1. La funzione prende come input due interi e restituisce un intero dopo aver eseguito alcune operazioni. Il compilatore fornisce la seguente traduzione di INTEL Assembly, che è incompleta. Le righe X1 e X2 vengono omesse e viene chiesto di indicare quale delle coppie proposte corrisponde alle righe corrette. (Sono rispettate le convenzioni di chiamata per argomenti e valore di ritorno specificate dall'ABI discusse durante le lezioni).

```
int funzione(unsigned int num, int e) {
      int t = num + num;
      for(int i = 0; i < t; i++) {
            numero = numero * e;
      }
      restituisce num + t;
}
```

```
      funzione:
            X1
            testl %eax, %eax
            jle .L2
            xorl  %edx, %edx
      .L3:
            addl  $1, %edx
            imull %esi, %edi
            X2
            jne .L3
      .L2:
            addl  %edi, %eax
            ret
```

- X1: `leal  (%rdi,%rdi), %eax`
  X2: `cmpl  %edx, %eax`
- X1: `leal  (%rdi,%rdi), %ecx`
  X2: `cmpl  %edx, %ecx`
- X1: `movl  (%rsi, %rsi), %eax`
  X2: `cmpl  %edx, %eax`
- X1: `movl  (%rdi,%rdi), %edx`
  X2: `cmpl  %edx, %ecx`
- Nessuna delle risposte

2. La funzione sumV prende in input gli indirizzi di tre array interi di u,v,z e la dimensione degli array, il compito è sommare gli elementi nella i-esima posizione di u e v e mettere il risultato della somma nella i-esima posizione di z. Il compilatore fornisce la seguente traduzione nell'assembly ARM che è incompleta: le righe X1, X2 e X3 vengono omesse. (Sono rispettate le convenzioni dichiarate per argomenti e valore di ritorno specificate dall'ABI discusse durante la lezione). Quale delle proposte X1, X2 e X3 corrisponde alle righe corrette?

```
void sumV(int*u, int*v, int*z, unsigned int size){
      for (int i = size - 1; i >= 0; i--){
            *(z+i) =*(v+ i) +*(u+i);
      }
      ritorno;
}

sumV:
      stmfd sp!,{r11, lr}
      X1
      ldmfdmi sp!,{r11, pc}
.LBB01:
      X2
      aggiungi r3, r3, r12
      str r3, [r2, lr, lsl #2 ]
      sub lr, lr, # 1
      cmn lr, #1
      X3
```

- X1: `mov lr, sp`
  `subs lr, r3, #1`
  X2: `ldr r11, [r1, lr,#2 ]`
  `lslldr r12, [r0, lr, lsl #2 ]`
  X3: `bgt . LBB01`
  `ldmfd sp!,{r11, pc}`

- X1: `mov r11, sp`
  `sub lr, r11, #1`
  X2: `ldr r12, [lr, r1,#2 ]`
  `lslldr r3, [lr, r0, lsl #2 ]`
  X3 : `beq .LBB01`
  `ldmfd sp!,{r11, pc}`

- X1: `mov r11, sp`
  `subs lr, r3, #1`
  X2: `ldr r12, [r1, lr,#2 ]`
  `lslldr r3, [r0, lr, lsl # 2 ]`
  X3: `bgt .LBB01`
  `ldmfd sp!,{r11, pc}`

- ```
  X1:   sub lr, r3, #1
        mov r11, sp
  X2:   ldr r3, [lr, r1,#2 ]
        lslldr r12, [lr, r0 , lsl #2 ]
  X3:   ldmfd sp!,{r11, pc}
        bgt .LBB01
  ```

- Nessuna delle risposte

3. Scegliere l'affermazione corretta sulle *librerie statiche*:

- Possono essere usate da programmi C, ma non da programmi scritti in linguaggio assembly.
- Nessuna delle altre risposte.
- Sono usati dall'Assembler per implementare macro/pseudo-istruzioni.
- In realtà sono collegati al programma solo quando viene caricato (in caso di non-lazy linking) o eseguito (in caso di lazy linking).
- Vengono utilizzati durante il collegamento, ma non vengono utilizzati per caricare o eseguire l'eseguibile finale.

4. Fare riferimento alla Figura 1. Indicare quale delle seguenti alternative è la più accurata per spiegare l'uso di PCSrc:

- Nessuna delle risposte
- La command line PCSrc viene utilizzata per verificare se esistono le condizioni per un salto condizionato e la command line RegWrite è utilizzata per selezionare l'indirizzo corretto per il salto
- La command line RegWrite viene utilizzata per verificare se esistono le condizioni per un salto condizionato e la command line PCsrc viene utilizzata per selezionare l'indirizzo corretto per il salto
- La command line PCSrc viene utilizzata per selezionare l'indirizzo corretto per il salto che viene deciso in base al valore di ALUOp
  ```

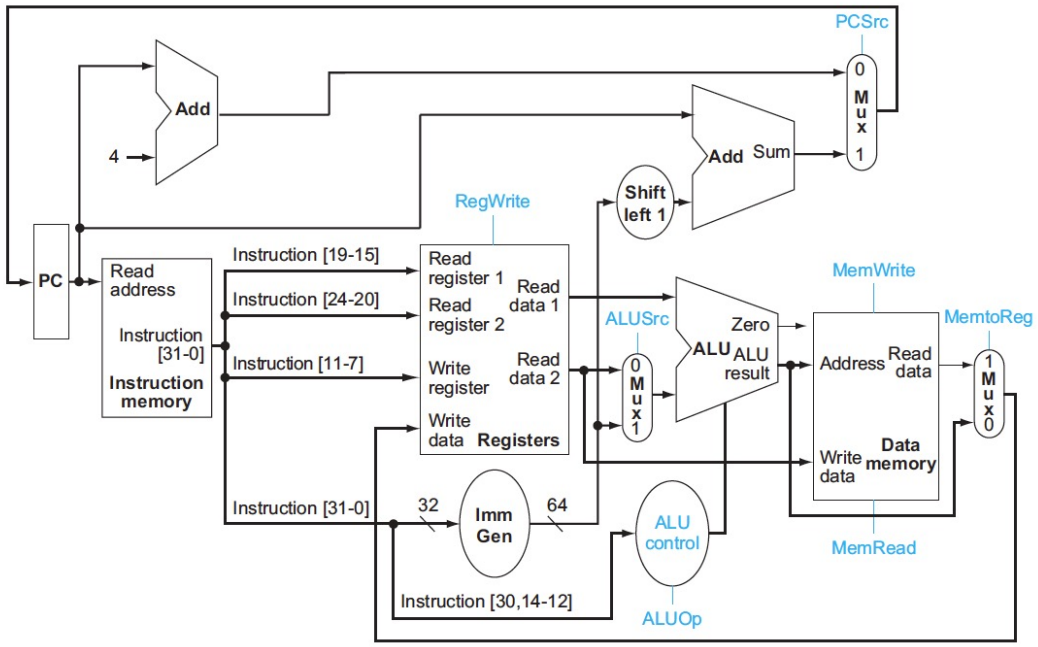

Figura 1: Rappresentazione schematica del datapath

5. Si consideri una CPU con 2 cache separate per dati e istruzioni. Il CPI ideale della CPU è 3, la cache delle istruzioni ha una frequenza di miss dell'1,5% e la cache dei dati ha una frequenza di miss del 5%. Supponendo che una miss cache richieda 150 cicli di clock per essere servito e che il 30% delle istruzioni acceda ai dati in memoria, il CPI effettivo (il numero di cicli necessari in media per eseguire un'istruzione tenendo conto degli stalli per l'accesso alla RAM) è .

- 7.5
- 5.25
- 5.7
- 3.5
- Nessuna delle risposte

6. Consideriamo una CPU in cui le 5 fasi di esecuzione di un'istruzione sono 500ps, 200ps, 500ps, 200ps e 100ps. L'aumento delle prestazioni che puoi aspettarti utilizzando una pipeline è:

- 2 volte
- 4 volte
- 3 volte
- 2,5 volte
- Nessuna delle risposte