APPELLO MARZO 2013 ARCHITETTURA DEGLI ELABORATORI

ATTENZIONE: le risposte sono quasi totalmente del buon Caesar, prese da altri testi scritti da lui. A lui vanno i meriti per le risposte e i ringraziamenti. Io, Nicolò Scapin, ho scritto la risposta 4 ed effettuato alcune correzioni minori. Se ci sono errori, modificate pure!

ESERCIZIO 1:

Quante volte la CPU deve accedere alla memoria quando preleva ed esegue un'istruzione cha ha due operandi, uno con modo di indirizzamento diretto e uno con modo di indirizzamento indiretto?

a) 2 b) 3

c) 1 d) 4

e) nessuna delle risposte precedenti

Risposta: D

Un accesso per il fetch, uno per l' indirizzamento diretto e due per indirizzamento indiretto. In totale 4

ESERCIZIO 2:

Si consideri la seguente rappresentazione in virgola mobile a singola precisione (IEEE 754) :

Il numero rappresentato è:

a) 10,125 b) -10,125

c) 1,7675781 d) -1,7675781

e) nessuna delle risposte precedenti

Soluzione

Rappresentazione in virgola mobile a singola precisione IEEE 754 con:

- 1 bit per il segno
- 8 bit per esponente, quindi:
  - esponente = esponente_polarizzato - (2^{8-1} -1) = esponente_polarizzato -127
  - esponente_polarizzato = esponente + 127
- 23 bit per la mantissa (+ 1 bit (implicito) del primo 1)

1 100001 01110001001000000000000

- segno = 1 → -
- esponente_polarizzato = 10000101 in base due = $2^7+2^2+2^0$= 133
- esponente = esponente_polarizzato -127=133 - 127 = 6

numero = - 1. 11000100100000000000000 x 2^6= $2^6+2^5+2^4+2^0+2^{-3}$= 113,125

quindi risposta E

ESERCIZIO 3:

Si consideri una pipeline a 4 stadi: fetch (IF), decodifica (ID), elaborazione (EI), e scrittura

dei risultati (WO), per cui:

- i salti incondizionati sono risolti (identificazione salto e calcolo indirizzo target) alla fine del secondo stadio (ID);
- i salti condizionati sono risolti (identificazione salto, calcolo indirizzo target e calcolo condizione) alla fine del terzo stadio (EI);
- il primo stadio (IF) indipendente dagli altri;

inoltre si assuma che non ci siano altre istruzioni che possano mandare in stallo la pipeline e che non sia implementato alcun meccanismo di trattamento dei salti.

Sapendo che:

- il 20% delle istruzioni sono di salto condizionale
- il 3% delle istruzioni sono di salto incondizionale
- il 45% delle istruzioni di salto condizionale hanno la condizione soddisfatta (prese)

Il fattore di velocizzazione della pipeline è:

a) 3.305785 b) 3.508772

c) 2.356502 d) 3.960031

e) nessuna delle risposte precedenti

Soluzione

Percentuali:

- 3% = 0.03 salto incondizionale
- 20% = 0.20 salto condizionale, di cui:
  - 45% = 0.45 preso ==> quindi il 0.20 * 0.45 = 0.09 delle istruzioni totali
  - 55% = 0.55 non preso ==> quindi il 0.20 * 0.55 = 0.11 delle istruzioni totali

Percentuali con cicli di stallo:

- salto incodizionato: 0.03*1 =0.03
- salto condizionato preso: 0.09*2 = 0.18
- salto condizionato non preso: 0.11*0=0

quindi la frazione di cicli di stallo = 0.03+0.18+0 = 0.21

Essendovi 4 stadi → k = 4

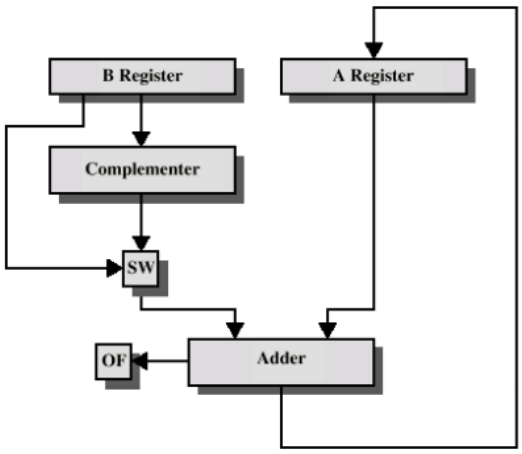
$$fattore\ di\ velocizzazione = S_k = \frac{1}{1 + frazione\ cicli\ stallo} k = \frac{1}{1 * 0{,}21} 4 = 3{,}305785$$

Quindi risposta A

ESERCIZIO 4:

si spieghi in dettaglio l'hardware adottato per realizzare la somma e sottrazione di numeri interi rappresentati in complemento a due.

L'elemento centrale dell'hardware è il sommatore binario(ADDER in figura), al quale vengono forniti gli addendi o sottraendo e minuendo. Il sommatore può produrre il risultato della somma o sottrazione oppure un overflow, che genera un flag, inserito in un apposito registro. Per la somma, i due numeri provengono da due registri, A e B ed il risultato può essere memorizzato in uno di questi due registri oppure in un terzo. Per la sottrazione, il sottraendo, proveniente dal registro B, viene modificato, facendo il complemento, e sommatore legge il suo complemento a due.



PS: per la sottrazione ricordo che per sottrarre un numero(sottraendo) da un altro (minuendo), si considera l'opposto del sottraendo e lo si somma al minuendo.

ESERCIZIO 5:

Si descriva sinteticamente l'implementazione delle istruzioni attraverso la tecnica della microprogrammazione. Si dica se questa tecnica viene utilizzata per i processori CISC o RISC, motivare la risposta.

Soluzione:

La microprogrammazione è utilizzata per implementare l'Unità di Controllo della CPU la quale, grazie al microprogramma, implementa ogni istruzione tramite una sequenza di micro-operazioni eseguite direttamente dall'hardware e di generare nella giusta sequenza i segnali di controllo che provocano l'esecuzione di ogni operazione elementare. Il microprogramma (firmware) nell'unità di controllo ha una struttura ciclica in cui alterna l'esecuzione di una operazione speciale con l'esecuzione di un'operazione esterna il cui codice e dati da elaborare sono stati acquisiti dalla operazione speciale. Il microprogramma riunisce quindi tutte le micro- operazioni necessarie per effettuare l'operazione speciale del microprogramma e le microoperazioni necessarie per effettuare ogni operazione esterna.In generale la parte operativa invia all'unità di controllo delle variabili di condizionamento. In base a queste variabili, l'Unità di controllo manda dei segnali di controllo (α β) che designano la microoperazione da eseguire.La microprogrammazione è la soluzione tipica di architetture CISC per implementare l'Unità di Controllo. Questo perché l'adozione di tale tecnica permette una maggiore flessibilità nella progettazione, cioè rende facile modificare le sequenze di micro-operazioni che implementano le istruzioni eseguite dalla CPU, e premette la realizzazione di un vasto numero di istruzioni da parte della CPU.

ESERCIZIO 6:

spiegare, all'interno del contesto mips, come e quando la cpu attiva i circuiti di bypass

soluzione:

mancante

ESERCIZIO 7:

Discutere le motivazioni alla base dei processori multicore.

Soluzione:

I microprocessori hanno visto una crescita esponenziale delle prestazioni grazie al miglioramento dell'organizzazione ed all'incremento delle frequenza di clock. Il miglioramento dell'organizzazione del cip è stato fortemente focalizzato sull'incremento delle parallelismo tra istruzioni. Si è passati infatti dall'introduzione della pipeline, a CPU superscalari, in cui vi sono pipeline parallele, sino a CPU con multithreading simultaneo (SMT), in cui alle pipeline parallele sono associati banchi di registri replicati.Tali miglioramenti hanno però richiesto un aumento di complessità, dalla quale segue quindi una logica più complessa quindi difficile da realizzare, progettare e verificare, ed un aumento dell'area del chip per permettere di supportare il parallelismo. Inoltre all'aumentare della densità del cip, dovuta all'aumento delle complessità, ed all'aumentare della frequenza di clock è seguito un aumento esponenziale della potenza richiesta, quindi maggiore energia consumata e maggior calore prodotto. Con le CPU SMT si era giunti al limite della potenza erogabile ed ai limiti del parallelismo a livello di istruzioni, quindi per permettere un aumento delle capacità delle CPU si è scelto di passare adarchitetture multicore.Si ipotizza infatti che con le architetture multicore sia possibile un incremento prestazionale quasi lineare, anche se i vantaggi prestazionali dipendono dallo sfruttamento efficace delle risorse parallele da parte dei programmi (piccole quantità di codice seriale ha un impatto significativo sulle prestazioni).

ESERCIZIO 8:

Sia data la seguente sequenza di istruzioni assembler, dove i dati immediati sono espressi in

esadecimale.

SW $8, 150($0)

ADD $2, $0, $8

LW $4, 10($2)

ADDI $4, $4, 3

ADDI $2, $4, 4

SW $4, 208($2)

ADD $5, $4, $2

Si consideri la pipeline MIPS a 5 stadi vista a lezione, con possibilità di data-forwarding e con possibilità di scrittura e successiva lettura dei registri in uno stesso ciclo di clock.mostrate come evolve la pipeline durante l'esecuzione del codice, spiegando nel dettaglio i motivi di un eventuale stallo o dell'utilizzo di un particolare circuito di by-pass.

Soluzione:

Riscrivo la consegna chiarificando le istruzioni macchina ed evidenziando le dipendenze:

SW	$8, 150($0)	mem[150 + [R0]] ← R8
ADD	$2, $0, $8	R2 ← [R0] + [R8]
LW	$4, 10($2)	R4 ← mem[10 + [R2]]
ADDI	$4, $4, 3	R4 ← [R4] + 3
ADDI	$2, $4, 4	R2 ← [R4] + 4
SW	$4, 208($2)	mem[208 + [R2]] ← R4
ADD	$5, $4, $2	R5 ← [R4] + [R2]

--- parentesi Teorico/Pratica ---
I forward possibili nella MIPS sono solo:

- da EX → a EX
- da MEM → a EX

Vi è inoltre la possibilità di scrittura e successiva lettura dei registri in uno stesso ciclo di clock e ciò permette il passaggio di dati da WB a ID entro lo stesso ciclo.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
SW	$8, 150($0)	IF	ID	EX	MEM	WB									
ADD	$2, $0, $8		IF	ID	EX	MEM	WB								
LW	$4, 10($2)			IF	ID	EX	MEM	WB							
ADDI	$4, $4, 3				IF	ID	ID	EX	MEM	WB					
ADDI	$2, $4, 4					IF	IF	ID	EX	MEM	WB				
SW	$4, 208($2)							IF	ID	ID	EX	MEM	WB		
ADD	$5, $4, $2								IF	IF	ID	EX	MEM	WB	

Stalli:

- ADDI $4, $4, 3: ho uno stallo perché il valore aggiornato del registro R4 deve ancora essere recuperato dalla memoria. Sarà disponibile solo dopo la fase MEM di LW $4, 10($2)
- SW $4, 208($2): ho uno stallo perché il valore di R4 viene aggiornato da ADDI $4, $4, 3 e non vi sono circuiti di bypass che permettano di passare il valore da ADDI a SW. Il dato sarà quindi disponibile dopo la scrittura nei registri che avviene durante la fase WB di ADDI.

ByPass:

- Valore aggiornato di R2 da fase EX di ADD $2, $0, $8 a fase EX di LW $4, 10($2), grazie al circuito di bypass EX ->EX
- Valore aggiornato di R4 da fase MEM di LW $4, 10($2) a fase EX di ADDI $4, $4, 3, grazie al circuito di bypass MEM -> EX
- Valore aggiornato di R4 da fase EX di ADDI $4, $4, 3 a fase EX di ADDI $2, $4, 4, grazie al circuito di bypass EX -> EX
- Valore aggiornato di R2 da fase MEM di ADDI $2, $4, 4 a fase EX di SW $4, 208($2),
- grazie al circuito di bypass MEM -> EX

Questa è la mia risoluzione dell'appello, alcune domande potrebbero mancare o essere incomplete o errate!
Lo carico affinchè possa essere d'aiuto a qualcuno e in caso vogliate correggere contattatemi via facebook con la modifica o fate voi!

Nicolò scapin