

AE 2

#uni

IF/Fetch -> scritto in program counter nuovo indirizzo della nuova istruzione (32-bit)

ID/Decode -> lettura degli indirizzi necessari (lettura in register file)

EX/Execute -> esecuzione di operazione aritmetico logica da parte dell'ALU

MEM/Memory -> preleviamo o scriviamo un dato delle memorie (Operazioni load_word/store_word)

WB/Writeback -> scrittura in un registro del register_File (quando necessario)

Cammini seguiti sul **data path** (e fasi necessarie):

Istruzione Aritmetico/Logica :	IF → ID → EX → WB	add, or, not, +, - ...
Accesso a memoria in lettura (lw) :	IF → ID → EX → MEM → WB	lw
Accesso a memoria in scrittura (sw) :	IF → ID → EX → MEM	sw
Salto condizionato (beq)	IF → ID → EX	beq, bne, bxx
Salto non condizionato (j)	IF → ID	j

Formato R

es. add rd, rs, rt

<i>OPCODE</i>	<i>r s</i>	<i>r t</i>	<i>r d</i>	<i>shamt</i>	<i>funct</i>
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

Formato I

es. beq rs, rt, OFFSET

lw rt, OFFSET(rs)

sw rt, OFFSET(rs)

<i>OPCODE</i>	<i>r s</i>	<i>r t</i>	<i>IMMEDIATO</i>
6 bit	5 bit	5 bit	16 bit

Formato J

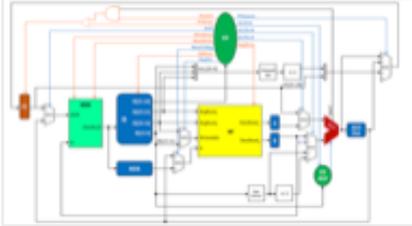
es. j ADDRESS

<i>OPCODE</i>	<i>PSEUDO-INDIRIZZO</i>
---------------	-------------------------

6 bit

26 bit

CPU CICLO MULTIPLIO



SEGNALI DI SELEZIONE

ALUSrcA	Selezione del primo operando ALU: PC (0), registro A (1)
ALUSrcB	Selezione del secondo operando ALU: Registro B (00), costante 4 (01), SignExt(OFFSET) (10), SignExt(OFFSET) * 4 (11)
lorD	Selezione per il campo Addr nel modulo memoria: indirizzo istruzione (0), indirizzo dato (1)
PCSrc	Selezione di quale indirizzo mandare al PC: risultato della ALU (00), contenuto di ALUOut (01), indirizzo della jump (10), (11 non usato)
RegDest	Selezione per il campo WriteAddr del RF: r1 (0), rd (1)
MemToReg	Selezione del dato presentato in scrittura al RF: contenuto di ALUOut (0), contenuto di MEM (1)
ALUOut	Selezione della modalità di operazione della ALU (analogo a CPU singolo ciclo)

SEGNALI DI CONTROLLO

PCWrite	Scrittura del Program Counter
Branch	Se posto a 1 abilita la scrittura del PC quando il bit di zero della ALU vale 1, da usare per i salti condizionati
IRWrite	Scrittura dell'Instruction Register
RegWrite	Scrittura del Register File
MemWrite	Scrittura della memoria
MemRead	Letture della memoria

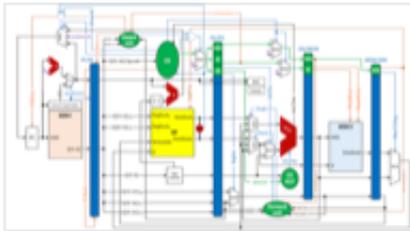
CONTROL UNIT

IF	
lorD = 0 MemRead = 1 IRWrite = 1 ALUSrcA = 0 ALUSrcB = 01 ALUOp = 00 PCSrc = 1 PCWrite = 1	
ID-1	ID-2

ALUSrcA = 0 ALUSrcB = 11 ALUOp = 00	PCWrite = 1 PCSrc = 10
---	---------------------------

EX (beq) ALUSrcA = 1 ALUSrcB = 00 ALUOp = 01 PCSrc = 01 Branch = 1	EX (A/L) ALUSrcA = 1 ALUSrcB = 00 ALUOp = 10	EX (lw/sw) ALUSrcA = 1 ALUSrcB = 10 ALUCtrl = 00
MEM (lw) lorD = 1 MemWrite = 0 MemRead = 1	MEM (sw) lorD = 1 MemWrite = 1 MemRead = 0	
WB (A/L) RegDst = 1 MemToReg = 0 RegWrite = 1	WB (lw) RegDst = 0 MemToReg = 1 MemWrite = 1	

CPU A PIPELINE



Consideriamo un programma fatto da N istruzioni, 5 fasi di esecuzione ciascuna di durata t_j con $j \in \{1,2,3,4,5\}$ e la seguente funzione:

$$f(i,j) = \begin{cases} 1, & \text{Istruzione } i \text{ necessita della fase } j \\ 0, & \text{altrimenti} \end{cases}$$

La durata del ciclo di clock T_{ck} è costante, quindi dobbiamo assumere che tutte le fasi durino in realtà $t = \max\{t_j\}$

Calcoliamo T definito come il tempo totale impiegato per eseguire il programma

- Nella CPU a singolo ciclo $T_s = NSt$ e abbiamo il vincolo che $T_{ck} \geq 5t$
- Nella CPU a ciclo multiplo $T_M = \sum_{i=1}^N \sum_{j=1}^5 f(i,j)t$ e abbiamo il vincolo che $T_{ck} \geq t$
- Nella CPU a pipeline $T_P = \underline{5t} + t(N-1)$ e abbiamo ancora il vincolo che $T_{ck} \geq t$

prima istruzione pipeline non a regime (busti)

Confronto dei tempi totali: $T_P \leq T_M \leq T_S$

Throughput (tasso di produzione), definizione: $\rho = \lim_{Lavoro \rightarrow \infty} \frac{Lavoro}{Tempo\ speso}$

Nella CPU singolo ciclo: $\rho_S = \lim_{N \rightarrow \infty} \frac{N}{N5t} = \frac{1}{5t}$

Nella CPU a pipeline: $\rho_P = \lim_{N \rightarrow \infty} \frac{N}{5t + t(N-1)} = \frac{1}{t}$ \Rightarrow $\rho_P = 5\rho_S$, la CPU a pipeline è 5 volte più veloce di una CPU a singolo ciclo

Il risultato si generalizza rispetto al numero di stadi:
a parità di durata delle fasi, una CPU a pipeline con k stadi è k volte più veloce di una CPU a singolo ciclo

SEGNALI

SEGNALI DI CONTROLLO			
SEGNALI DI SELEZIONE		SEGNALI DI COMANDO	
ALUSrc	Selezione del secondo operando ALU: (rt) (0) immediato esteso di segno (1)	RegWrite	Scrittura del Register File
RegDest	Selezione campo WriteAddr del RF: rt (0), rd (1)	MemWrite	Scrittura della memoria
MemToReg	Selezione del dato presentato in scrittura al RF: risultato ALU (0), contenuto di dato letto da MEM (1)	MemRead	Letture della memoria
AluCtrl	Selezione della modalità di operazione della ALU (uguale a CPU singolo ciclo)	*Il registro di memoria non ha segnali di controllo perché legge ad ogni ciclo di clock	
Branch	Indica se istruzione corrente è una beq		
PCSrc	Selezione di quale indirizzo mandare al PC: PC+4 (0), BTA (1)		

Nota 1: PCSrc sarà calcolato direttamente nel datapath come **Branch AND Zero**

	EX			MEM			WB	
	ALUSrc	RegDst	AluCtrl	Branch	MemWrite	MemRead	MemToReg	RegWrite
A/L	0	1	10	0	0	0	0	1
lw	1	0	00	0	0	1	1	1
sw	1	x	00	0	1	0	x	0
beq	0	x	01	1	0	0	x	0

ECCEZIONI

- Le **branch** e le **jump** causano deviazioni **strutturate** del flusso di controllo: sono appositamente progettate dal programmatore affinché il programma svolga il compito desiderato, sono **previste e fondamentali** alle funzioni svolte
- Esistono deviazioni del flusso causate da eventi imprevisi: **eccezioni**

Eccezione: una deviazione imprevista del flusso di controllo la cui causa può genericamente verificarsi internamente o esternamente alla CPU

Interrupt: un'eccezione la cui causa si verifica esternamente alla CPU

Evento	Origine	Terminologia MIPS
Istruzione non riconosciuta (es. opcode non valido)	Interna	Eccezione
Overflow	Interna	Eccezione
Sycall (il programma invoca un servizio del sistema operativo)	Interna	Eccezione
Richiesta da una periferica I/O	Esterna	Interrupt
Malfunzionamento hardware	Interna/Esterna	Eccezione/Interrupt

NOTAZIONE REGISTRI

Identificatore	Numero	Utilizzo in MIPS
\$zero	0	contiene la costante 0
\$at	1	riservato all'assembler
\$v0, \$v1	2,3	valori di ritorno di una procedura
\$a0, \$a1, \$a2, \$a3	4,5,6,7	parametri da passare a una procedura
\$t0, \$t1, ... \$t7	8,9, ... 15	registri temporanei (non preservati su chiamate di procedura)
\$s0, \$s1, ... \$s7	16,17, ... 23	registri salvati (preservati su chiamate di procedura)
\$t8, \$t9	24,25	registri temporanei (non preservati su chiamate di procedura)
\$k0, \$k1	26,27	gruppone delle eccezioni
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address

CACHE

d	un dato che ha un indirizzo in memoria principale, per noi è il byte che può anche rappresentare una parola di 32 bit
$M(d)$	l'indirizzo del dato d in memoria principale, per noi è sempre su 32 bit
L	numero di linee della cache o, in alternativa, il numero di blocchi che la cache può contenere
B	numero di dati contenuti sequenzialmente in un blocco (la dimensione di un blocco), in byte
$N(d)$	numero del blocco in memoria in cui si trova d : se guardiamo la memoria come ad una sequenza di blocchi di dimensione B numerati progressivamente con 0,1,2,... quale è il numero del blocco che contiene d ?
$div(a,b)$	quoziente della divisione intera tra a e b : $\lfloor \frac{a}{b} \rfloor$
$mod(a,b)$	resto della divisione intera tra a e b : $a - \lfloor \frac{a}{b} \rfloor \cdot b$
$I(d)$	indice della linea di cache a cui si trova il blocco che contiene il dato d



MAPPATURA DIRETTA

Numero di blocco $\rightarrow N(d) = div(M(d), B)$

Linea $\rightarrow L = Ddata / B$ (set | linee da $n = 1$ posto)

Indice di cache $\rightarrow Id = mod(N(d), L)$

Indice di linea $\rightarrow k = log_2(L)$

Offset nel blocco $\rightarrow m = log_2(B) - 2$

Tag (bit) $\rightarrow 32 - (m + 2) - k$

Ddata $\rightarrow L \cdot n \cdot B$

Dimensione totale

$$D_{tot} = L \times \frac{8B + (32 - \log_2 B - \log_2 L) + 1}{8}$$

CACHE FULLY-ASSOCIATIVE

Numero di blocco $\rightarrow N(d) = div(M(d), B)$

Linea $\rightarrow L=1$

Indice di cache $\rightarrow Id = mod(N(d), L)$

Indice di linea $\rightarrow k = 0$ (non esiste, è come se fosse un unica linea)

Offset nel blocco $\rightarrow m = log_2(B) - 2$

Tag (bit) $\rightarrow 32 - (m + 2) - k$

Ddata $\rightarrow L \cdot n \cdot B$

Dimensione totale

Codice di Hamming

- Codice su n bit dove alcuni bit sono di parità mentre altri sono utilizzati per il dato
- Quali bit hanno il ruolo di parità?
- Numeriamo i bit da 1 a n , da sinistra a destra (al contrario rispetto a quello che facciamo con la codifica binaria), per non confonderci con la posizione di un bit chiamiamo questo numero **indice**
- 1 bit di parità sono quelli il cui indice è una potenza di 2

Esempio con $n = 20$



Se ho 20 bit in totale, vi saranno 5 bit di parità e ne restano 15 per il dato

Codice di Hamming



- Come si sceglie il bit di parità? Bisogna far vedere la parità su una specifica gruppo di bit, e se non va bene, si cambia il bit di parità.
- Il bit di parità p_1 è il bit di parità per il gruppo di bit {1, 2, 3, 4}.
- Il bit di parità p_2 è il bit di parità per il gruppo di bit {1, 2, 5, 6, 3, 4, 7, 8}.
- Il bit di parità p_4 è il bit di parità per il gruppo di bit {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}.
- Il bit di parità p_8 è il bit di parità per il gruppo di bit {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}.
- Il bit di parità p_{16} è il bit di parità per il gruppo di bit {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}.

Indice	Bit
1	0
2	1
3	0
4	1
5	1
6	0
7	1
8	0
9	1
10	0
11	1
12	0
13	1
14	0
15	1
16	0
17	1
18	0
19	1
20	0

Codice di Hamming

Se ho un dato in cui un gruppo di bit di parità risulta essere allora c'è un solo bit dati che può essere errato. Funziona rispetto ad ogni gruppo di bit di parità. Se non funziona...

Codice di Hamming, leggi del pattern

- Il pattern determina p_i il numero di bit di parità p_i il numero di bit per il dato, da cui dedurremo $n_{dat} = n - p_i$.
- Problema 1: Dato n , quanto scegliere p_i e il numero n_{dat} ?
- Il bit di parità p_i è il bit di parità per il gruppo di bit {1, 2, 3, 4, ..., 2^{i-1} }.
- Il bit di parità p_i è il bit di parità per il gruppo di bit {1, 2, 3, 4, ..., 2^{i-1} }.
- Il bit di parità p_i è il bit di parità per il gruppo di bit {1, 2, 3, 4, ..., 2^{i-1} }.
- Il bit di parità p_i è il bit di parità per il gruppo di bit {1, 2, 3, 4, ..., 2^{i-1} }.

Codice di Hamming

Esempio

$d = 11100101$

servono in totale 12 bit, 4 bit di parità e 8 bit per il dato

p_1 parità su $d_1 + d_2 + d_4 + d_8 = 2$, quindi $p_1 = 0$

p_2 parità su $d_1 + d_3 + d_4 + d_6 + d_7 = 3$, quindi $p_2 = 1$

p_4 parità su $d_2 + d_3 + d_4 + d_8 = 3$, quindi $p_4 = 1$

p_8 parità su $d_5 + d_6 + d_7 + d_8 = 2$, quindi $p_8 = 0$

codifica: 01111100101

Bit aggiuntivo di parità

- Aggiungendo un ulteriore bit di parità sul pattern otteniamo $\delta = 4$ quindi possiamo identificare e correggere errori singoli e anche identificare (ma non correggere) errori doppi
- Nuovo bit di parità p_{n+1} si applica a tutti gli altri bit (che ora diventano $n_{dat} - 1$)
- Si possono verificare 4 casi in fase di verifica
 - $ECC = 0$ e p_{n+1} corretto \rightarrow non ci sono errori
 - $ECC > 0$ e p_{n+1} errato \rightarrow singolo errore che si può correggere
 - $ECC = 0$ e p_{n+1} errato \rightarrow singolo errore proprio in p_{n+1} , si può correggere
 - $ECC > 0$ e p_{n+1} corretto \rightarrow doppio errore, sappiamo che c'è stato ma non si può correggere

AE2_teorìa_summary.pdf
PDF Document · 4.5 MB

AE2-Teoria.pdf
PDF Document · 12.7 MB