

# CALCOLATORI ELETTRONICI M

## ARCHITETTURA DEI CALCOLATORI ELETTRONICI M

14/02/2022 – Online – Esercizio 1 & 2

**Tempo disponibile: 150 minuti**

– **Link al testo in formato pdf**

– **Link ai fogli per lo svolgimento Es.1, Es.2**

Si fornisca la soluzione al compito riempiendo il foglio di calcolo disponibile ai link sopra riportati e consegnandolo allegandolo al quiz. In caso non sia possibile usare il foglio di calcolo si inviino sempre allegandole al quiz le foto alla soluzione cartacea. Inserire in tutti i nomi dei fili

Cognome\_Nome\_Matricola altrimenti il file non verrà valutato.

### Esercizio 1 - Descrizione Sistema

Un RV32IMAFD con  $T_{ck} = T$  dispone di **tre** unità funzionali, A, M e D “multiciclo” capaci di eseguire le seguenti istruzioni su operandi in virgola mobile:

**A: FADD (in 2T)      M: FMUL (in 3T)      D: FDIV (in 4T).**

Si consideri il seguente frammento di codice e si faccia l’ipotesi che in T1 si abbia  $F_i = i$  per ogni valore di  $i$  compreso tra 0 e 31.

```
fadd.s f1, f2, f3
fmul.s f5, f1, f4
fdiv.s f5, f16, f8
fdiv.s f6, f8, f1
fmul.s f7, f8, f5
fdiv.s f8, f16, f5
```

1. Si stimi il numero di colpi di clock al di sotto del quale non è possibile scendere nell’esecuzione del codice assegnato, qualunque sia il numero di RS, CRB e stadi di Fetch e Decode disponibili, nell’ipotesi che esista una sola unità di tipo **A**, una sola di tipo **D** e una sola di tipo **M** (tre unità funzionali in tutto) e si motivi la risposta (*In questa stima non si tenga conto delle dipendenze nel codice*) (**1 punto**)
2. Si disegni il grafo delle dipendenze e si deduca il numero minimo di periodi di clock necessario a eseguire il codice assegnato tenendo conto solo delle dipendenze trovate (**1 punto**).
3. Si mostri la dinamica dell’esecuzione nel caso della CPU considerata nel punto 1 con 1 CDB, uno stadio di IF e uno di ID, e **2 RS** per ognuna delle tre unità funzionali (*si ipotizzi che, in caso di conflitti sul CDB, la fase di WB dell’unità D abbia la priorità sulle altre*) (**3 punti**).
4. E’ possibile ridurre il numero di periodi di clock necessario ad eseguire il codice assegnato rispetto al valore risultante dalla risposta al punto 3? Si punti al numero minore possibile di periodi di clock e al numero minimo di modifiche, supponendo di poter apportare solo le seguenti modifiche all’architettura:
  - o aumento di RS da due a tre in una o più unità funzionali
  - o raddoppio o triplicazione del CDB per poter eseguire due o tre fasi di WB per clock
  - o aggiunta di una o più unità funzionali con due RS

*In caso di raddoppio di una unità funzionale, si ipotizzi di fare lo scheduling a rotazione sulle due unità funzionali uguali.*

Quali modifiche converrebbe apportare? Si motivi la risposta e si disegni la nuova dinamica di esecuzione. (3 punti).

5. Si disegni il film del registro f5 nel caso della dinamica di esecuzione di cui al punto 4. (2 punti).

## Esercizio 2 - Descrizione Sistema

Un sistema multicores S è composto da 2 cores RV64GC dotati ciascuno di una cache L1 dati privata a 2 vie da 2KiB e linee da 32 bytes, gestita con stato MESI e con politica di scrittura Write Allocate in caso di miss.

Nel sistema S esegue un'applicazione che applica un filtro (convoluzione) con pesi contenuti nell'array h agli elementi dell'array x e ne salva il risultato nell'array y. Il calcolo del filtro viene eseguito dal core 0 al termine del quale il core 1 legge i valori per calcolarne la media.

L'array x è composto da 256 float a singola precisione a partire dall'indirizzo 0x0101 0000.

L'array h è composto da 8 float a singola precisione a partire dall'indirizzo 0x0102 0000.

L'array y contiene 248 elementi (256-8) a partire dall'indirizzo 0x0103 0000.

Con l'elemento y[0] che corrisponderà alla convoluzione dei primi 8 elementi di x con i coefficienti h

Al fine di eseguire il calcolo del vettore y l'applicazione in esecuzione su S esegue interamente sul core 0. Al termine del calcolo del vettore y, il core 1 esegue una routine che legge tutti gli elementi di y in sequenza per calcolarne la media.

### RISC-V assembly (ridotto)

```

00| .L0:
01|     addi x11, x16, x0 # x11 = &h[0]
02|     addi x13, x0, 7   # x13 = 7
03| .L1:
04|     flw f1, 0(x10) # f1 = x[i-k]
05|     flw f2, 0(x11) # f2 = h[k]
06|     fmul.s f1, f2, f1
07|     fadd.s f0, f0, f1
08|     addi x10, x10, -4
09|     addi x11, x11, 4
10|     addi x13, x13, -1
11|     bnez x13, .L1
12| .return:
13|     fsw f0, 0(x12) # *y[i] = f0
14|     addi x15, x15, 4
15|     add x10, x15, x0 # x10 = &x[i+8]
16|     addi x12, x12, 4
17|     addi x14, x14, -1
18|     bnez x14, .L0
19|     ret

```

B1

B0

$$y[i] = \sum_{k=0}^7 x[i+7-k] * h[k]$$

Il codice di fianco corrisponde al compilato del blocco di codice B0 che implementa un filtro FIR che calcola i valori di y[i] a partire da x[i] e con pesi h[k]. Ciascun valore y[i] è calcolato facendo la convoluzione di x[i] con h[k] B1.

In questo codice:

- x16 contiene l'indirizzo dell'elemento h[0]
- x15 contiene l'indirizzo dell'elemento x[i]
- x14 conta gli elementi di y ancora da calcolare (i)
- x13 conta le iterazioni ancora da eseguire per la convoluzione (k)
- x12 contiene l'indirizzo dell'elemento y[i]
- x11 contiene l'indirizzo dell'elemento h[k]
- x10 contiene l'indirizzo dell'elemento x[i+8-k]
- f0, f1 e f2 sono registri floating-point temporanei.

```

for (unsigned int i=0; i<248; i++) #B0
    for (unsigned int k=0; k<7; k++) #B1
        y[i] += x[i+7-k] * h[k];

```

### Quesiti:

2.a) Facendo riferimento al sistema S, si completi la mappa delle memoria con i tre vettori x, y e h. (Punti.3)

Si analizzi la dinamica della cache dati, e, tenendo ben presente che il sistema ha un solo caching agent, si risponda in modo preciso, schematico, conciso e tabellare ai seguenti quesiti: (14 Punti)

2.b) Quali sono gli indici di set e linea, e i tag associati agli array x, y e h? Quante linee di cache occuperanno del loro insieme? Possono essere contenute interamente nella cache L1 privata del core 0?

2.c) Si consideri la dinamica della cache nel calcolo dell'elemento y[0] svolto dal core0 - disegnando lo stato MESI, il contenuto della cache e il valore del bit LRU dopo ogni operazione di accesso alla memoria (Load e Store). Una sola iterazione del blocco di codice B1 – istruzioni 00 – 13 incluse.

**2.d)** Si mostri lo stato MESI, il contenuto e il valore del bit LRU della cache al termine del calcolo di tutti gli elementi di  $y$ , considerando i calcoli svolti solo dal core0. Si riporti lo stato delle prime due e delle ultime due linee di cache per il core. Si indichi il numero di miss, il numero di accessi, si calcoli la miss rate e si calcoli il numero di cicli di WB.

**2.e)** Si calcoli l'arithmetic intensity associata al blocco di codice B1?

**2.f)** Si mostri lo stato MESI, il contenuto e il valore del bit LRU della cache per entrambi i core al fine dell'esecuzione del codice sul core1 – lettura di tutti gli elementi di  $y$  e calcolo della media. Facendo l'ipotesi che il core1 esegua dopo il termine dell'esecuzione sul core0. Si riporti lo stato della prima e dell'ultima linea di cache.