eseguire semplici operazioni, o spostare dati (micro-istruzione)
- un'istruzione del Java bytecode viene realizzata mediante una sequenza di micro-istruzioni,
- sequenza composta da diverse parti: fetch-decode-execute.

Mic-1: data path



Unità aritmetica-logica (ALU)

Componente di ogni processore,
può eseguire le operazioni aritmetiche e logiche:

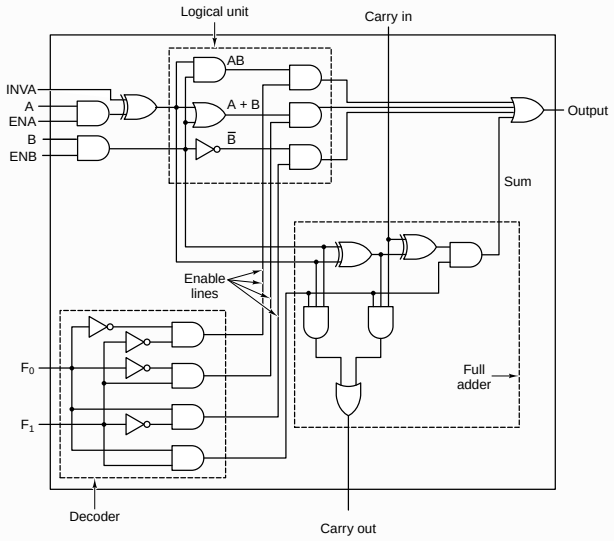
- Ingressi:
  - due argomenti: 32, 64 bit;
  - codifica operazione da eseguire.
- Uscite:
  - risultato: 32, 64 bit;
  - bit di condizione: segno, zero, overflow.

Unità aritmetica-logica (ALU)

MIC1 usa una semplice ALU:

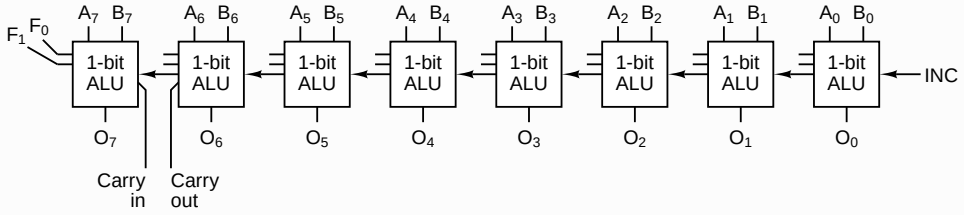
- implementa poche operazioni: AND, OR, NOT, somma, sottrazione, incremento, decremento 1, opposto.
- Operazioni eseguite bit per bit (locali): ALU scomposta in unità elementari, ciascuna operante su una coppia di bit.
- segnali di controllo: selezionano l'operazione da svolgere, abilitano o meno gli ingressi, forniscono il riporto per le cifre meno significative.

ALU da un bit



ALU da 8-32 bit

ALU completa formata da un sequenza di ALU a 1 bit



Mic 1 contiene una ALU a 32 bit.

Funzione calcolabili dalla ALU

Oltre alle ovvie:

- $A + B + 1$, somma con *Carry in* $= 1$
- $A + 1$, somma con $ENB = 0$, *Carry in* $= 1$
- $B - A$, somma con $INVA = 1$, *Carry in* $= 1$
- $B - 1$, somma con $ENA = 0$, $INVA = 1$
- $-A$, somma con $ENB = 0$, $INVA = 1$, *Carry in* $= 1$

Registri (specializzati)

- MAR: Memory Address Register,
  MDR: Memory Data Register;
  comunicazione memoria dati;
- PC: Program Counter,
  MBR: Memory Bytecode Register;
  comunicazione memoria codice;
- SP: Stack Pointer, TOS: Top Of Stack;
  stack degli operandi;
- LV: Local Variable; puntatore stack variabili;
- CPP: Constant Pool Pointer; punt. costanti;
- OPC, H: registri ausiliari.

Esempio di implementazione

L'istruzione IADD viene realizzata dalla sequenza di micro-istruzioni:

- Main1: PC = PC + 1; fetch; goto (MBR)
- iadd1: MAR=SP=SP-1; rd
- iadd2: H = TOS
- iadd3: MDR=TOS=MDR+H; wr; goto Main1

Le altre istruzioni implementate in maniera analoga.

Circuito di controllo

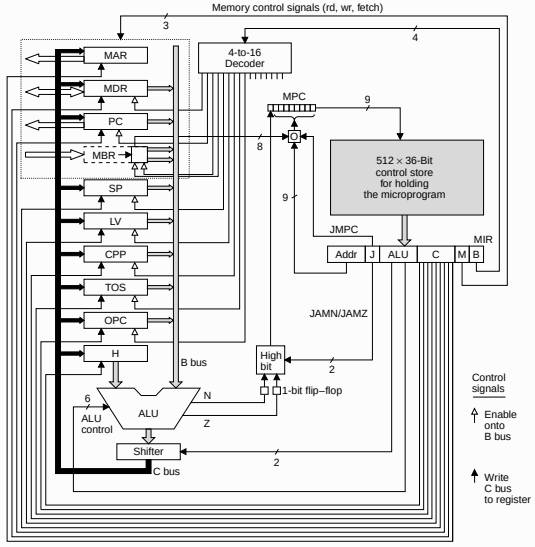
Micro-programmato. Formata da memoria ROM (contiene il micro-codice), 2 registri, un multiplexer.

- semplice sia il circuito che la progettazione
- relativamente lento

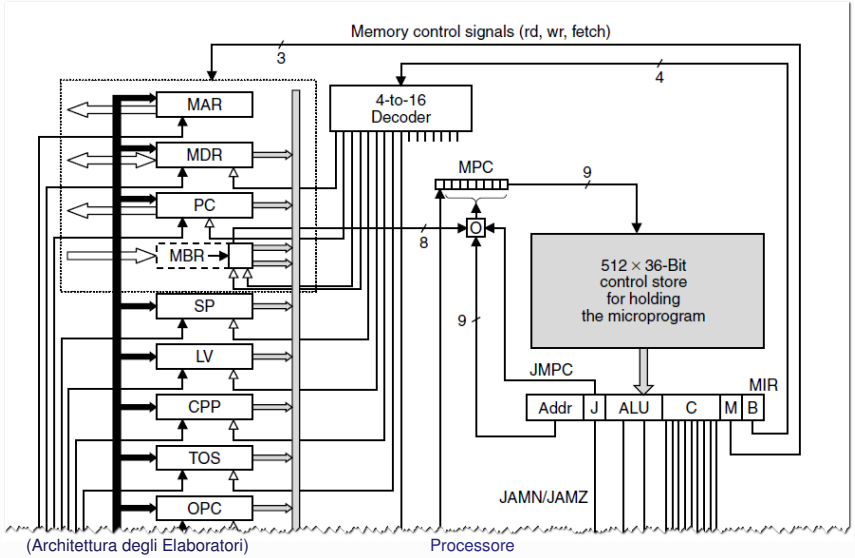
Micro-istruzioni di 36 bit

- in parte segnali di controllo,
- in parte determinano la prossima micro-istruzione,

Circuito di controllo



Circuito di controllo: Control store



Circuito di controllo

Invia:

- ai registri i segnali di lettura e scrittura;
- alla memoria le istruzioni: read, write, fetch;
- alla ALU il codice dell'istruzione da eseguire.

Determina micro-istruzione successiva via:

- MBR (prima micro-istruzione nelle esecuzione di un istruzione macchina);
- la micro-istruzione corrente (micro-istruzioni successive);
- in alcuni casi, il primo bit dell'indirizzo determinato dall'uscita della ALU (per poter implementare i salti condizionati).

Prestazioni

MIC-1, processore (relativamente) completo ma inefficiente (molto più semplice dei processori attuali).

Prestazioni: computazione eseguibile nell'unita di tempo.

Miglioramento delle prestazioni

Due metodi:

- Utilizzare istruzioni macchina più potenti, e flessibili.
  Esempi:
  - Processori Intel x86: 8086 (1978) – 80286 – 386 (1985) – MMX — 3DNow — x86-64 (2000) — AVX;
  - ARM: ARMv1 — . . . — ARMv8;
  - MIPS: R2000 — R3000 — R4000;
  - IJVM $\rightarrow$ JVM.
- Eseguire più istruzione nell'unità di tempo

Più istruzioni al secondo

Il funzionamento della CPU scandito da un clock,

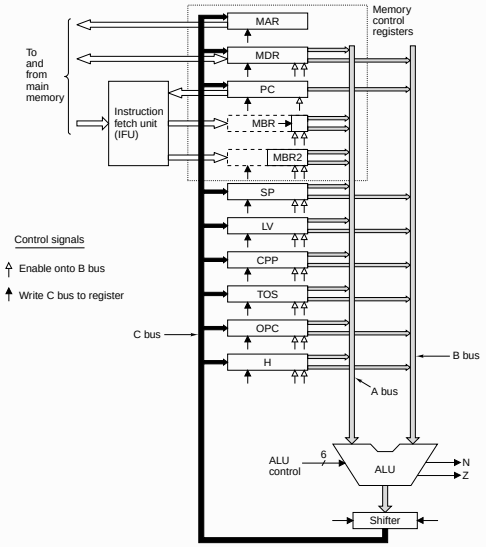
- diminuire il numero di micro-istruzioni (cicli di clock) necessari per eseguire un'istruzione macchina;
  aumentare la computazione svolta nel ciclo di clock:
- ridurre il ciclo di clock.

Più computazione per ciclo di clock

Aumentare la potenza di calcolo del data path, micro-istruzioni più potenti.

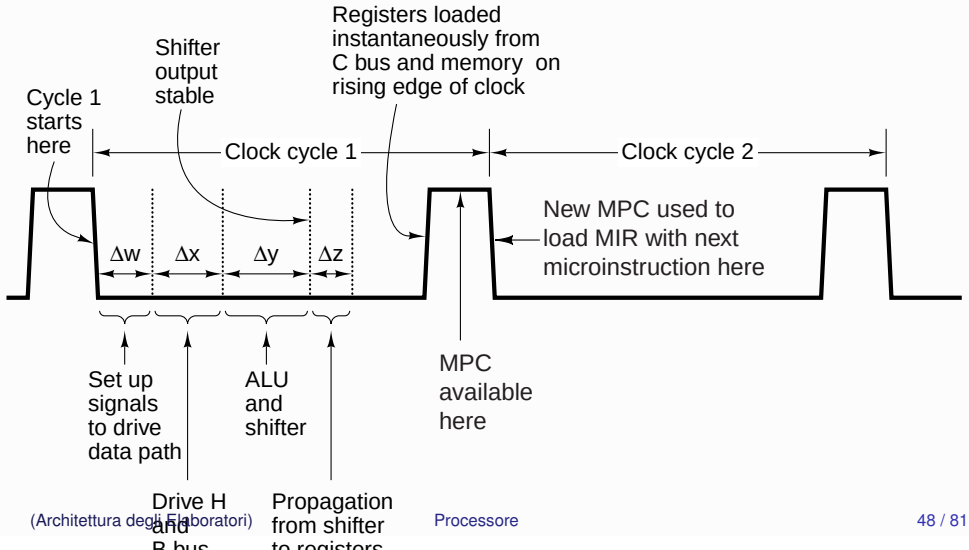
- ALU con più operazioni,
- più registri disponibili,
- più possibilità di scambio dati (3 bus distinti)
- un'unità separata per il caricamento delle istruzioni: Instruction Fetch Unit (IFU)

Secondo progetto MIC-2



Ridurre il ciclo di clock

Segnale di clock abilita la scrittura dei registri.
Periodo di clock dato dalla somma dei ritardi.



Ridurre il ciclo di clock

Due modi:

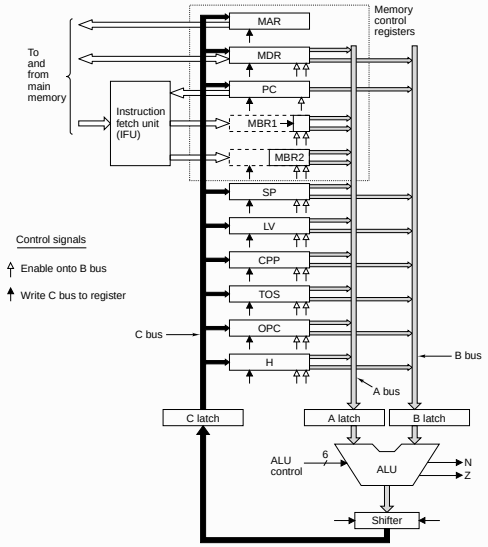
- migliorare la tecnologici nei circuiti integrati: transistor più veloci,
- migliorare la strutture dei circuiti: realizzare circuiti con meno ritardi (a parità di tipo di transistor impiegati).

Architettura più veloce

Velocizzare il data-path:

- circuito di controllo cablato,
- ALU e circuiti combinatori più veloci, (un intero settore di ricerca):
  l'ALU presentata nel testo particolarmente inefficiente.
- esecuzione parallela, spezzare l'esecuzione della micro-operazione in più stadi: la tecnica della pipeline

Terzo progetto: Mic-3



Mic-3 – Pipeline

Pipeline:

- l'esecuzione della micro-operazione viene divisa in più stadi
- ogni stadi eseguito più rapidamente
- i diversi stadi eseguiti in contemporanea su più istruzioni

Pipeline – Paragoni

Possibile paragone preso dalla vita comune, una lavanderia:

vestiti da lavare	micro-operazioni
lavaggio	acquisire i dati di ingresso
asciugatura	calcolo del risultato
stiratura	memorizzazione del risultato

Le varie fasi di lavoro sono portata avanti in parallelo.

La pipeline, secondo paragone.

La catena di montaggio:

il lavoro viene scomposto in fasi, ognuna eseguita da un dispositivo specifico,

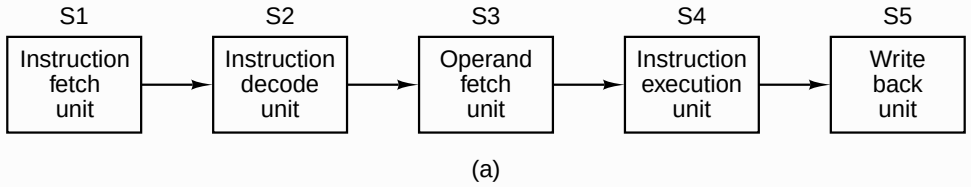
si svolgono le diverse fasi, su componenti diverse, in parallelo.

In un processore si parallelizza l'esecuzione delle micro-operazioni, (ciclo di clock);

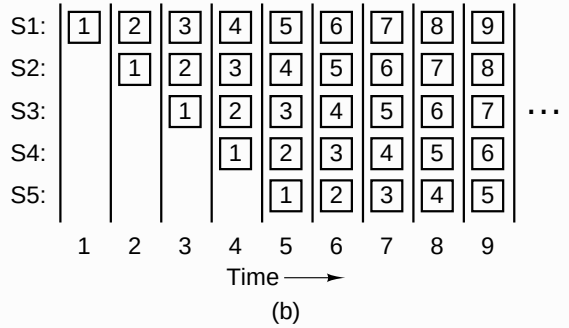
si migliora la banda passante ma non i tempi di risposta.

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

Esempio di scomposizione:

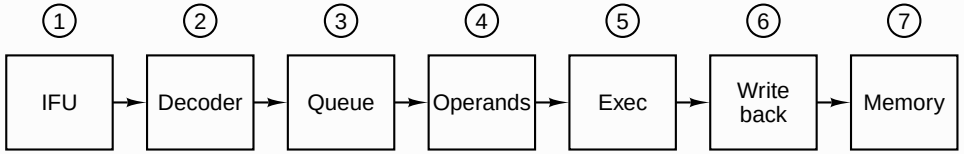


(a)



(b)

Altro esempio:

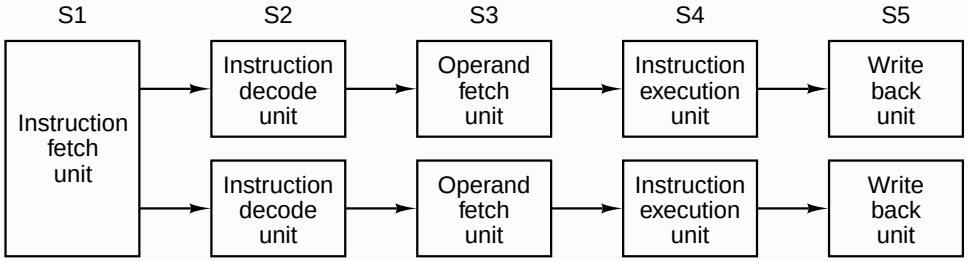


Processori diversi usano scomposizioni, strutture della pipeline, diverse.
Lunghezza tipica di una pipeline: 7–14 stadi.
Caso limite Pentium IV: 20 stadi di pipeline (guerra dei GHz).

Processori superscalari

Aumentano ulteriormente il parallelismo:
si migliora il rapporto $\frac{istruzioni}{cicli\ di\ clock}$,

Iniziano contemporaneamente l'esecuzione di più istruzioni: più pipeline operanti in parallelo.

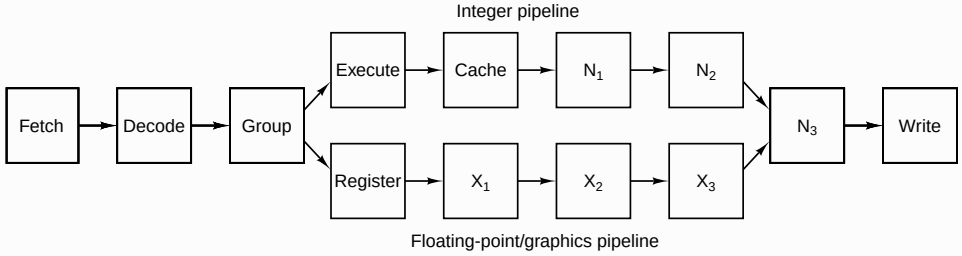


Processori superscalari

- I primi stadi (singoli) prelevano più istruzioni dalla memoria, e le decodificano.
- Le istruzioni smistate su stadi successivi multipli.
- Spesso, uno stadio finale singolo che termina, in ordine, le istruzioni.

Esempio: processore SPARC

Pipeline specializzate.



Attualmente processori con 4-15 pipeline, decine di micro-istruzione in contemporanea.

Problemi del parallelismo

Processori superscalari possono, potenzialmente, eseguire decine di istruzioni contemporaneamente.

Due fenomeni impediscono un completo sfruttamento del parallelismo.

- Dipendenza tra istruzioni.
- Istruzioni di salto.

Dipendenza tra istruzioni:

In un programma le istruzioni sono stato pensate per essere eseguite in ordine.

Un'esecuzione parallela, senza controlli, può portare a risultati scorretti. Tre casi:

- 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
```


Dipendenza tra istruzioni

Le dipendenze vengono rilevate mediante una tabella delle dipendenze (scoreboard):
memoria interna al processore che conta
per ogni registro,
le operazioni, di lettura e scrittura, in sospeso su quel registro.

Le istruzioni dipendenti devono essere sospese:

decadimento delle prestazioni, eseguiamo meno operazioni di quelle teoricamente possibili.

Si creano bolle, zone inattive, nella pipeline.

Gestire la dipendenza tra istruzioni

Tecniche per recuperare le prestazioni perse:

- esecuzione fuori ordine: si mandano in esecuzione le istruzioni non dipendenti;
- registri ombra: si usano copie di registri su cui memorizzare temporaneamente i dati;
- register renaming: si usano nuovi registri, non specificati dal codice.
- multi-threading (hyper-threading): si eseguono più programmi contemporaneamente, necessario duplicare i registri. Primo passo verso processori multicore.

Istruzioni di salto condizionato

Problematiche per i processori con pipeline:

- il processore impiega alcuni cicli di clock per valutare la condizione,
- nel frattempo, non sa quali istruzioni eseguire.

Due possibili soluzioni:

- stall non si inizia alcuna istruzione: corretta ma con decadimento delle prestazioni;
- si fa una predizione di salto: taken, not taken, si inizia l'esecuzione condizionata di alcune istruzioni:
  esecuzione annullata se la previsione si rivela errata

Tecniche per la predizione di salto

Due classi:

- predizione statica, sul codice:
  - semplice: si eseguono i salti indietro,
  - suggerite dal compilatore, programmatore: istruzioni di salto con suggerimento,
- predizione dinamica, sull'esecuzione: viene usata una history table (ricorda il comportamento passato di alcune istruzioni di salto, poche istruzioni, pochi bit),

In un programma numerose istruzioni di salto: una buona predizione di salto è fondamentale per le prestazioni.

Esecuzione Speculativa:

nel caso di salto condizionato,
non si tenta la predizione ma
si eseguono entrambe le possibili continuazioni del programma.

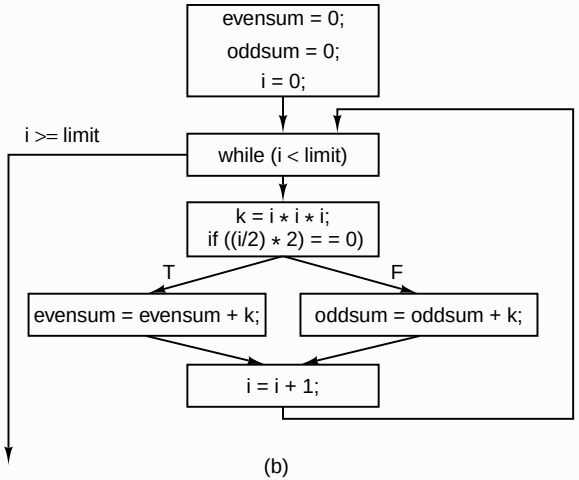
Pur di alimentare il processore, si eseguono anche alcune istruzioni che sicuramente verranno scartate.
Problemi:

- l'esecuzione deve essere reversibile: registri ombra, istruzioni che generano trap,
- evitare l'esecuzione di istruzioni costose: SPECULATIVE LOAD.

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)

La memoria principale troppo lenta

La differenza di velocità tra processore e memoria è aumenta col tempo.

L'accesso in memoria operazione costosa:
il processore deve attendere il dato
per più di una decina di cicli di clock.

Memoria Cache: memoria piccola e veloce:
contiene i dati utilizzati più frequentemente.

La memoria cache

Funzionamento:

- prima si cerca il dato in cache (cache hit),
- in caso di fallimento (cache miss): si carica il dato in cache dalla memoria principale.

Migliori prestazioni solo con numerosi cache hit.

Cache a più livelli

Aumenta il divario di velocità tra CPU RAM;
per evitare cache miss troppe costose, un secondo livello di cache:

- più ampio
- tecnologia meno costosa
- più lento
- contiene un sovrainsieme della memoria di primo livello

Spesso sono presenti 3 livelli di cache.

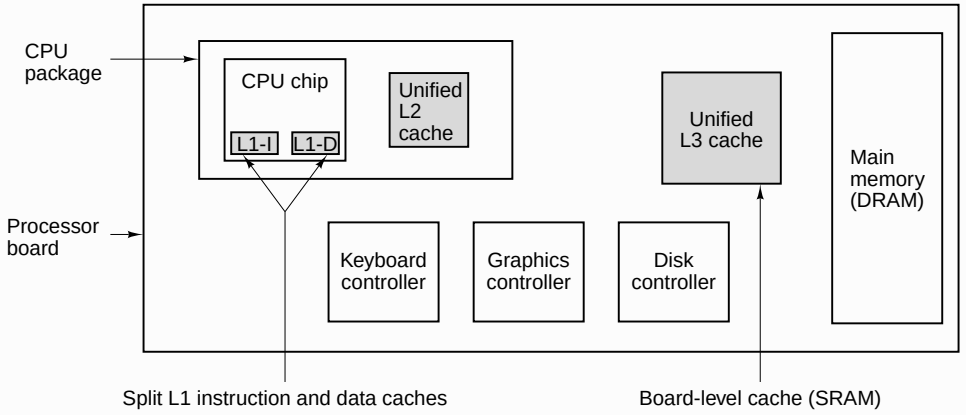
Esempio - Sandy Bridge: livello 1: 32KB; livello 2: 256KB, livello 3: 1–20MB;

Split cache

cache divisa in due parti: dati e istruzioni

- normalmente cache L1 (di primo livello)
- permette di parallelizzare l'accesso in memoria
  - l'IFU accede alle istruzioni
  - l'unità Dispatch/Execute accede ai dati

Esempio di configurazione



Valutazione delle prestazioni

Le velocità di un calcolatore con processore superscalare dipende fortemente da quanto viene sfruttato il potenziale parallelismo:

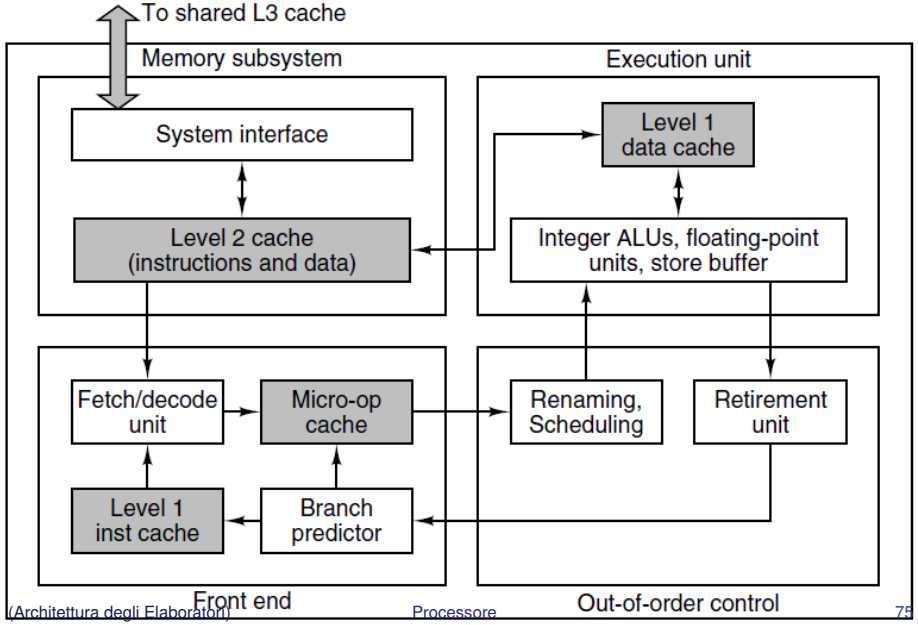
- percentuale delle istruzioni non bloccate per dipendenze,
- percentuale di predizioni di salto corrette,
- percentuale cache hit.

Queste percentuali difficilmente valutabili a tavolino, dipendono dal tipo di programmi eseguiti.
Una corretta valutazione delle prestazione può essere fatto solo tramite test.

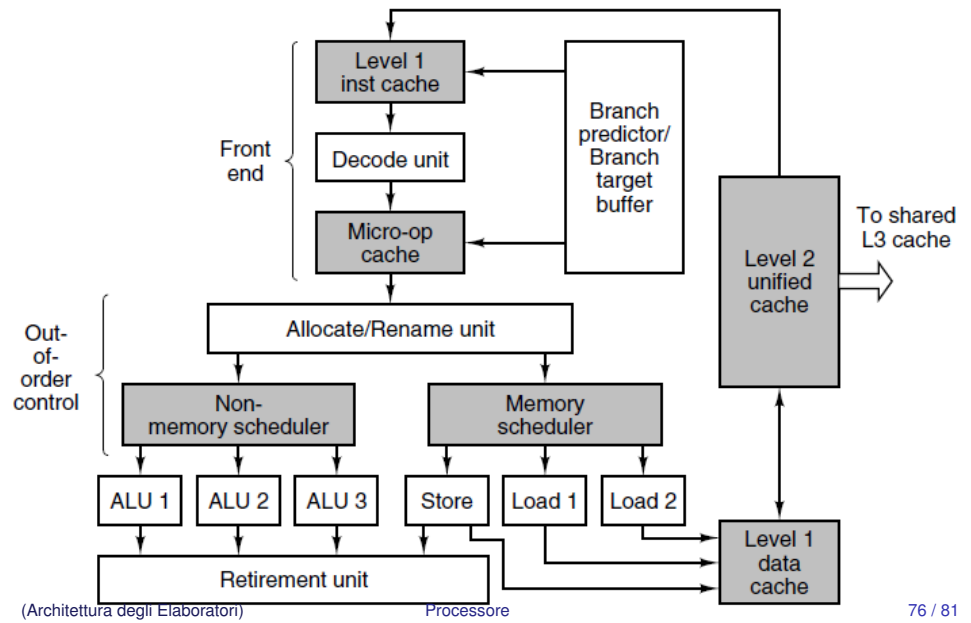
CPU Corei-7, Architett. Sandy-Bridge

- multicore, hyper-threading;
- processore CISC con cuore RISC;
- i primi stadi della pipeline traducono codice CISC in istruzioni RISC,
- depositate nella cache L0;
- più simile al Pentium II (P6) che al Pentium 4;
- risparmio energetico;
- predizione di salto sofisticata (algoritmo segreto)
- controllore memoria, cache L3 (condivisa) e processore grafico integrati nello stesso chip;
- nuove istruzioni grafiche AVX (Advanced Vector Extensions);

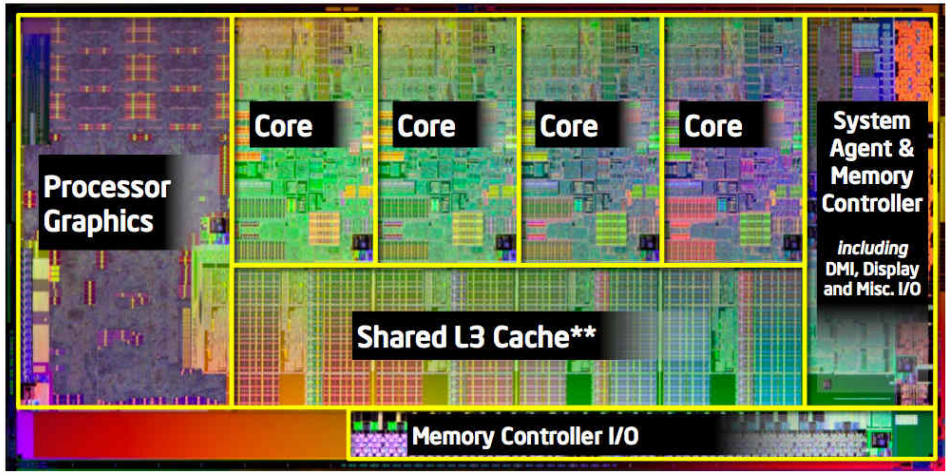
Architettura Sandy Bridge



Sandy Bridge, pipeline



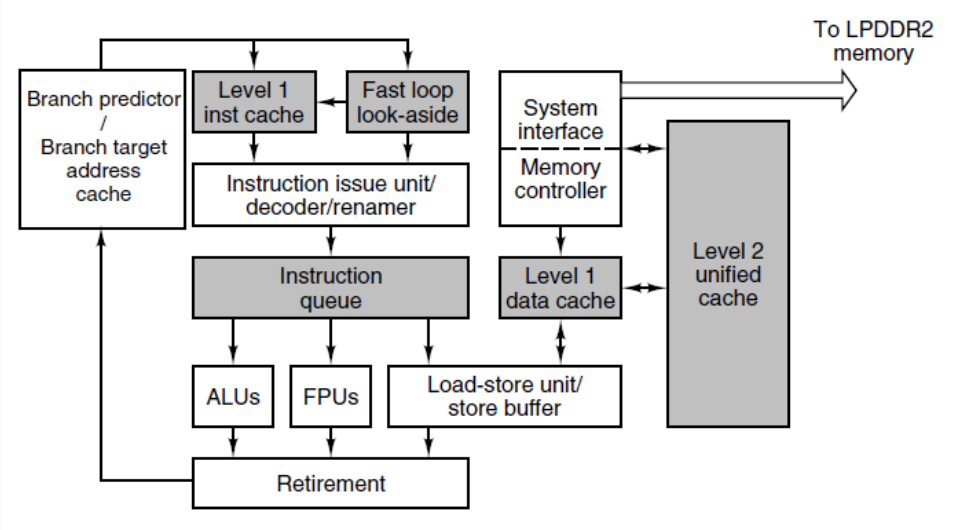
Sandy bridge, chip



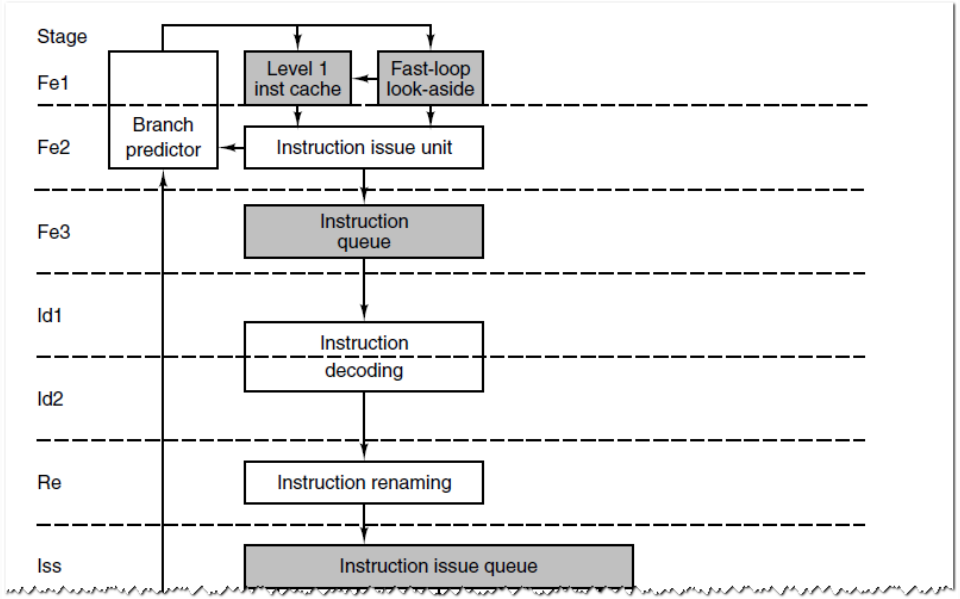
Architettura Cortex A9

- Progettata ARM ltd, realizzata da vari costruttori;
- core che viene integrato in SoC (System on Chip), (calcolatori su un singolo chip)
- implemeta istruzione ARMv7
- strutturalmente abbastanza simile al Sandy Bridge,
- manca lo stadio iniziale di traduzione istruzioni CISC -> RISC.

Architettura Cortex A9



Architettura Cortex A9



Architettura Cortex A9