ESAME 15 DICEMBRE 2011
versione B

SECONDA PARTE

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: B

ESERCIZIO 2
Si consideri la seguente rappresentazione in virgola mobile a singola precisione (IEEE 754) :
01000010111111000100000000000000
Il numero rappresentato è:
a) -126.125 b) -1.9707031
c) 1.9707031 d) 126.125
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*
- 23 bit per la mantissa (+ 1 bit (implicito) del primo 1)

segno esponente polarizzato mantissa
0 10000101 11111000100000000000000

segno = 0 → +

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

numero = + 1. 11111000100000000000000 x 2^6 = $2^6+2^5+2^4+2^3+2^2+2^1+2^{-3}$ = 126,125

quindi risposta B

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 18% delle istruzioni sono di salto condizionale
- il 8% delle istruzioni sono di salto incondizionale
- il 60% delle istruzioni di salto condizionale hanno la condizione soddisfatta (prese)

Il fattore di velocizzazione della pipeline è:
a) 3.230988 b) 3.508772
c) 3.086420 d) 3.960031
e) nessuna delle risposte precedenti

Soluzione
Percentuali:
- 8% = 0.08 salto incondizionale
- 18% = 0.18 salto condizionale, di cui:
  - 60% = 0.60 preso ==> quindi il 0.18 * 0.60 = 0.108 delle istruzioni totali
  - 40% = 0.40 non preso ==> quindi il 0.18 * 0.40 = 0.072 delle istruzioni totali

Nessun trattamento dei salti. Verrà quindi processata l'istruzione successiva.

- Salti Incodizionati: Ho 1 ciclo di stallo: non ho caricato l'istruzione corretta.
- Salti Codizionati Presi: Ho 2 cicli di stallo: ho caricato l'istruzione errata, ma per scoprirlo devo aspettare la fine di EI.
- Salti Condizionati Non Presi: Non ho nessun ciclo di stallo in quanto è stata caricata l'istruzione corretta.
  (vedi rappresentazione grafica su altri ESAME 15 DICEMBRE 2011- versione C)

Percentuali con cicli di stallo:
- salto incodizionato: 0.08*1 =0.08
- salto condizionato preso: 0.108*2 = 0.216
- salto condizionato non preso: 0.072*0=0

quindi la *frazione di cicli di stallo* = 0.08+0.216+0 = 0.296
Essendovi 4 stadi → k = 4

$$fattore\ di\ velocizzazione = S_k = \frac{1}{1 + frazione\ cicli\ stallo}k = \frac{1}{1 + 0.296}4 = 3{,}086419$$

Quindi risposta C

ESERCIZIO 4:
Si spieghi in dettaglio lo schema per realizzare la moltiplicazione tra numeri reali dello standard IEEE 754.

Svolgimento:
1. controllo dello zero, se uno dei due numeri è zero allora il risultato sarà zero, quindi non è necessario fare ulteriori conti. Se sono entrambi diversi da zero procedo.
2. somma degli esponenti.
3. nell'esponente risultate dal passo precedente è stata duplicata la polarizzazione, in quanto era stata applicata ad entrambi gli esponenti ed avendone fatto la somma la polarizzazione è stata considerata due volte. La polarizzazione va quindi sottratta.
   Se non ci sono underflow od overflow di esponente la procedura continua.
4. moltiplicazione degli operandi

5. normalizzazione del risultato, cioè l'esponente è aggiustato in modo che il bit più significativo della mantissa sia 1.
6. se il prodotto risulta essere in un registro più lungo del formato massimo consentito della virgola mobile, grazie all'utilizzo dei formati estesi per i risultati intermedi, è necessario arrotondarlo. Tale arrotondamento può avvenire per: arrotondamento al più vicino, per eccesso, per troncamento.

ESERCIZIO 5:
Si discuta nel dettaglio in cosa consista il formato variabile per le istruzioni. Se possibile, dare esempi di formati variabili.

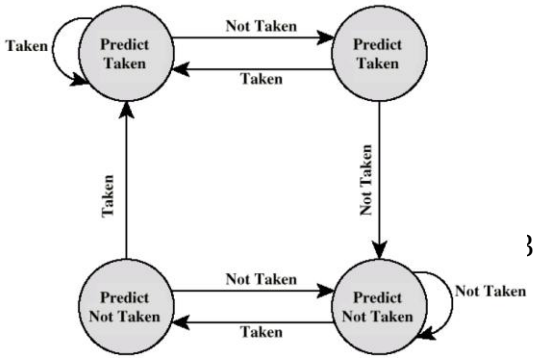
Svolgimento:
Un formato variabile per le istruzioni è un formato che permette una grande varietà di istruzioni ed una grande flessibilità.
Tali istruzioni hanno lunghezze variabili, cioè ogni istruzione ha una lunghezza che dipende dal numero di operandi ed ha un campo opcode generalmente espanso, così da permettere la codifica di un vasto numero di operazioni e metodi di indirizzamento ed inoltre il campo nel campo opcode viene anche specificare il numero di operandi presenti nell'istruzione. Una istruzione con formato variabile può inoltre avere un numero variabile di campi di tipo diverso, come ad esempio nel PENTIUM.
L'utilizzo di questo tipo di istruzioni complica notevolmente la CPU, la rende più costosa, rende difficile il fetch e globalmente diminuisce la reattività sulle operazione più frequentemente utilizzate.
Tale caratteristiche si avvicinano alla filosofia CISC.
Due esempi di formati variabili delle istruzioni li troviamo storicamente nel:

- VAX: un sistema estremamente variabile con lunghezza varabile e formati diversi, in quanto l'opcode può stare su un byte o su due. E' un sistema molto flessibile e potente che facilita il lavoro di programmatori e compilatori, ma il sistema è molto complesso.
- PENTIUM: le istruzioni sono composte da vari pezzi i quali vengono assemblati in base alle necessità. Il risultato sono quindi istruzioni a lunghezza variabile e di vari formati che richiedono una complessa decodifica. Tale approccio è stato utilizzato per mantenere la retro compatibilità.

ESERCIZIO 6:
Nel contesto di una pipeline, descrivere nel dettaglio la tecnica della predizione di salto utilizzando 2 bit di predizione.

Svolgimento:
Con la tecnica della predizione del salto si cerca di prevedere se il salto sarà preso oppure no. Tale predizione può essere effettuata con approcci statici o dinamici.
La predizione a 2 bit è un approccio dinamico in cui si cerca di migliorare la qualità di predizione del salto memorizzando la storia delle istruzioni di salto condizionato di uno specifico programma. In particolare nell'approccio a due bit, in un'apposita locazione di memoria temporanea ad accesso rapido associata allo stadio di fetch della pipeline (la tabelle della storia dei salti), ad ogni istruzione di salto si associano due bit che codificano la storia recente. Avendo a disposizione due bit è possibile memorizzare 4 stati, codificati da 00,01,10,11.
I quattro stati sono predisposti per predire che:

- 00 - salto preso: si ipotizza avverrà un salto. In caso di salto ci si sposta allo stato 01, altrimenti si resta nello stato 00, avendo generato un errore di predizione.

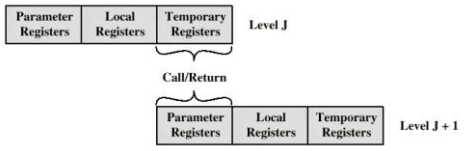


- 01 - salto preso: si ipotizza avverrà un salto. In caso di salto ci si sposta allo stato 00, altrimenti si passa allo stato 10, avendo generato un errore di predizione.
- 10 - salto non preso: si ipotizza non avverrà un salto. In caso di salto ci si sposta allo stato 11, avendo generato un errore di predizione, altrimenti si resta nello stato 10.
- 11 - salto non preso: si ipotizza non avverrà un salto. In caso di salto ci si sposta allo stato 00, avendo generato un errore di predizione, altrimenti si passa allo stato 10.

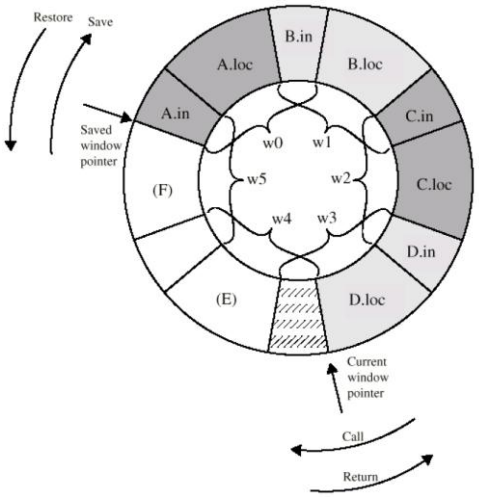
All'inizio tale sistema è posto nello stato 00.
Grazie a tale predizione, nei cicli a singolo loop viene effettuato in tutta l'esecuzione del ciclo un solo errore, nei cicli con un loop annidato in un loop vengono generati globalmente 2 errori.
Tale tecnica ha però un problema: quando si decide di saltare è necessario decodificare l'indirizzo dell'istruzione destinazione del salto a cui seguirà il fetch dell'istruzione stessa. Tale problema può essere evitato anticipando il fetch, ma tale anticipazione richiede delle opportune informazioni che possono essere salvate nella tabella della storia dei salti.

ESERCIZIO 7:
Spiegare in dettaglio come una architettura RISC possa trattare efficientemente la chiamata annidata di procedure.

Svolgimento:
Dall'osservazione che le chiamate di procedura tipicamente coinvolgono pochi parametri e non presentano un grado di annidamento elevata, si è pensato di usare molti gruppi di registri, detti finestre di registri, per gestire le chiamate annidate.
Una chiamata seleziona automaticamente un nuovo gruppo di registri e quando essa è conclusa ed effettua il ritorno, riseleziona il gruppo di registri riferito alla chiamata che l'aveva chiamata.
Ogni gruppo di registri è suddiviso in tre sottogruppi: parametri passati alla procedura, registri che memorizzano il contenuto delle variabili locali della procedura e registri temporanei che gestiscono il ritorno della procedura.



I registri temporanei di un gruppo si sovrappongono perfettamente con quelli che contengono i parametri del gruppo successivo, cioè del gruppo riferito ad una chiamata annidata. Tali registri sono fisicamente gli stessi e ciò permette il passaggio dei parametri senza trasferimento dei dati.
La realizzazione fisica di finestre di registri sovrapposte avviene tramite buffer circolare.
Nel buffer circolare, quando avviene una chiamata il puntatore alla finestra corrente (CWP) viene aggiornato per farlo puntare alla finestra attiva. Se si esaurisce la capacità del buffer, cioè tutte le finestre sono in uso a causa di chiamate annidate, la finestra che per prima è stata inserita nel buffer viene salvata in memoria principale e quindi sovrascritta dalla nuova. Quando una procedura termina, una finestra viene liberata e grazie ad un apposito puntatore (SWP) è possibile ripristinare l'ultima finestra salvata in memoria principale.



ESERCIZIO 8:
Sia data la seguente sequenza di istruzioni assembler, dove i dati immediati sono espressi in esadecimale.

```
LW    $3, 150($0)
ADD   $2, $0, $3
SW    $4, 10($3)
ADDI  $6, $4, 3
ADDI  $2, $2, 4
SW    $6, 208($2)
ADD   $5, $6, $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:

LW	$3, 150($0)	R3 ← mem[150 + [R0]]
ADD	$2, $0, $3	R2 ← [R0] + [R3]
SW	$4, 10($3)	mem[10 + [R3]] ← R4
ADDI	$6, $4, 3	R6 ← [R4] + 3
ADDI	$2, $2, 4	R2 ← [R2] + 4
SW	$6, 208($2)	mem[208 + [R2]] ← R6
ADD	$5, $6, $2	R5 ← [R6] + [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
LW	$3, 150($0)	IF	ID	EX	MEM	WB									
ADD	$2, $0, $3		IF	ID	ID	EX	MEM	WB							
SW	$4, 10($3)			IF	IF	ID	EX	MEM	WB						
ADDI	$6, $4, 3					IF	ID	EX	MEM	WB					
ADDI	$2, $2, 4						IF	ID	EX	MEM	WB				
SW	$6, 208($2)							IF	ID	ID	EX	MEM	WB		
ADD	$5, $6, $2								IF	IF	ID	EX	MEM	WB	

Stalli:

- ADD $2, $0, $3: ho uno stallo perché il valore aggiornato del registro R3 deve ancora essere recuperato dalla memoria. Sarà disponibile solo dopo la fase MEM di LW $3, 150($0).
- SW $6, 208($2): ho uno stallo perché il valore di R6 viene aggiornato da ADDI $6, $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:

- R3 da fase MEM di LW $3, 150($0) a fase EX di ADD $2, $0, $3, grazie al circuito di bypass MEM ➔ EX
- R2 da fase MEM di ADDI $2, $2, 4 a fase EX di SW $6, 208($2), grazie al circuito di bypass MEM ➔ EX

NON HO LA PRESUZIONE DI DIRE CHE QUESTO E' IL CORRETTO MODO DI SVOLGERE IL COMPITO.
LO CARICO PERCHE' POTREBBE ESSERE UTILE AVERE UN CONFRONTO O MAGARI UNO SPUNTO PER RISOLVERE ALCUNI ESERCIZI.
SE VEDETE ERRORI AGGIORNATE PURE IL FILE CON UNA NUOVA VERSIONE CORRETTA E MIGLIORATA.
ABBIATE SPIRITO CRITICO =)

SALUTI

Caesar