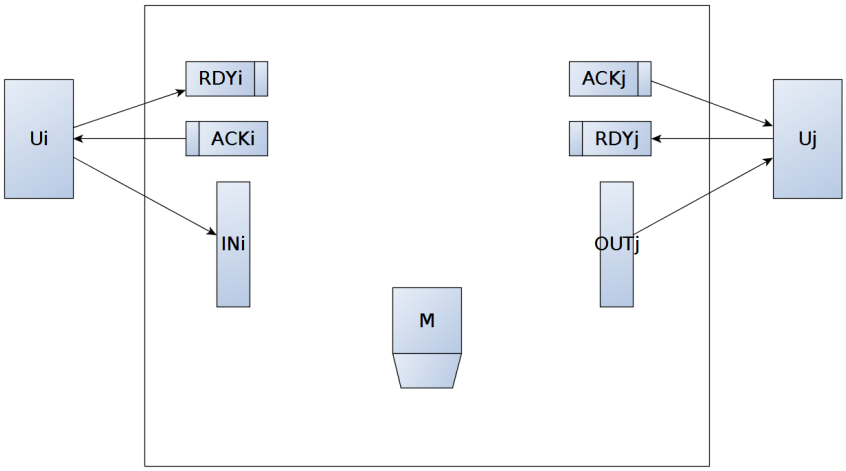
Architettura degli elaboratori – Anno Accademico 2016 – 2017

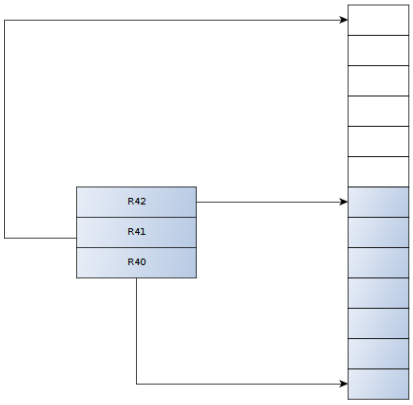
Appello del 4 settembre 2017

Riportare su tutti i fogli consegnati, in alto a destra, nome, cognome, numero di matricola e corso A o B. I risultati e il calendario degli orali saranno pubblicati via WEB sulle pagine dei docente (corso A) e didawiki (corso B) appena disponibili.

Domanda 1



Una unità firmware implementa una coda FIFO che interfaccia 4 unità (U_0, … , U_3) che eseguono inserimenti e 4 unità (U_4, … , U_7) che eseguono estrazioni di singole parole. La priorità delle unità che inseriscono ed estraggono è fissa (U_i ha sempre priorità su U_j (j=i+k, k>0). La coda FIFO è implementata mediante una memoria di 1K posizioni, gestita come un vettore circolare. Tentativi di inserimento con coda piena o estrazione da coda vuota devono comportare il blocco dell'unità che fa la richiesta fino al momento un cui l'operazione non possa essere portata a compimento. Si progetti l'unità cercando di ottimizzare il tempo medio di elaborazione, motivando le scelte effettuate e i risultati ottenuti (si consideri per la memoria $t_a=10t_p$ e per le operazioni aritmetico logiche $t_{alu}=5t_p$).



Domanda 2

In un processore D-RISC non parallelo, si consideri la possibilità di aggiungere operazioni che implementano uno stack in memoria centrale: una **`PUSH Ri`**, che inserisce il contenuto del registro Ri in testa allo stack e una **`POP Ri`** che estrae il primo elemento in cima allo stack restituendolo in Ri. In caso di push su stack pieno o pop da stack vuoto, il valore del registro non deve essere modificato. Per implementare lo stack si assuma che i due registri R40 e R41 contengano rispettivamente come l'indirizzo della base e il del limite superiore per lo stack e il registro R42 sia utilizzato come stack pointer. Si valutino le implicazioni sulla parte operativa dell'unità firmare processore D-RISC e sul ciclo di clock.

Bozza di soluzione

Domanda 1

Ognuna delle unità ha una interfaccia dedicata: RDY_i (indicatore a transizione di livello in ingresso, indica la volontà ad eseguire una operazione) ACK_i (indicatore in uscita, utilizzato per comunicare la fine, con successo, dell'operazione richiesta), IN_i (OUT_i) utilizzato per comunicare il valore da inserire in coda (ricevere il valore estratto dalla coda).

Ad ogni inserzione occorre controllare che la coda non sia piena, e ad ogni estrazione che non sia vuota. Per gestire la coda come un vettore circolare, utilizziamo due registri, puntatore alla fine della coda e puntatore alla testa della coda, oltre ad un registro che mantiene la lunghezza della coda. Inizialmente sono tutti a 0. All'inserzione, si inserisce alla posizione fine e si incrementa il puntatore di fine, se e solo se il numero degli elementi della coda non è già il massimo. All'estrazione, si preleva l'elemento puntato dal registro testa della coda, incrementandolo, se e solo il numero di elementi della coda non è 0. Tutte le operazioni di incremento vengono effettuate modulo 1K (lunghezza della memoria) ovvero X++ deve essere letta come (X+1)%1K→X.

La priorità fissa indicata nel testo si ottiene semplicemente testando i RDY delle unità secondo lo schema:
1--------
01-------
001-----
0001----
00001---
000001--
0000001-
00000001
una combinazione per frase. Va tenuto conto delle situazioni che comportano un blocco dell'operazione richiesta da parte di un'unità con priorità: richiesta di inserimento con coda piena e di estrazione con coda vuota. Nel primo caso, occorre passare in uno stato in cui la priorità è data esclusivamente alle unità che richiedono estrazioni. Nel secondo caso, occorre passare in uno stato in cui la priorità è alla unità che fanno inserimenti, che di fatto è quella di default secondo lo schema proposto.

Detto questo il microcodice si può scrivere come

0. // tutte le X++ devono essere lette come (X+1)%1K→X
   // priorità "standard"
   // nessuna richiesta, si cicla
   (RDY0, … , RDY7, CNT0, OR(CNT) = 00000000--) nop, 0
   // richiesta di inserimento dalla prima unità, coda non piena, inserimento e si ricomincia
   (=1-------0-) IN0→M[FINECODA], FINECODA++, CNT++, reset RDY0, set ACK0, 0
   // richiesta di inserimento dalla seconda unità, coda piena, precendenza alle estrazioni
   (=1-------1-) nop, 1,
   // richiesta dalla seconda unità … etc.
   (=01------0-) IN1→M[FINECODA], FINECODA++, CNT++, reset RDY1, set ACK1, 0
   (=01------1-) nop, 1,
   ….
   // richiesta di estrazione dalla prima unità che estrae, coda non vuota
   (=00001----1) M[INIZIOCODA]→OUT4,INIZIOCODA++, CNT--,

reset RDY4, set ACK4, 0
// richiesta di estrazione dalla prima unità che estrae, coda vuota
(=00001----0) nop, 0
…
// richiesta di estrazione dall'ultima unità che estrae, coda non vuota
(=00000001-1) M[INIZIOCODA]→OUT7,INIZIOCODA++, CNT--,
reset RDY7, set ACK7, 0
// richiesta di estrazione dall'ultima unità che estrae, coda vuota
(=00000001-0) nop, 0

1. // richiesta di estrazione dalla prima unità che estrae, coda non vuota
(RDY0, … , RDY7, CNT0, OR(CNT) = ----1000-1) M[INIZIOCODA]→OUT4,
INIZIOCODA++, CNT--, reset RDY4, set ACK4, 0
// non occorre testare il caso di coda vuota perchè arriviamo alla 1. solo con coda piena
(=----0100-1) M[INIZIOCODA]→OUT5,
INIZIOCODA++, CNT--, reset RDY5, set ACK5, 0
...
// richiesta di estrazione dall'ultima unità che estrae, coda non vuota
(RDY0, … , RDY7, CNT0, OR(CNT) = ----0001-1) M[INIZIOCODA]→OUT7,
INIZIOCODA++, CNT--, reset RDY7, set ACK7, 0

Valutiamo il ciclo di clock.
Per la parte controllo, visto che testiamo 9 variabili di condizionamento contemporaneamente e abbiamo due stati (10 ingressi livello AND) avremo bisogno di due livelli di porte AND. La presenza di 13 frasi richiede anche 2 livelli di porte OR. Dunque il ritardo introdotto dalla parte controllo, sia per TωPO che TσPO sarà pari a $4t_p$.
Per la parte operativa, abbiamo nella stessa frase un accesso alla memoria e un'operazione di incremento. La memoria ha bisogno di un commutatore sugli indirizzi, quindi rappresenta il maggior ritardo con 12tp. Le variabili di condizionamento richiedono 2tp per il calcolo di OR(CNT) visto che CNT è da 11 bit. Tutto questo porta ad un ciclo di clock τ pari a 2tp + 4tp + 12tp + tp = 19tp. Il tempo medio di servizio, in caso di operazione di inserimento con coda non piena ed estrazione con coda non vuota, è di un solo ciclo di clock. Quindi non è possibile fare di meglio. L'unica cosa che si può cercare di fare è ridurre la lunghezza del ciclo di clock. Per questo abbiamo due possibilità:

- utilizzare una memoria a doppia porta per eliminare il commutatore che sceglie fra INIZIOCODA e FINECODA riduce τ di 2tp
- calcolare l'esito di OR(CNT) durante l'aggiornamento di CNT può eliminare i 2tp della variabile di condizionamento. Questa cosa va fatta *cum grano salis*. La cosa evidente di utilizzare un bit assegnato durante l'assegnamento:
  CNT--, OR(CNT) → PIENA
  non funziona perchè l'OR lavorerebbe sul vecchio valore di CNT. Il flag PIENA dovrebbe essere messo a 0 se e solo se quando si va a decrementare CNT il registro vale 1. Dunque quando (CNT11 or … or CNT1) or (not CNT0) vale 0. Questo calcolo richiede 2 tp che tuttavia vanno in parallelo alle altre operazioni della frase. Utilzzare il falg PIENA così calcolato comporta quindi un risparmio di 2tp sulla TωPO.

Applicando entrambe le modifiche, portiamo quindi il ciclo di clock a 15tp.

Domanda 2

Stabiliamo il formato delle istruzioni. Entrambe hanno un unico paramtro di tipo registro. Quindi la rappresentazione in memoria prevederà 8 bit per il codice operativo e 6 bit per il numero del registro utilizzato come parametro. I rimanenti 18 rimarranno inutilizzati.
La ch0 e ch1 non cambiano. Va fatto vedere come sono fatte le microistruzioni che implementano la PUSH e la POP. Per la PUSH, assumendo che lo stack pointer punti alla prima posizione libera, dobbiamo semplicemente controllare di non aver terminato le posizioni dello stack e quindi richiedere la scrittura in memoria del valore del registro, incrementando lo stack pointer. In caso di POP dobbiamo controllare che lo stack non sia vuoto e quindi leggere nel registro il valore dalla memoria puntata dallo stack pointer decrementato.
Il punto critico è come testare le condizioni che portano al fallimento delle operazioni. Testare la fattibilità della push richiede di testare che R[42] sia diverso da R[41]+1. Testare che lo stack non sia vuoto richiede di testare che R[42] sia diverso da R[40]. Si potrebbe integrare il microcodice dell'interprete fw come segue:

push0. (INT,zero(R[42]-(R[41]+1)=00) Reg[IR.Ra]→DATAOUT, R42→IND, "write"→OP,
set RDYm, push1
(=01) nop, IC+1→ IC, ch0
(=11) nop, IC+1→ IC, tratt_int
push1. (ACKm,INT,or(ESITO)=0--) nop, push1
(=100) R42+1→R42, IC+1→ IC, ch0
(=110) R42+1→R42, IC+1→ IC, tratt_int
(=1-1) tratt_ecc
pop0. (INT, zero(R[42]-R[40])=00) "read"→OP, M42-1→IND, set RDYm, pop1
(=01) nop, IC+1→ IC, ch0
(=11) nop, IC+1→ IC, tratt_int
pop1. (ACKm,INT,or(ESITO)=0--) nop, pop1
(=100) R42-1→R42, IC+1→ IC, ch0
(=110) R42-1→R42, IC+1→ IC, tratt_int
(=1-1) tratt_ecc

Tuttavia, l'introduzione di variabili di condizionamento complesse comporta un allungamento del ciclo di clock e dunque un rallentamento del tempo medio di elaborazione. Distribuendo test e operazioni fra due microistruzioni, come già avviene per le istruzioni di salto condizionale possiamo risolvere il problema:

push0. zero(R[42]-(R[41]+1)→F, push1
push1. (INT,F=00) Reg[IR.Ra]→DATAOUT, R42→IND, "write"→OP, set RDYm, push2
(=01) nop, IC+1→ IC, ch0
(=11) nop, IC+1→ IC, tratt_int
push2. (ACKm,INT,or(ESITO)=0--) nop, push2
(=100) R42+1→R42, IC+1→ IC, ch0
(=110) R42+1→R42, IC+1→ IC, tratt_int
(=1-1) tratt_ecc
pop0. zero(R[42]-R[40])→F, pop1
pop1. (INT, F)=00) "read"→OP, M42-1→IND, set RDYm, pop2
(=01) nop, IC+1→ IC, ch0
(=11) nop, IC+1→ IC, tratt_int
pop2. (ACKm,INT,or(ESITO)=0--) nop, pop2
(=100) R42-1→R42, IC+1→ IC, ch0

(=110) R42-1→R42, IC+1→ IC, tratt_int
(=1-1) tratt_ecc

Questo riporta alla situazione in cui tutte le variabili di condizionamento sono semplici. Va notato che il test per stack pieno richiede due alu in cascata. Questo normalmente non accade nelle istruzioni dell'interprete fw originale e potrebbe comportare un allungamento del TσPO. Questo problema potrebbe essere semplicemente risolto facendo sì che R[41] punti alla prima posizione *oltre* lo spazio riservato allo stack.