Cognome e Nome: _____ **Matricola:** _____ **Mod 2:** ☐ **Mod 1+2:** ☐

Architettura degli Elaboratori (modulo II)

(Compito 21 Giugno 2021)

Esercizio 1

Individuare le dipendenze RAW (Read After Write) nel seguente codice MIPS:

```
1.   lw    $2, 25($27)
2.   add   $7, $2, $3
3.   sw    $4, 0($2)
4.   and   $4, $4, $7
5.   ori   $4, $4, 5
```

Rispondere alle seguenti domande rispetto alle diverse caratteristiche di una CPU pipeline a 5 stadi:

1. pipeline senza forwarding e con register file speciale (che nella prima parte del ciclo scrive e nella seconda parte del ciclo legge i registri). La CPU **non** ha hazard detection unit, per cui la gestione delle dipendenze è affidata al software: è necessario inserire le `nop` opportune nel codice.
   Disegnare il diagramma di esecuzione del codice con l'aggiunta delle `nop` e calcolare il no. di cicli totali spesi per l'esecuzione. Specificare anche come vengono risolte le varie dipendenze grazie all'inserimento delle `nop`.
2. pipeline con forwarding, con un register file speciale e con hazard detection unit. Disegnare il diagramma temporale di esecuzione, ed evidenziare come vengono risolte le varie dipendenze. In particolare, per i forwarding, da quale registro intermedio a quale stadio.

Soluzione

Le dipendenze RAW sono le seguenti:
$1 \rightarrow 2$ ($2)
$1 \rightarrow 3$ ($2)
$2 \rightarrow 4$ ($7)
$4 \rightarrow 5$ ($4)

Consideriamo il caso 1.:

```
                        1    2    3    4    5    6    7    8    9    10   11   12   13   14
1. lw  $2, 25($27)    | IF | ID | EX | ME | WB |
   nop                     | IF | ID | EX | ME | WB |
   nop                          | IF | ID | EX | ME | WB |
2. add $7, $2, $3               | IF | ID | EX | ME | WB |
3. sw  $4, 0($2)                     | IF | ID | EX | ME | WB |
   nop                               | IF | ID | EX | ME | WB |
4. and $4, $4, $7                         | IF | ID | EX | ME | WB |
   nop                                    | IF | ID | EX | ME | WB |
   nop                                         | IF | ID | EX | ME | WB |
5. ori $4, $4, 5                                    | IF | ID | EX | ME | WB |
```

La porzione di codice impiega 14 cicli per essere completato.
Le dipendenze sono risolte come segue:
$1 \rightarrow 2$ ($2): Register file speciale
$1 \rightarrow 3$ ($2): ID di 3 avviene dopo il WB di 1
$2 \rightarrow 4$ ($7): Register file speciale
$4 \rightarrow 5$ ($4): Register file speciale

Consideriamo il caso 2., con register file speciale, forwarding e hazard detection unit.

```
                        1    2    3    4    5    6    7    8    9    10
1. lw  $2, 25($27)    | IF | ID | EX | ME | WB |
2. add $7, $2, $3          | IF | ID |<ID>| EX | ME | WB |
3. sw  $4, 0($2)                | IF |<IF>| ID | EX | ME | WB |
4. and $4, $4, $7                    | IF | ID | EX | ME | WB |
5. ori $4, $4, 5                          | IF | ID | EX | ME | WB |
```

Per cui la porzione di codice impiega 10 cicli.
Le dipendenze sono risolte come segue:
$1 \rightarrow 2$ ($2): Forwarding da registro EX/ME a EX dell'istruzione 2
$1 \rightarrow 3$ ($2): Register file speciale
$2 \rightarrow 4$ ($7): Forwarding da registro ME/WB a EX dell'istruzione 4
$4 \rightarrow 5$ ($4): Forwarding da registro EX/ME a EX dell'istruzione 5

Esercizio 2

Dato un sistema di memoria virtuale paginata con pagine da 2048 B, e una Page Table di 2^{21} ingressi, calcolare la dimensione dell'*indirizzo fisico* e dell'*indirizzo virtuale*, considerando che la dimensione delle entry della Page Table è di 3 B, e include 4 bit di valid, dirty, ecc.
Calcolare la *dimensione (parte dati)* della una cache associativa a 2 vie, che fa parte dello stesso sistema di memoria, considerando che TAG = 16 b e BLOCK OFFSET = 4 b.
Dati la seguente sequenza di indirizzi fisici di accesso alla cache (espressi in esadecimale), determinare se trattasi di miss (indicare se trattasi di conflitto) o hit, considerando che la cache sia all'inizio vuota (tutti gli ingressi siano non validi).

```
0F AA BA 00
0F AA BA 04
00 A0 BA 04
00 B0 BA 04
```

Qual è la condizione finale della cache?

Soluzione

Page_Offset = $\log_2 2048$ = 11 b.
PPN = 3 B - 4 b = 24 - 4 = 20 b.
VPN = $\log_2 2^{21}$ = 21 b.
Indirizzo virtuale = VPN + Page_Offset = 21 + 11 = 32 b.
Indirizzo fisico = PPN + Page_Offset = 20 + 11 = 31 b.
INDEX = Ind_virtuale - TAG - OFFSET = 31 - 16 - 4 = 11 b.
Dim_blocco = 2^{OFFSET} = 2^4 B = 16 B
Dim_cache = 2^{INDEX} * n_vie * Dim_blocco = 2^{11} * 2 * 16 B = 2^{16} B = 64 KB.

tag	**ind**	**off**	
$0FAA_1$	$011A0$	0	miss (copia nella prima via)
$0FAA_1$	$011A0$	4	hit
$00A0_1$	$011A0$	4	miss (copia nella seconda via)
$00B0_1$	$011A0$	4	miss (conflitto, copia nella prima via - polita LRU)

INDEX = $011A0$ 1^a via: TAG=$00B0_1$ 2^a via: TAG=$00A0_{011}$

Esercizio 3

Tradurre in assembly MIPS la seguente porzione di codice C, dove la funzione `sum_even_pos` somma gli elementi in posizione pari di un array di interi:

```
int ar[10] = {1,-1,2,-2,4,-4,5,-5,6,-6};

int sum_even_pos(int a[], int n) {
    int i, acc;
    acc = 0;
    for (i=0; i<n; i += 2)
       acc += a[i];
    return acc;
}

int main() {
    int ret;
    ret = sum_even_pos(ar, 10);
    printf("%d", ret);
}
```

1. Tradurre in C con `if-goto` e `goto`.
2. Tradurre in assembly MIPS[1].

Soluzione

Il codice con `if-goto`:

```
#include <stdio.h>

int ar[10] = {1,-1,2,-2,4,-4,5,-5,6,-6};

int sum_even_pos(int a[], int n) {
   int i, acc;
   acc = 0;
   i = 0;
init_for:
   if (i>=n) goto exit_for;
   acc += a[i];
   i += 2;
   goto init_for;
```

[1] *E' possibile usare pseudo-istruzioni come `blt`, `ble`, `bgt` e `bge`. Per stampare un intero con `syscall`, `$v0=1` e argomento in `$a0`.*

```
exit_for:
   return acc;
}

int main() {
   int ret;
   ret = sum_even_pos(ar, 10);
   printf("%d", ret);
}
```

Traduzione MIPS:

```
.data

ar: .word 1, -1, 2, -2, 4, -4, 5, -5, 6, -6    # int ar[10] = {1,-1,2,-2,4,-4,5,-5,6,-6};

.text

sum_even_pos:               # int sum_even_pos(int a[], int n)
                            # int i, acc;
   li $t0, 0                # acc = 0;   (reg. temporaneo, da non salvare sullo stack)
   li $t1, 0                # i = 0;     (reg. temporaneo, da non salvare sullo stack)
init_for:
   bge $t1, $a1, exit_for   # if (i>=n) goto exit_for;
   sll $t2, $t1, 2          # t2 <- i*4
   add $t2, $a0, $t2        # t2 <- &a[i]
   lw $t3, 0($t2)           # t3 <- a[i]
   add $t0, $t0, $t3        # acc += a[i];
   addi $t1, $t1, 2         # i += 2;
   j init_for               # goto init_for;
exit_for:
   move $v0, $t0            # return acc;
   jr $ra

main:                       # int main()
   addi $sp, $sp, -4
   sw $ra, 0($sp)
                            # int ret;
   la $a0, ar
   li $a1, 10
   jal sum_even_pos         # ret = sum_even_pos(ar, 10);
   move $a0, $v0
   li $v0, 1
   syscall                  # printf("%d", ret);

   lw $ra, 0($sp)
   add $sp, $sp, 4
   jr $ra
```

Domande

1. Dopo un cache miss è possibile che si verifichi un page-fault (motivare)?
   Dopo la risoluzione di un page-fault, la page-table viene aggiornata, e l'istruzione rieseguita. Potrebbe verificarsi un cache miss?
2. Come può essere modellato l'accesso al disco? Le prestazioni del disco (banda di trasferimento reale dei dati) dipendono solo dall'hardware o dalla disposizione dei dati?
3. Discutere uso e caratteristiche dei bus sincroni e asincroni.
4. In alcuni casi certe istruzioni assembly MIPS non possono essere tradotte direttamente con una singola istruzioni macchina, per cui diventano una sorta di *pseudo-istruzioni*. Ad esempio `lw r1, costante(r2)`. Discutere.