

Cognome e nome:

Matricola:

1. [8] Modificare la pipeline in Figura 1 perché diventi una pipeline superscalare. Spiegare la ragione e lo scopo di **tutte le modifiche** più rilevanti da apportare ai diversi stadi. Che differenza c'è tra pipeline super-scalare e pipeline dotata di VLIW? Quali sono i vantaggi e gli svantaggi di un approccio rispetto all'altro. Qual è il migliore e perché? Descrivere come funzionano le seguenti tecniche e dire se sono tecniche principalmente **software** o **hardware** e perché. In alcuni casi la risposta corretta può essere entrambi gli approcci. Identificare quali sono i **punti forti** ed i **punti deboli**.

- a) Predizione dei salti
- b) Branch prediction buffer
- c) Speculazione
- d) Parallelizzazione dell'esecuzione
- e) Pipeline superscalari
- f) Esecuzione fuori ordine
- g) Reservation station
- h) Buffer di riordino
- i) Ridenominazione dei registri
- j) Issue

2. [2] Come si implementa l'esecuzione vettoriale in una pipeline super-scalare? Mostrarlo modificando un cammino di esecuzione di una pipeline non dotata di esecuzione vettoriale.

3. [2] Descrivere come funzionano le seguenti tecniche e dire se sono tecniche principalmente **software** o **hardware** e perché. In alcuni casi la risposta corretta può essere entrambi gli approcci. Identificare quali sono i **punti forti** ed i **punti deboli**.

- a) Superpipeline
- b) Branch delay slot
- c) Hazard
- d) Bolla
- e) Stallo

4) [7] Data la CPU N. 2, specificare il contenuto di **tutte le linee** (dati e controllo), quando è in esecuzione il seguente segmento di codice:

```
0x00000400 add $s5, $t2, $t1
0x00000404 lw $s1, 8($s0)
0x00000408 and $t4, $s5, $s1
0x0000040C addi $t1, $s1, 100
0x00000410 sw $s2, 32($t1)
0x00000414 sub $s2, $s0, $s2
```

quando l'istruzione di add si trova in fase di WB. **Sottolineare quali linee trasportano segnali utili.**

5. [7] Identificare le dipendenze e gli hazard su questo segmento di codice.

```
0x00000400 add $s5, $t2, $t1
0x00000404 addi $s1, $s5, 32
0x00000408 and $t4, $s6, $s1
0x0000040C sw $s5, 32($s0)
0x00000410 lw $s5, 32($s1)
0x00000414 sw $s5, 32($s2)
0x00000418 add $t4, $s6, $s1
0x0000041C beq $t4, $t5, 24
```

E modificare la CPU N.3, dove è possibile, per evitare stalli e gestire correttamente gli hazard.

6. [6] Cosa sono gli interrupt e le eccezioni? Come vengono gestiti dalle architetture Intel e dalle architetture MIPS/ARM? Specificare gli elementi della CPU MIPS che sono dedicati alla gestione delle eccezioni e cosa contengono. Modificare la CPU in Figura N.3 per potere gestire un'eccezione di "Overflow". Cosa si intende per mascheramento degli interrupt? Viene praticato nei MIPS? Come vengono gestite le eccezioni e gli interrupt dai sistemi operativi sul MIPS? Scrivere uno scheletro di funzione assembler per gestire un'eccezione.

7. [4] Riscrivere il codice seguente in modo che sia eseguito nel minor tempo possibile su una pipeline con 4 cammini di esecuzione di cui 3 general purpose (in grado di eseguire tutte le istruzioni) e 1 dedicato solo alle istruzioni di memoria. Qual è lo speed-up? (rapporto tra le prestazioni prima della modifica, sul codice sequenziale, e dopo la modifica). Applicare lo srotolamento dei cicli per un numero massimo di 8 iterazioni del ciclo. Si supponga di avere a disposizione un numero di registri interni di pipeline sufficientemente grande e che si possa applicare la ridenominazione dei registri.

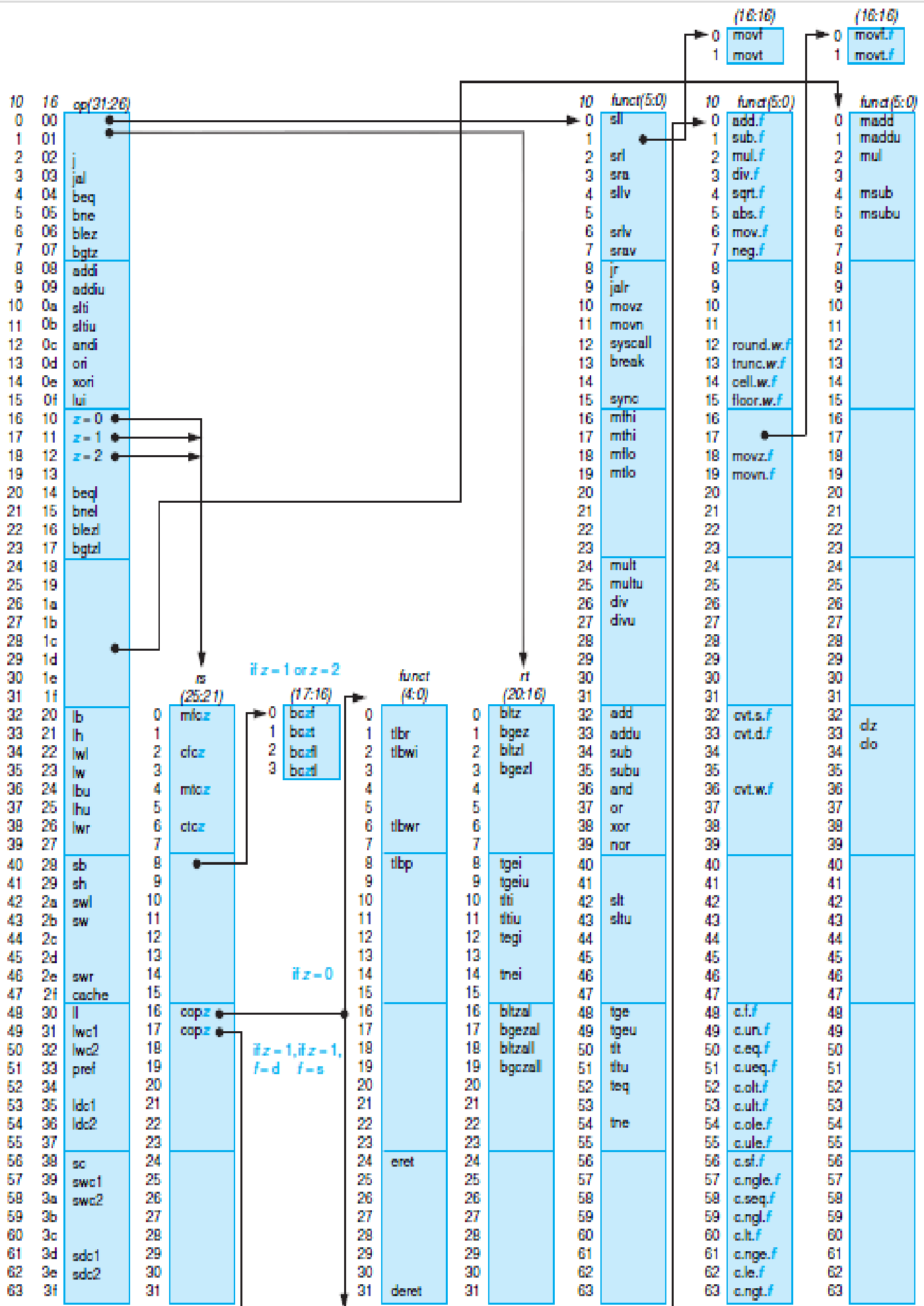
```
Ciclo:    beq  $s1, $zero, Esci
          lw   $t0, 0($s1)      # M[s1] -> t0
          sub  $t0, $t0, $s2    # t0 = t0 - s2
          sw   $t0, 0($s1)      # M[s1] <- t0
          addi $s1, $s1, -4     # next element
          j   Ciclo
Esci:     or   $s6, $s7, $s5
```

Il codice assembler corrisponde al seguente codice C:

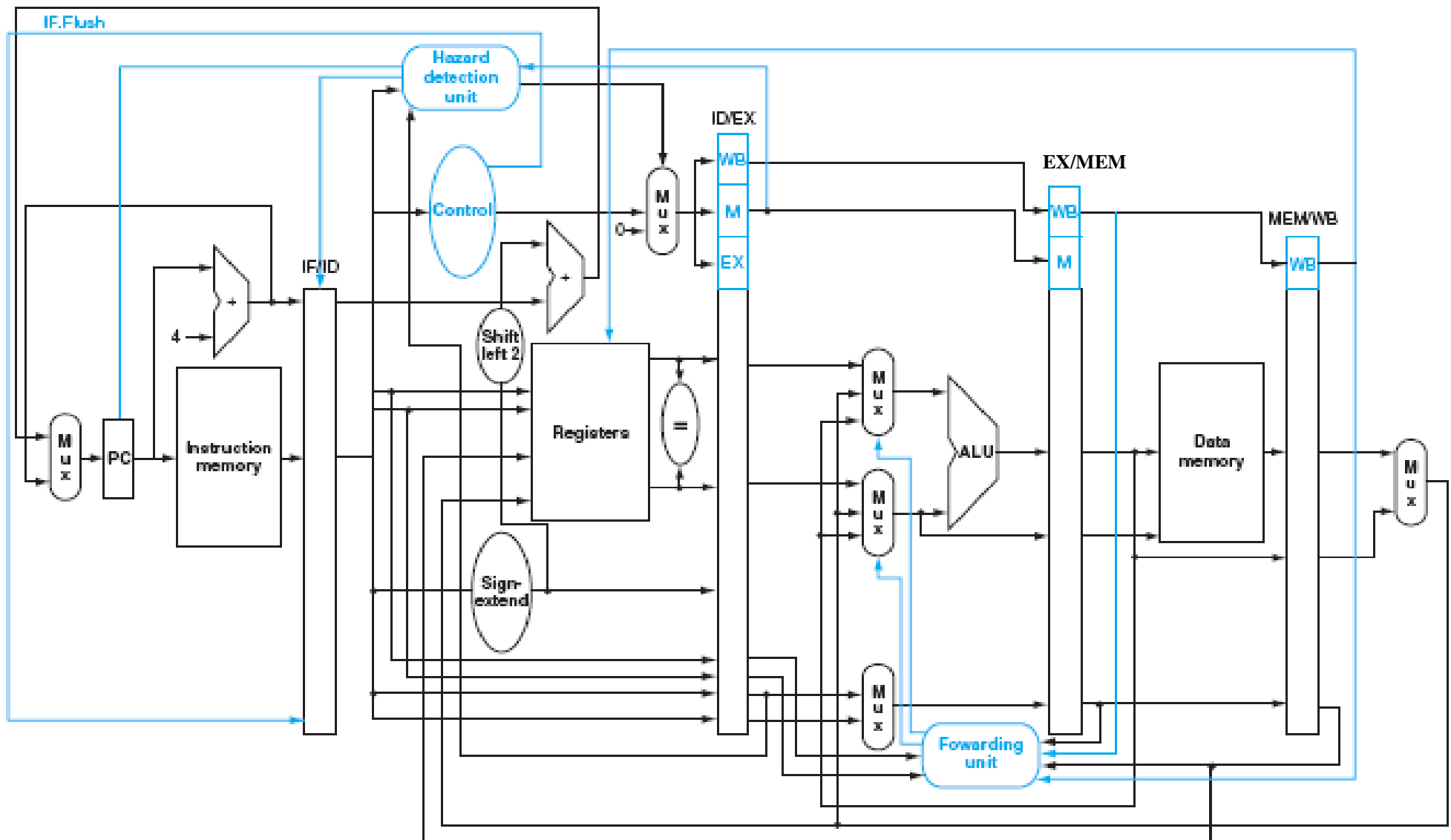
```
Ciclo:    if (s1 == 0) goto Esci
          v[s1] = t0 + v[s1];
          s1--;
          goto Ciclo;
Esci:
```

Registri del register file

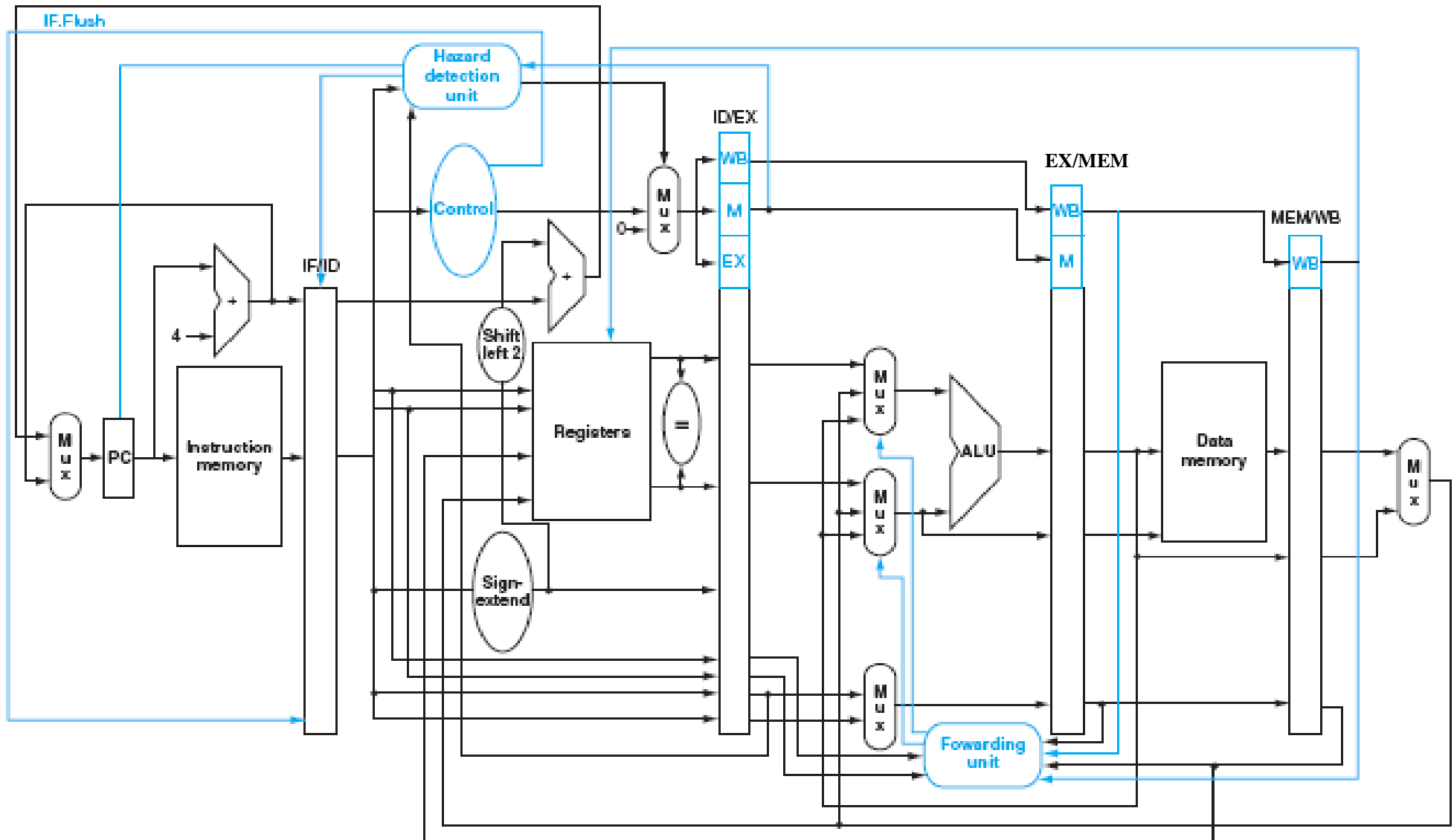
| | | | | | |
|-----|------|-------------------------|-----|-----|------------------------|
| 0 | zero | constant 0 | 16 | s0 | callee saves |
| 1 | at | reserved for assembler | ... | ... | (caller can clobber) |
| 2 | v0 | expression evaluation & | 23 | s7 | |
| 3 | v1 | function results | 24 | t8 | temporary (cont'd) |
| 4 | a0 | arguments | 25 | t9 | |
| 5 | a1 | | 26 | k0 | reserved for OS kernel |
| 6 | a2 | | 27 | k1 | |
| 7 | a3 | | 28 | gp | Pointer to global area |
| 8 | t0 | temporary: caller saves | 29 | sp | Stack pointer |
| ... | ... | (callee can clobber) | 30 | fp | frame pointer (s8) |
| 15 | t7 | | 31 | ra | Return Address (HW) |



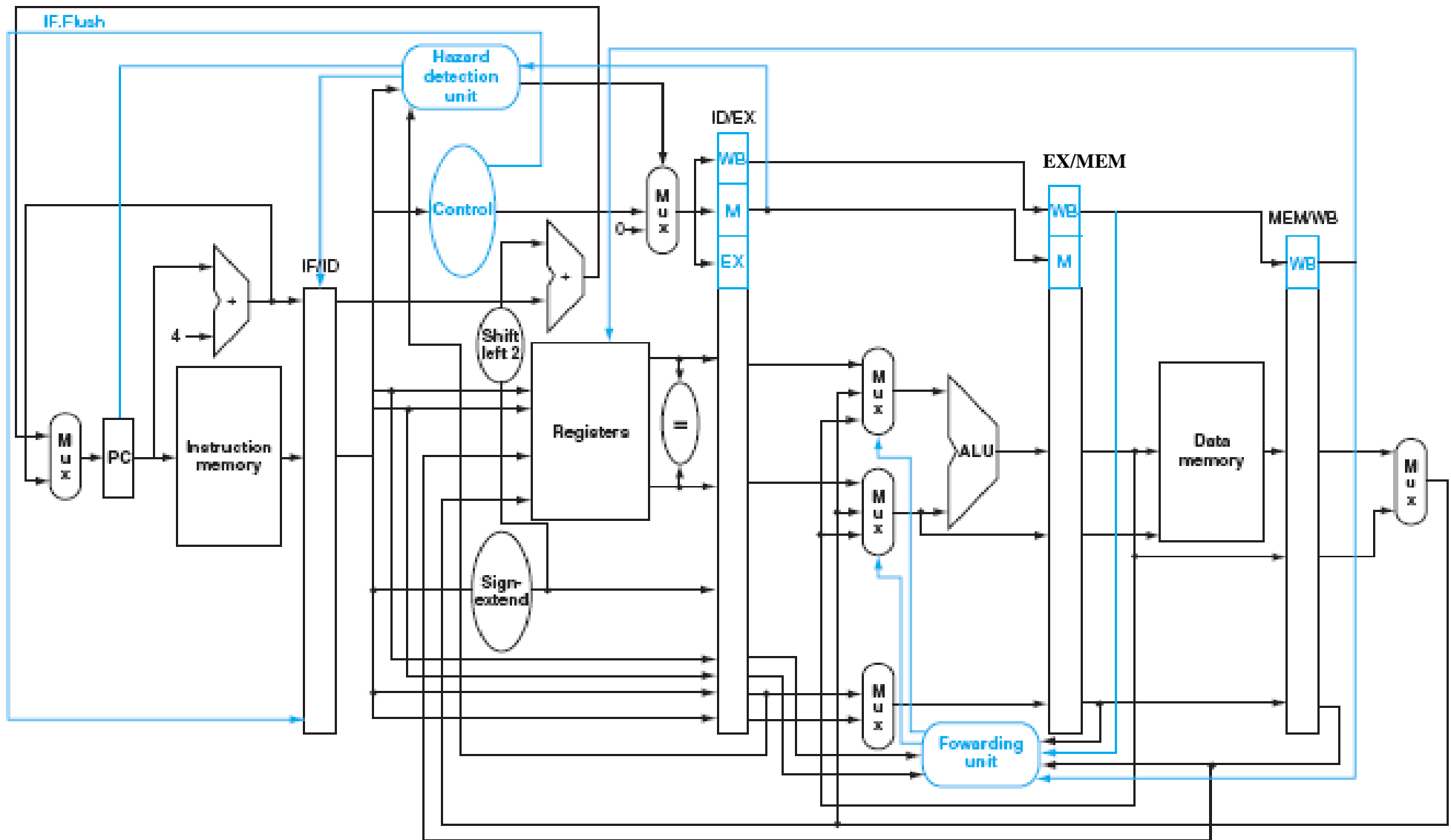
CPU N. 1



CPU N. 2



CPU N. 3



CPU N. 4

