


Gestione delle criticità nella pipeline

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

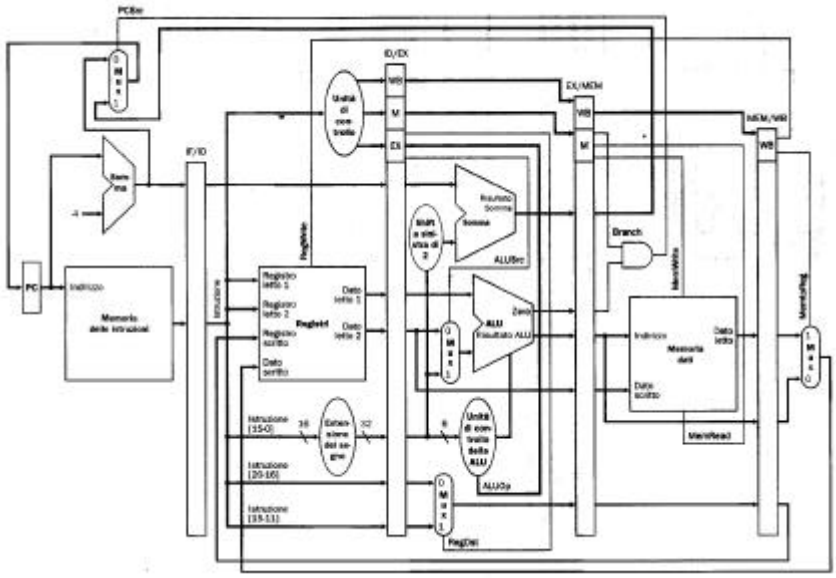

Sommario

Risoluzione degli Hazard sul controllo.

Trend di sviluppo delle pipeline.

Cenni sulla pipeline del Pentium 4.

CPU con pipeline



Esempio di Hazard sul controllo

800:	sub $s2, $s1, $s3	IF	ID	EX $s1-$s3	MEM	WB s->$2			
804:	beq $t2, $s6, tag		IF	ID	EX Zero if ($s2 == $s5)	MEM	WB		
808:	or $t7, $s6, $s7			IF	ID	EX	MEM	WB	
812:	add $t4, $s8, $s8				IF	ID	EX	MEM	WB
816:	and $s5, $s6, $s7					IF	ID	EX	MEM
Tag	add $t0, $t1, $t2						IF	ID	EX

In caso di salto: dovrei avere disponibile all'istante in cui inizia l'esecuzione dell'istruzione or l'indirizzo dell'istruzione add e non eseguire la or, la add e la and. NB L'indirizzo scritto nel PC corretto deve essere disponibile prima dell'inizio della fase di fetch. Ho 3 istruzioni sbagliate in pipeline.

L'indirizzo è già pronto al termine della fase di EX, posso quindi risparmiare un ciclo di clock. Ho 2 istruzioni da eliminare.

Soluzioni alla criticità nel controllo

Riordinamento del codice (delayed branch).

Modifiche strutturali per l'anticipazione dei salti.

Delayed branch

Decisione ritardata (ci si affida al compilatore).

Aggiunta di un "branch delay slot"

– l'istruzione successiva ad un salto condizionato viene sempre eseguita

– contiamo sul compilatore/assemblatore per mettere dopo l'istruzione di salto una istruzione che andrebbe comunque eseguita indipendentemente dal salto = posticipo un'istruzione precedente la branch.

Esempio di riorganizzazione del codice

```
        if (a = = b)
        {       s2 = s0 + s1;
        }

        s3 = s4 + s5;
salta:  s6 = 2;
```

```
if (a = = b)
{       s2 = s0 + s1;
        s3 = s4 + s5;
}
else
{       s3 = s4 + s5;
}

s6 = 2;
```


Esempio di delayed branch

```
        sub $t5, $t8, $s8
        add $s4, $t0, $t1
        beq $s5, $s6, salto
        add $s0, $s0, $s1
salto:
        add $t5, $t4, $t3
        add $t6, $t7, $t7
```

```
        sub $t5, $t8, $s8
        add $s4, $t0, $t1
        beq $s5, $s6, salto
        add $t5, $t4, $t3
        add $s0, $s0, $s1
salto:
        add $t6, $t7, $t7
```

```
        add $s4, $t0, $t1
        beq $s5, $s6, salto
        sub $t5, $t8, $s8
        add $s0, $s0, $s1
salto:
        add $t5, $t4, $t3
        add $t6, $t7, $t7
```

L'istruzione add $t5, $t4, $t3 o sub $t5, $t8, $s8 viene comunque eseguita, il salto (se richiesto) avviene all'istante successivo.

Esempio di riorganizzazione del codice - II

```
if (a = = b)
{       s2 = s0 + s1;

}
else
{
        t2 = t0 + t1;

}
s5 = s4 + s3;
t5 = 2;
```

```
if (a = = b)
{       s2 = s0 + s1;
        s5 = s4 + s3;
}
else
{
        t2 = t0 + t1;
        s5 = s4 + s3;
}
t5 = 2;
```


Comportamento della CPU: Hazard sul controllo

800:	sub $s2, $s1, $s3	IF	ID	EX $s1-$s3	MEM	WB s->$2			
804:	beq $t2, $s6, tag		IF	ID	EX Zero if ($s2 == $s5)	MEM	WB		
808:	or $t7, $s6, $s7			IF	ID	EX	MEM	WB	
812:	add $t4, $s8, $s8				IF	ID	EX	MEM	WB
tag:	and $s5, $s6, $s7					IF	ID	EX	MEM
tag +4	add $t0, $t1, $t2						IF	ID	EX

Scrivo il PC durante la fase di MEM, il nuovo indirizzo diventa disponibile solamente per l'istruzione successiva. Ho 2 istruzioni da eliminare.

L'indirizzo è già pronto al termine della fase di EX.

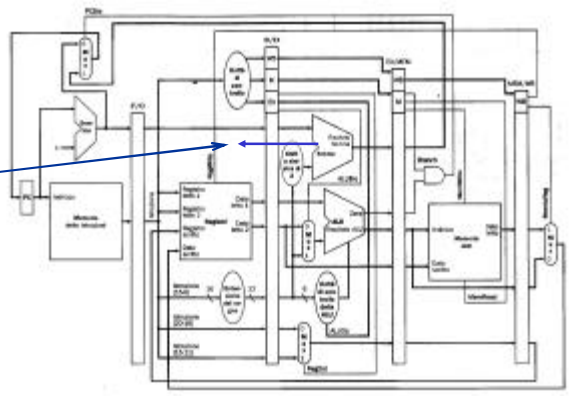
Modifica della CPU

Obbiettivi:

- Identificare l'hazard durante la fase ID di esecuzione della branch.
- Scartare una sola istruzione.

800:	sub $s2, $s1, $s3	IF	ID	EX $s1-$s3	MEM	WB s->$2			
804:	beq $t2, $s6, tag		IF	ID	EX Zero if ($s2 == $s5)	MEM	WB		
808:	or $t7, $s6, $s7			IF	ID	EX	MEM	WB	
tag:	add $t4, $s8, $s8				IF	ID	EX	MEM	WB
tag +4:	and $s5, $s6, $s7					IF	ID	EX	MEM
Tag +8	add $t0, $t1, $t2						IF	ID	EX

Come identificare l'Hazard nella fase ID



ALU calcolo indirizzo di salto
+
Comparatore per confrontare Rs e Rt

Anticipazione della valutazione della branch: Modifica della CPU nella gestione dei salti: anticipazione del calcolo dell'indirizzo di salto.

- HW addizionale: un comparatore all'uscita del Register File.
- Anticipazione del sommatore .

Soluzione dell'Hazard

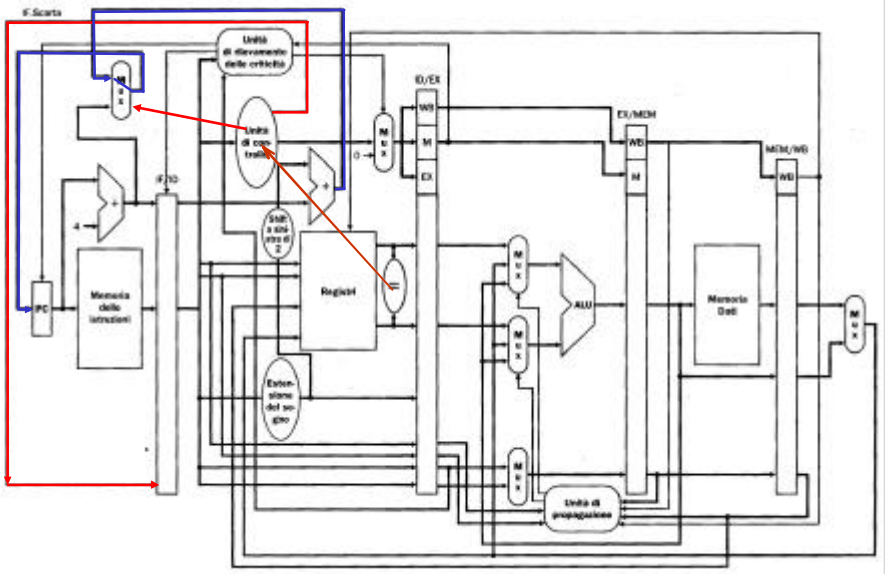
Stallo della pipeline.

Dalla fase ID alla WB la beq non fa nulla.

Nella fase di IF è l'istruzione successiva che viene trasferita nel registro IF/ID.

Occorre annullare l'istruzione nel registro IF/ID.

CPU con pipeline completa della gestione degli hazard.



Come scartare un'istruzione

IF.Scarta → carica nel registro IF/ID un'istruzione nulla.

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sll $s1, $s2, 7`	000000	X	10010	10001	00111 (7)	000000

$s1 = $s2 = $zero
Shmt = 0

Sommario

Risoluzione degli Hazard sul controllo.

Trend di sviluppo delle pipeline.

Cenni sulla pipeline del Pentium 4.

Branch prediction buffer

Branch prediction buffer (4 kbyte nel Pentium 4).

#bit < 32
Bit meno significativi del PC

Bit che indica se l'ultima volta il salto era stato eseguito o meno.

Problema:
Previsione relativa ad una beq con gli stessi bit meno significativi del PC.
E' un problema?

```
        beq $t0, $t1, SALTA
        add $s0, $s1, $s2
        sub $s3, $s4, $s5
        or  $s6 $s7, $s8
SALTA:  and $t2, $t3, $t4
```

In questo caso suppongo di non dovere saltare. Procedo in sequenza.
Se la previsione è sbagliata, devo annullare la add e saltare a SALTA.

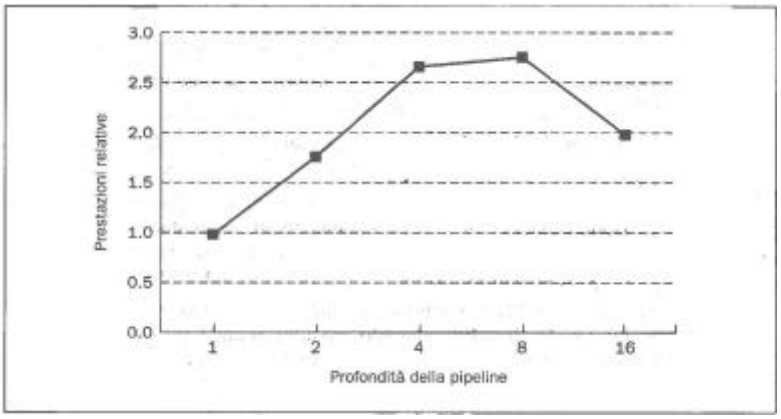
Algoritmi di ottimizzazione dello scheduling per predizione ottima del salto.

Pipeline più spinte

- Superpipelining.
- Superscalar.
- Scheduling dinamico.

Superpipeline

Pipeline più lunghe (e.g. Digital DecAlpha 21264, 5-7 stadi).
Teoricamente: guadagno in velocità proporzionale al numero di stadi.



Problemi:

- Criticità sui dati: stalli più frequenti.
- Criticità sul controllo: numero maggiore di stadi di cui annullare l'esecuzione.
- Maggior numero di registri: il clock non si riduce linearmente con il numero degli stadi.

Pipeline Superscalari: MIPS

Duplico unità funzionali (e.g. ALU e ALUfp) ed eseguo più istruzioni in parallelo.

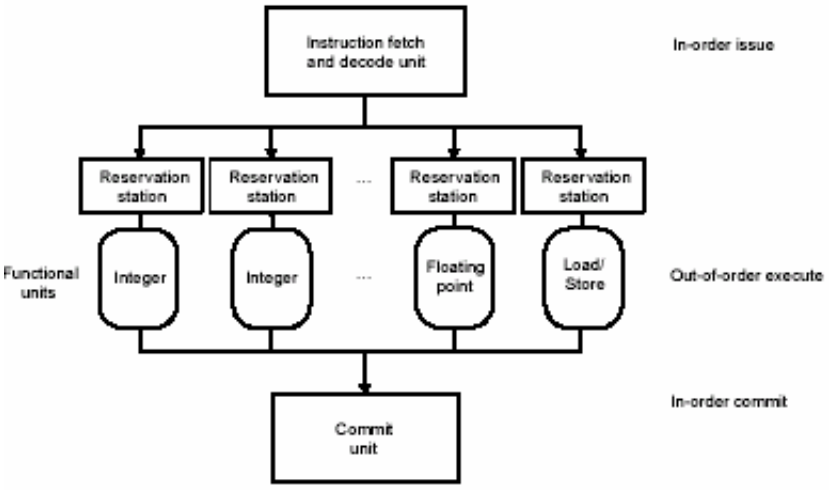
Ottengo più di 1 istruzione per ciclo di clock.

Per il MIPS: 4 porte di lettura, 2 porte di scrittura, 2 ALU (general purpose e base+offset)

Principi della pipeline con schedulazione dinamica

Si supera lo stallo: cerca di eseguire istruzioni successive *in attesa che lo stallo sia risolto*.

Pipeline con schedulazione dinamica



Esistono diversi cammini paralleli (per la fase di esecuzione e memoria) dell'istruzione.

Principi della schedulazione dinamica

Obbiettivo: mettere in esecuzione istruzioni che non presentino criticità.

Le istruzioni vengono bufferizzate dalla **reservation station**, la quale gestisce la coda delle istruzioni che hanno bisogno della stessa unità funzionale.

Al termine dell'esecuzione la **reorder station**, provvede ad ordinare le istruzioni nella sequenza con la quale devono essere restituite.

NB Le istruzioni non sono eseguite sequenzialmente.

Sommario

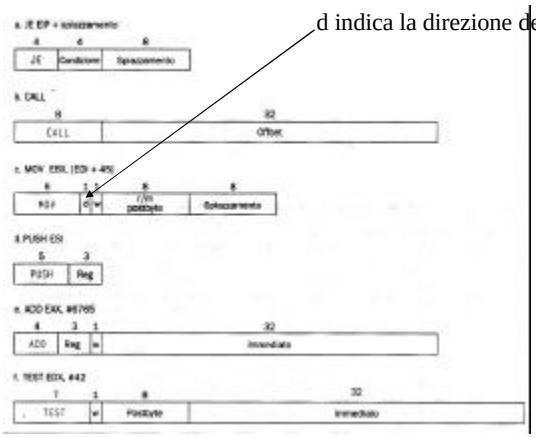
Risoluzione degli Hazard sul controllo.

Trend di sviluppo delle pipeline.

Cenni sulla pipeline del Pentium 4.

L'ISA dell'Intel

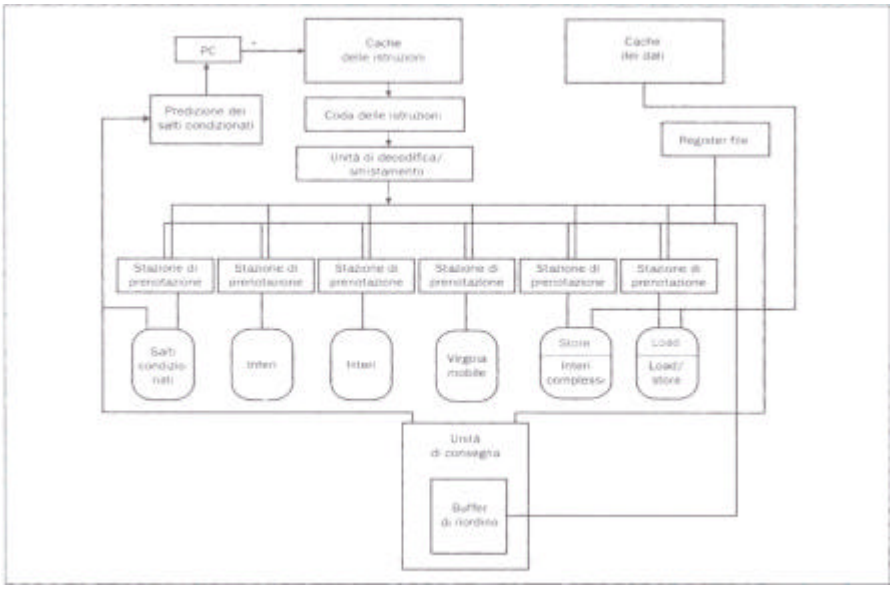
Molti formati: ampiezza 1-15byte. Codice operativo su 1 o 2 byte.
CLC (Clear Carry è un'istruzione su 1 byte, non ha operandi).
*mov EAX, ind1:[ind2 + ind3*4 + 2] (richiede 17 byte, mod = 2, r/m = 4)*



d indica la direzione del trasferimento

w specifica se lavora sul byte o sulla parola a 32 bit
Post_byte (r/m specifica la modalità di indirizzamento)

La pipeline del PentiumPro



Il funzionamento della pipeline del PentiumPro

Fetch: Viene prelevato un blocco di memoria istruzioni di 16 byte ed inviato alla coda istruzioni.

Decodifica:

1. Le istruzioni dell'ISA INTEL vengono convertite in microistruzioni di lunghezza fissa pari a 72 bit. **Le microistruzioni sono estremamente simili all'ISA del MIPS.**
2. In questa fase vengono valutate le branch (su un certo numero di istruzioni) attraverso una ***Branch Prediction Table*** con 512 elementi**.**
3. Recupero degli operandi ed assegnazione loro dei registri replicati (rename buffer).

Esecuzione:

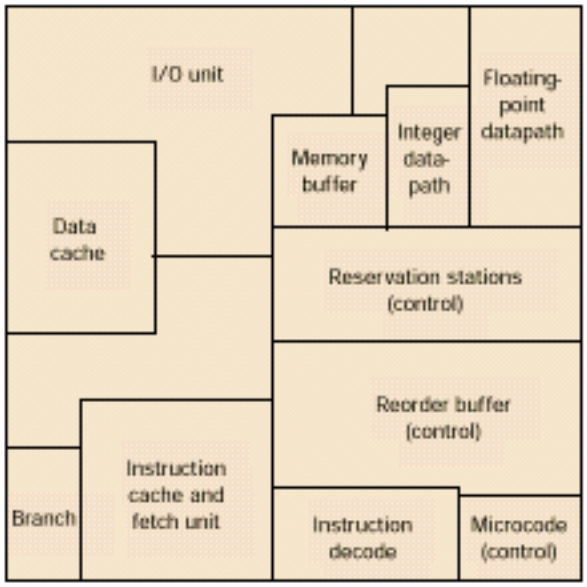
1. Smistamento. Le istruzioni vengono alla stazione di prenotazione corrispondente all'unità funzionale richiesta.
2. Un riferimento all'istruzione viene inserito nel buffer di riordino.
3. Esecuzione vera e propria (calcolo)
4. Riconsegna. **L'unità di consegna** tiene tracia di tutte le istruzioni pendenti all'interno del buffer di riordino. Essa **riordina le istruzioni** in modo che termino in modo sequenziale. Può fornire in uscita più itruzioni in un ciclo di clock.

Osservazioni sulla fase di esecuzione

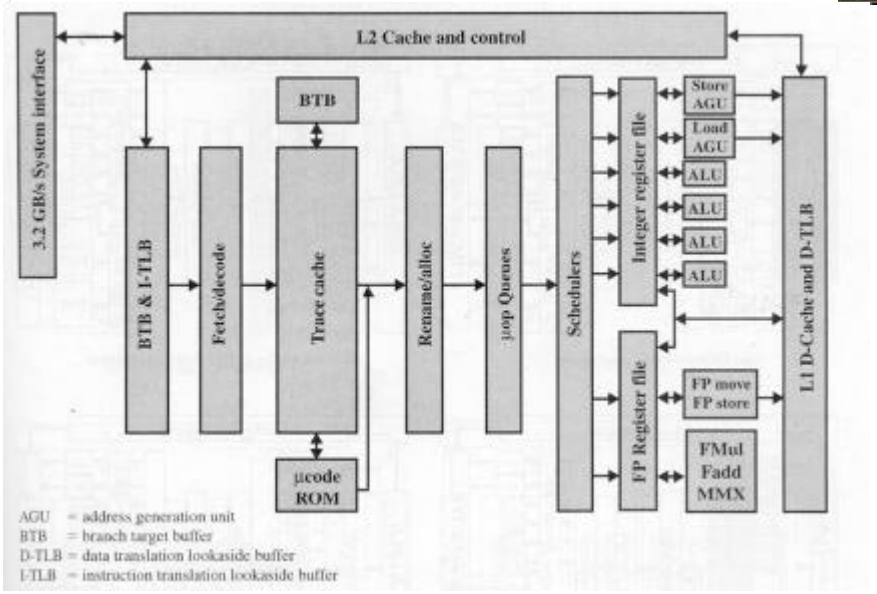
- **Per considerare un'istruzione occorre che ci sia spazio nel Buffer di Riordino e nella stazione di prenotazione relativa alla pipeline di esecuzione richiesta.**
- La CPU contiene duplicazione dei registri in cui scrivere il risultato. Con cosi' tante istruzioni in esecuzione, è possibile avere il registro di destinazione occupato e occorre scrivere quindi su un registro ausiliario (**rename registers or rename buffers**). In questi registri vengono memorizzati i risultati in attesa che l'unità di consegna **dia il permesso di scrivere i registri effettivi della CPU**.
- L'istruzione viene eseguita quando il corrispondente elemento all'interno della stazione di prenotazione possiede tutti gli operandi, e la relativa unità funzionale è libera.
- Un'istruzione non viene terminata (riconsegnata) fino a quando non lo decide l'unità di consegna. Se la predizione di un salto è scorretta, vengono scartate le istruzioni sbagliate dalle stazioni di prenotazione e dai buffer di riordino e liberati i registri interni.

Il processore Pentium Pro

Esecuzione "speculativa": scheduling dinamico + predizione dei salti (e.g. Intel 80x86 dal Pentium).



LA CPU del Pentium 4



Funzionamento della Pipeline del Pentium 4

Fetch delle istruzioni della Memoria (Cache)

Ciascuna istruzione viene tradotta in una o più microistruzioni di lunghezza fissa.

Il processore esegue le micro-istruzioni in una pipeline superscalare a schedulazione dinamica.

Il processore restituisce il risultato (finale) dell'esecuzione delle micro-istruzioni relative alla stessa istruzione ai registri nell'ordine nel quale le istruzioni sono state inviate in esecuzione.

Il Pentium 4 trasforma un'ISA CISC in un'ISA interno RISC costituito dalle micro-operazioni.

Stadi della pipeline del Pentium 4: Front end

Le istruzioni passano dalla cache primaria al buffer L2 (64 byte). A questo stadio vengono gestite le miss ed i trasferimenti da cache.

In parallelo i dati vengono caricati da cache nel buffer dati L1.

Nel trasferimento da L1 ed L2 agiscono Branch Target Buffer e lookaside buffer che gestiscono le predizioni sulle branch.

A questo punto inizia la fase Fetch e Decodifica che legge il umero di byte pari alla lunghezza dell'istruzione.

Il decoder traduce l'istruzione in da 1 a 4 micro-operazioni: 118bit, appartenenti ad un'ISA RISC.

Stadi della pipeline del Pentium 4: Trace cache

Riordina localmente il codice in modo da evitare le criticità: la pipeline ha 20 stadi.

Invia alla fase di esecuzione (Allocate e Renaming).

Per istruzioni complesse (> 4 microistruzioni) vengono create in questa fase con una macchina a stati finiti implementata con una ROM + memoria.

Stadi della pipeline del Pentium 4: Esecuzione

Elemento centrale è il **Reorder Buffer** – **ROB** (Buffer circolare che può contenere fino a 126 micro-istruzioni).

Contiene le micro-istruzioni con il loro stato, la presenza o assenza dei dati, per tutta la durata dell'esecuzione.

Alloca i registri (traducendo i registri simbolici dell'Assembly in uno dei 128 registri di pipeline).

Avvia alla coda di esecuzione l'istruzione (coda per le lw/sw e coda per le altre istruzioni).

Lo scheduling avviene prendendo la prima istruzione che può essere eseguita nella coda. Dagli scheduler si può passare alle ALU in modi diversi.

La fase di esecuzione scrive poi i risultati nei registri o nella cache L1.

C'è una parte dedicata ai flag che vengono poi inviati alla BTB per predirre correttamente i salti.

Storia della pipeline

Primo calcolatore con pipeline: Stretch: IBM 7030, 1957.

CDC6600 della Digital (1964). Istruzioni semplici e sfruttamento massimo della pipeline.

IBM 360/91. Introduzione delle *reservation station* e rilevamento dinamico delle criticità (branch). Anticipazione del PowerPC e del PentiumPro.

John Cocke nel 1980 inventò il termine superscalare, ed una CPU che prendesse decisioni dinamicamente.

RISC (1980). Delayed branch. ISA con istruzioni di dimensione fissa.

Sommario

Risoluzione degli Hazard sul controllo.

Trend di sviluppo delle pipeline.

Cenni sulla pipeline del Pentium 4.