ARCHITETTURE AVANZATE DEI CALCOLATORI A.A. 2007/08
Prima prova in itinere del 9/11/2007
Compito A

Domanda 1
Descrivere il concetto di speculazione hardware ed il meccanismo del ReOrder Buffer.

Domanda 2
Descrivere la tecnica del salto ritardato e proporre un esempio della sua applicazione nel processore MIPS.

Esercizio 1
Si consideri il processore MIPS con pipeline (il cui schema è allegato) e la seguente sequenza di codice.

```
add $2, $1, $2
sw $2, 0($3)
lw $2, 4($3)
add $2, $2, $2
addi $5, $5, 4
sw $2, 0($5)
```

Si individuino le criticità sui dati e si indichi il valore assunto dai segnali di controllo ForwardA, ForwardB, ALUSrc, PCWrite e IF/ID.Write per i cicli di clock da 3 a 9. Si assuma che il PC sia inizializzato alla prima istruzione della sequenza e che i cicli di clock 1 e 2 siano già stati eseguiti (ovvero la prima istruzione `add` è nello stadio EXE). Si usi il simbolo X nel caso in cui il segnale di controllo abbia valore `don't care`.

Esercizio 2
Sia dato il seguente ciclo di un programma in un linguaggio ad alto livello:

```
do {
  if (A[i] >= 0) {
    C[i] = B[i] + C[i];
    B[i] = B[i] + K;
  }
  else {
    B[i] = 0;
    C[i] = 0;
  }
  i++;
} while (i != N);
```

Il programma sia stato compilato nel codice in assembler MIPS. Si supponga che i registri `$t5`, `$t6` e `$t7` siano stati inizializzati rispettivamente ai valori `0`, `N*4` e `K`. I simboli `BASEA`, `BASEB` e `BASEC` sono costanti a 16 bit, prefissate. La frequenza di clock del processore vale 2 GHz.

```
Loop: beq $t5, $t6, End
      lw $t1, BASEA($t5)
      slt $t0, $t1, $zero          # if ($t1<$zero) $t0=1; else $t0=0;
      bne $t0, $zero, L1
      lw $t2, BASEB($t5)
      lw $t3, BASEC($t5)
      add $t3, $t3, $t2
      sw $t3, BASEC($t5)
      add $t2, $t2, $t7
      sw $t2, BASEB($t5)
      j L2
L1:   sw $zero, BASEB($t5)
      sw $zero, BASEC($t5)
L2:   addi $t5, $t5, 4
      j Loop
End:
```

Si supponga che il ciclo venga iterato N volte (con N grande a piacere) tramite una pipeline MIPS a 5 stadi. Si consideri una generica iterazione del ciclo e si ipotizzi che nel 60% dei casi risulti `(A[i] >= 0)`. Si consideri l'istruzione `j` come un'istruzione di branch (`beq` o `bne`) la cui condizione viene sempre valutata vera.

a) Si supponga che la pipeline sia priva di ottimizzazioni.
   - Nel diagramma a ciclo multiplo della pipeline individuare le criticità sui dati di tipo RAW e le criticità sul controllo ed indicare il tipo di criticità.
   - Indicare il numero di stalli necessari a risolvere le criticità individuate.
   - Calcolare il numero totale di stalli inseriti, il CPI asintotico ed il throughput asintotico espresso in MIPS.

b) Si supponga che nella pipeline siano state introdotte le seguenti ottimizzazioni:

- nel banco dei registri è possibile la lettura e la scrittura nello stesso ciclo di clock;
- forwarding dei dati;
- anticipazione del salto nello stadio ID per le istruzioni `beq`, `bne` e `j`.
    - Nel diagramma a ciclo multiplo della pipeline individuare le criticità sui dati di tipo RAW e le criticità sul controllo rimaste ed indicare il tipo di criticità; indicare i percorsi di forwarding utilizzati, specificando i registri di pipeline coinvolti.
    - Indicare il numero di stalli necessari a risolvere le criticità rimaste.
    - Calcolare il numero totale di stalli inseriti, il CPI asintotico ed il throughput asintotico espresso in MIPS.

Esercizio 3

Si consideri il seguente frammento di codice in linguaggio C, dove `A` e `B` sono due vettori di interi:

```
for (i=0; i<N; i++) {
      A[i] = A[i] + 2*B[i];
      B[i] = 0;
}
```

Il sorgente sia stato compilato nel seguente codice in assembler MIPS. Si assuma che i registri `$t0` e `$t1` siano stati inizializzati rispettivamente ai valori `0` e `N*4`. I simboli `BASEA` e `BASEB` sono costanti a 16 bit, prefissate.

```
Loop: lw $t3, BASEB($t0)
      sll $t3, $t3, 1
      lw $t2, BASEA($t0)
      add $t3, $t3, $t2
      sw $t3, BASEA($t0)
      sw $zero, BASEB($t0)
      addi $t0, $t0, 4
      bne $t0, $t1, Loop
Exit:
```

a) Effettuare il loop unrolling (con un fattore di srotolamento pari a 2) e riordinare il codice sul processore MIPS scalare con pipeline ottimizzata allo scopo di massimizzare le prestazioni.
    - Calcolare il valore del CPI per una generica iterazione del ciclo originale prima di effettuare il loop unrolling.
    - Calcolare il valore del CPI per una generica iterazione del ciclo modificato dopo aver effettuato il loop unrolling ed il riordinamento.

b) *(facoltativo)* Indicare un riordinamento del ciclo originale (prima del loop unrolling) che, mantenendo la stessa semantica del frammento, minimizzi il numero di stalli sul processore MIPS scalare con pipeline ottimizzata.