Spiegazione Appello Giugno 2025

Architettura Degli Elaboratori

Zhewei Zhang

Appello Giugno 2025

Esercizio 1

Esercizio 1 (11 punti).

Sia data una CPU con processore a **8GHz** e **16 CPI** (Clock per Instruction) che adoperi indirizzi da 32 bit e memoria strutturata su due livelli di **cache** (L1, L2), il cui setup è come segue:
L1 è una cache **set-associativa** a **2 vie** con **4 set** e **blocchi da 32 word**; adopera una politica di rimpiazzo **LRU**. Ricordiamo che consideriamo la linea come l'insieme del blocco in cache con tag e bit di validità, mentre il set è il gruppo di linee con il medesimo indice.
L2 è una cache **direct-mapped** con **2 linee** e **blocchi da 256 word**.

1) Supponendo che all'inizio nessuno dei dati sia in cache, indicare quali degli accessi in memoria indicati di seguito sono HIT o MISS in ciascuna delle due cache. Per ciascuna **MISS** indicare se sia di tipo Cold Start (**Cold**), Capacità (**Cap**) o Conflitto (**Conf**). Utilizzare la tabella sottostante per fornire i risultati e indicare la metodologia di calcolo.

	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												
L2	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

2) Calcolare le dimensioni in bit (compresi i bit di controllo ed assumendo che ne basti uno per la LRU, e ignorando il bit dirty) delle due cache: (a) L1 e (b) L2.
3) Assumendo che gli accessi in **memoria** impieghino **400 ns**, che gli **hit** nella cache **L1** impieghino **10 ns** e gli **hit** nella cache **L2** impieghino **20 ns**, calcolare (a) il **tempo totale** per la sequenza di accessi, (b) il tempo **medio** per la sequenza di accessi, e (c) **quante istruzioni** vengono svolte nel tempo medio calcolato.
4) Calcolare il word offset del sesto indirizzo per la cache L2 spiegando i calcoli effettuati.
5) Supponendo che gli indirizzi nella tabella siano virtuali e la memoria virtuale consti di **128 pagine** di **4KiB** ciascuna, indicarne i numeri di pagina virtuale.

Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
Page#												

Appello Giugno 2025

Esercizio 1.1

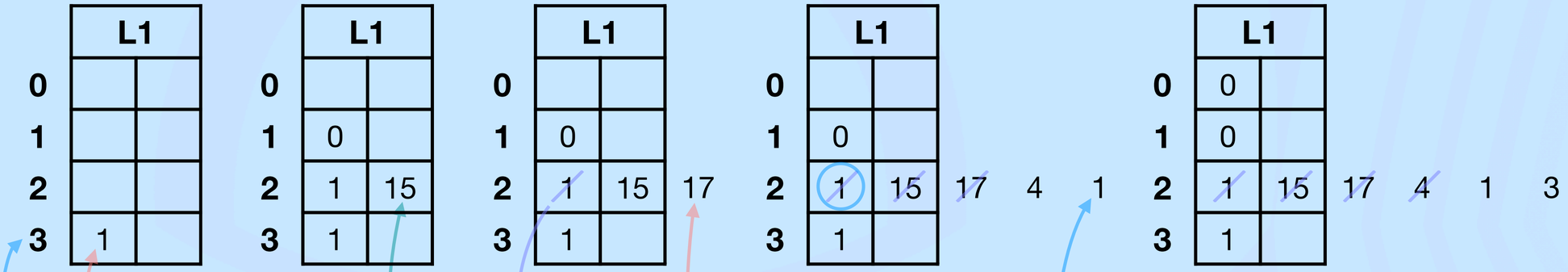
L1: cache **set-associativa** a **2 vie** con **4 set** e **blocchi da 32 word**

L1
Block# = **Address // dim. blocco in byte** (128 byte)
Index = **Block# % num. set** (4)
Tag = **Block# // num. set** (4)

	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#	7	1	6	62	62	70	18	6	0	1	14	6
	Index	3	1	2	2	2	2	2	2	0	1	2	2
	Tag	1	0	1	15	15	17	4	1	0	0	3	1
	HIT/MISS												
	Miss type												
L2	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

Appello Giugno 2025

Esercizio 1.1



HIT: quando già presente nella cache

MISS cold: quando non presente nella cache perché è il primo accesso

Tecnica di rimpiazzo LRU: quando un set della cache è pieno, viene rimosso il blocco meno recentemente usato

MISS conf/cap: quando non è MISS cold, bisogna distinguere tra conf e cap (slide successiva)

Inserimento in ordine

	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#	7	1	6	62	62	70	18	6	0	1	14	6
	Index	3	1	2	2	2	2	2	2	0	1	2	2
	Tag	1	0	1	15	15	17	4	1	0	0	3	1
	HIT/MISS	MISS	MISS	MISS	MISS	HIT	MISS	MISS	MISS	MISS	HIT	MISS	HIT
	Miss type	cold	cold	cold	cold	—	cold	cold	?	cold	—	cold	—
L2	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

Appello Giugno 2025

Esercizio 1.1

MISS conflict: quando un dato non può essere memorizzato nella cache perché il suo blocco nel set è già occupato da un altro indirizzo mappato nella stessa posizione

MISS capacity: quando la cache è troppo piccola per contenere tutti i dati necessari, causando l'eliminazione di informazioni ancora utili

Per distinguere tra conf e cap si può utilizzare una cache **fully-associative** di dimensione **set × vie**

L1							
7	1	6	62	70	18		

Si può utilizzare direttamente il Block#

Nella cache fully-associative possono verificarsi sono capacity miss, perché non ci sono conflitti tra set. Se un accesso è hit in fully-associative ma miss in set-associative, allora quel miss è necessariamente un conflict miss.

Inserimento in ordine

	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#	7	1	6	62	62	70	18	6	0	1	14	6
	Index	3	1	2	2	2	2	2	2	0	1	2	2
	Tag	1	0	1	15	15	17	4	1	0	0	3	1
	HIT/MISS	MISS	MISS	MISS	MISS	HIT	MISS	MISS	MISS	MISS	HIT	MISS	HIT
	Miss type	cold	cold	cold	cold	—	cold	cold	conf	cold	—	cold	—
L2	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

Appello Giugno 2025

Esercizio 1.1

L2: cache **direct-mapped** con **2 linee** e **blocchi da 256 word**

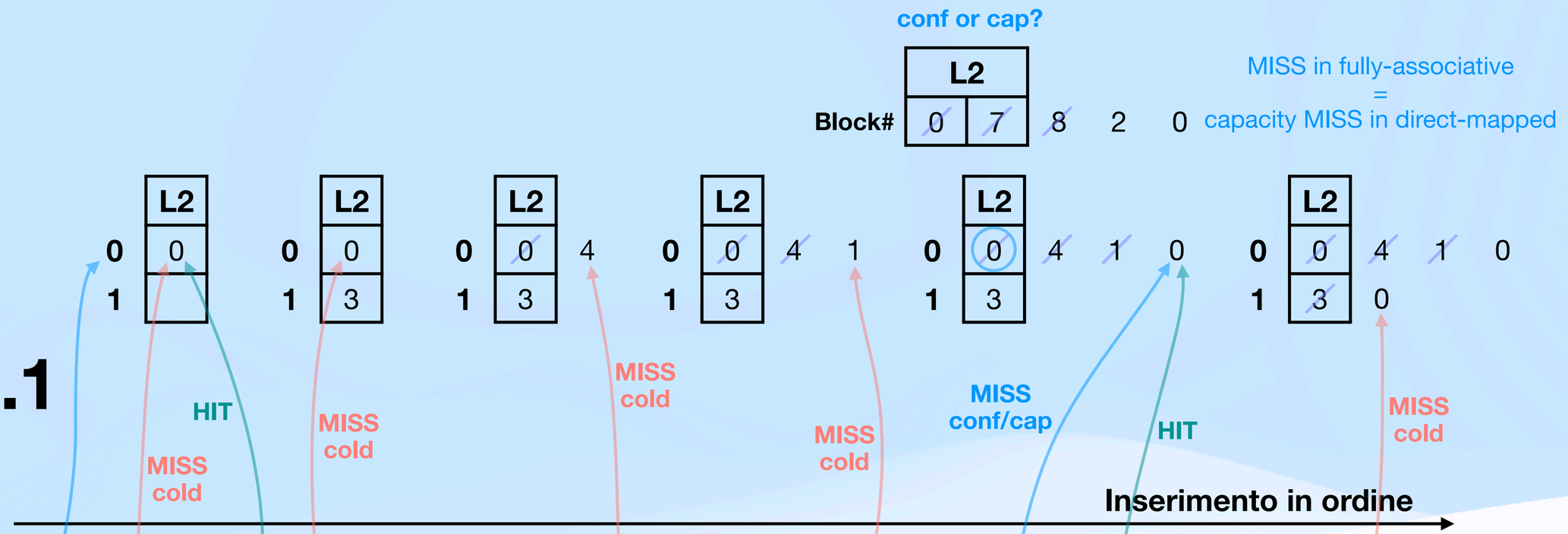
L2
Block# = **Address // dim. blocco in byte** (1024 byte)
Index = **Block# % num. linee** (2)
Tag = **Block# // num. linee** (2)

Se un dato viene trovato in L1 (hit), la cache L2 viene ignorata

	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#	7	1	6	62	62	70	18	6	0	1	14	6
	Index	3	1	2	2	2	2	2	2	0	1	2	2
	Tag	1	0	1	15	15	17	4	1	0	0	3	1
	HIT/MISS	MISS	MISS	MISS	MISS	HIT	MISS	MISS	MISS	MISS	HIT	MISS	HIT
	Miss type	cold	cold	cold	cold	—	cold	cold	conf	cold	—	cold	—
L2	Block#	0	0	0	7		8	2	0	0		1	
	Index	0	0	0	1		0	0	0	0		1	
	Tag	0	0	0	3		4	1	0	0		0	
	HIT/MISS												
	Miss type												

Appello Giugno 2025

Esercizio 1.1



	Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
L1	Block#	7	1	6	62	62	70	18	6	0	1	14	6
	Index	3	1	2	2	2	2	2	2	0	1	2	2
	Tag	1	0	1	15	15	17	4	1	0	0	3	1
	HIT/MISS	MISS	MISS	MISS	MISS	HIT	MISS	MISS	MISS	MISS	HIT	MISS	HIT
	Miss type	cold	cold	cold	cold	—	cold	cold	conf	cold	—	cold	—
L2	Block#	0	0	0	7		8	2	0	0		1	
	Index	0	0	0	1		0	0	0	0		1	
	Tag	0	0	0	3		4	1	0	0		0	
	HIT/MISS	MISS	HIT	HIT	MISS		MISS	MISS	MISS	HIT		MISS	
	Miss type	cold	—	—	cold		cold	cold	cap	—		cold	

Appello Giugno 2025

Esercizio 1.2

Calcolare le **dimensioni in bit** (compresi i bit di controllo ed assumendo che ne basti uno per la LRU, e ignorando il bit dirty) delle due cache: **(a) L1** e **(b) L2**

L1: cache **set-associativa** a **2 vie** con **4 set** e **blocchi da 32 word**

Dimensione di L1 (in bit)

(	Used	+	Valid	+	Tag	+	Blocco	)	×	Vie	×	Set		
(	1	+	1	+	23	+	1024	)	×	2	×	4	=	**8392**

Bit Used viene usato in set-associativa per tenere traccia degli accessi recenti in un blocco

Bit Valid indica se un blocco della cache contiene dati attualmente validi o meno

32 - 5 - 2 - 2

Offset word $\log_2 32$ (32 word)

Offset byte $\log_2 4$ (4 byte in 1 word)

Index $\log_2 4$ (4 set)

32 × 4 × 8 (word to bit)

L2: cache **direct-mapped** con **2 linee** e **blocchi da 256 word**

Dimensione di L2 (in bit)

(	Valid	+	Tag	+	Blocco	)	×	Linee		
(	1	+	21	+	8192	)	×	2	=	**16428**

Bit Valid

32 - 8 - 2 - 1

Offset word $\log_2 256$ (256 word)

Offset byte $\log_2 4$ (4 byte in 1 word)

Index $\log_2 2$ (2 linee)

256 × 4 × 8 (word to bit)

Appello Giugno 2025

Esercizio 1.3
Esercizio 1.4
Esercizio 1.5

Esercizio 1.3

Assumendo che gli accessi in **memoria** impieghino **400 ns**, che gli **hit** nella cache **L1** impieghino **10 ns** e gli **hit** nella cache **L2** impieghino **20 ns**, calcolare:

(a) Tempo totale per la sequenza di accessi

Miss in L2	×	Tempo accesso memoria	+	Hit in L1	×	Tempo hit in L1	+	Hit in L2	×	Tempo hit in L2		
6	×	400	+	3	×	10	+	3	×	20	=	**2490**

(b) Tempo medio per la sequenza di accessi = tempo totale / num. accessi = 2490 / 12 = **207.5**
(num. indirizzi nella tabella di es.1)

(c) Istruzioni eseguite nel tempo medio = (207.5 / 16) × 8 = **103.75**
[(tempo medio / CPI) × periodo di clock (GHz)]
(Informazioni descritte nel testo)

Esercizio 1.4

Calcolare il **word offset** del **sesto indirizzo** per la **cache L2** spiegando i calcoli effettuati:

Word offset di 9008 per la cache L2 = (Address % dim. blocco in byte) / 4 = (9008 % 1024) / 4 = **204**

Esercizio 1.5

Supponendo che gli indirizzi nella tabella siano virtuali e la memoria virtuale consti di **128 pagine** di **4KiB** ciascuna, indicarne i numeri di pagina virtuale:

Page# = **Address // dim. pagina** (4KiB = 4096 byte)

Address	1000	250	820	8000	8020	9008	2350	820	112	252	1913	770
Page#	0	0	0	1	1	2	0	0	0	0	0	0

Appello Giugno 2025

Esercizio 2

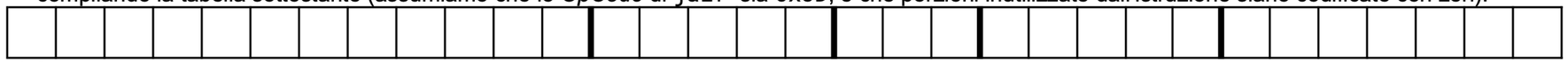
Esercizio 2 (11 punti).

Considerare l'architettura RISC-V a ciclo singolo nella figura in basso (e in allegato).
Si vuole aggiungere alla CPU l'istruzione **jump and link if divisible by 4** (`jalf`), di **tipo I** e sintassi assembly:
`jalf` **`rd, rs1, immediate`**
L'istruzione deve operare come segue:
a) carica dal banco registri `rs1`;
b) estende di segno il campo immediato `immediate` (si può usare Genera cost per questa operazione). Chiameremo questa word $\texttt{immediate}_{32}$;
c) se $\texttt{immediate}_{32}$ è divisibile per 4, salva il valore di PC + 4 nel registro `rd`;
d) se $\texttt{immediate}_{32}$ è divisibile per 4, aggiorna il PC con il valore $\texttt{rs1 + immediate}_{32}$.

Esempio: Supponiamo che **`rd`** sia `a0`, che **`rs1`** sia `t4` contenente il valore 3, e che **`immediate`** valga `0xFC = 0b000011111100 = 252`
In tal caso:

- `0xFC` è divisibile per 4, quindi il valore del PC + 4 verrà salvato in a0.
- `0xFC` è divisibile per 4, quindi la parte immediata verrà estesa di segno (rimanendo 252 in questo caso), e sommata a t4.
- Tale valore verrà infine scritto nel PC, che conterrà 252+3 = 255 alla fine dell'istruzione.

1) Mostrare le **modifiche all'architettura** della CPU RISC-V, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.
2) Indicare il contenuto in bit della word che esprime l'istruzione
   `jalf` **`t4, s4, 0x203`**
   compilando la tabella sottostante (assumiamo che lo *OpCode* di `jalf` sia `0x3D`, e che porzioni inutilizzate dall'istruzione siano codificate con zeri).



3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 150 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati*.
4) Indicando con `jalf` il segnale di controllo che viene asserito per eseguire la nuova istruzione, assumiamo che
   a) tutti i segnali di tipo *don't care* siano pari a `0` e che
   b) la Control Unit della CPU RISC-V modificata per supportare `jalf` sia difettosa e sovrascriva il segnale `RegWrite` come segue:
   `RegWrite` ≔ `MemWrite` (il simbolo ≔ denota che la variabile a sinistra assume il valore dato della variabile a destra)
   In tal caso, indicare quale valore sia assegnato a `a7` al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `a7`, `s0` e `s1` siano inizializzati a `0`. Motivare la propria risposta.

```
      addi s0, s0, 0x4
      sw   t0, 0(s0)
      beq  s1, s0, Out
      jal  x0, Exit
Out:  add  a7, zero, s1
Exit: addi a7, a7, 0x8002
```

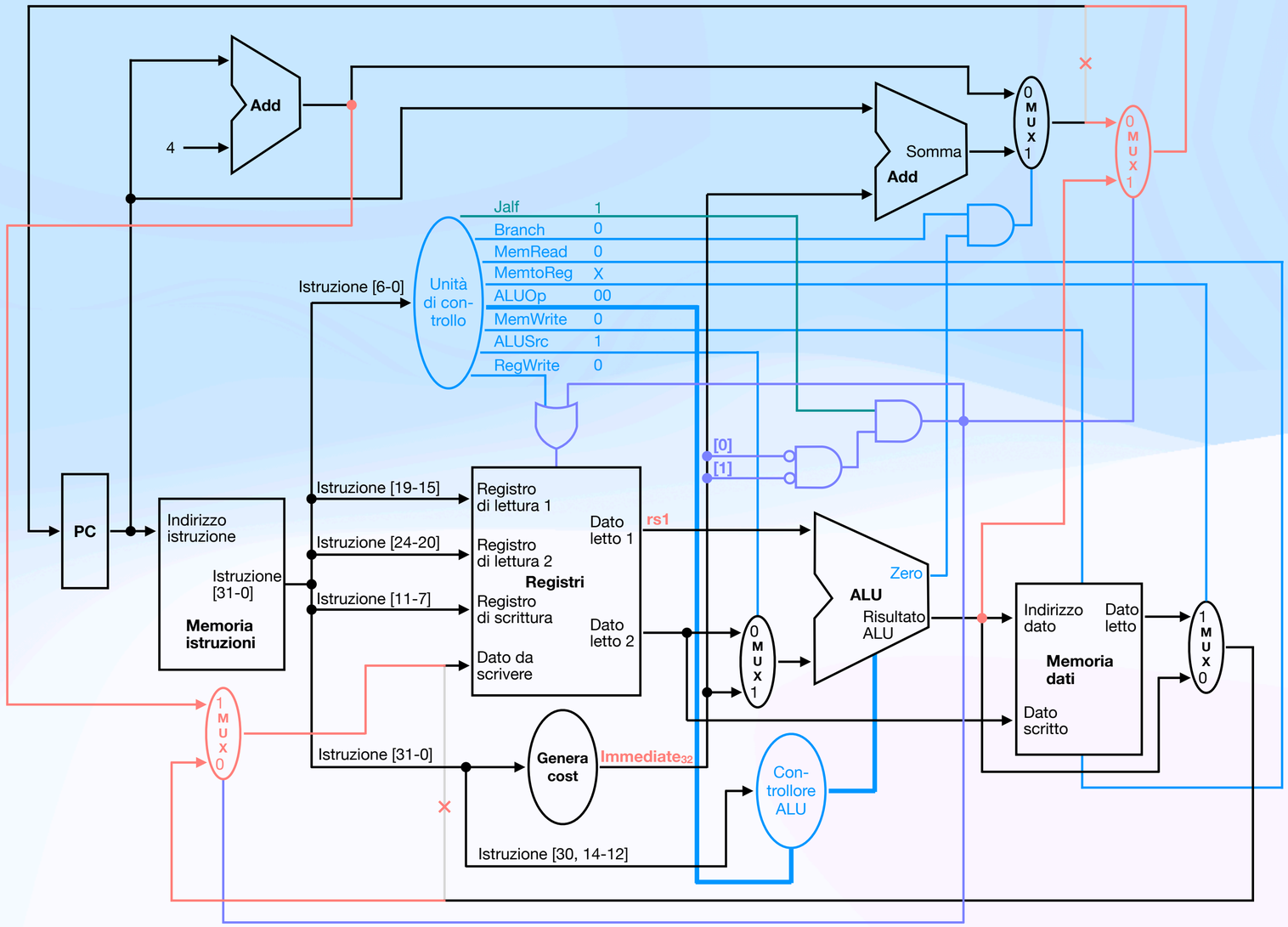
Appello Giugno 2025

Esercizio 2.1

Si vuole aggiungere alla CPU l'istruzione **jump and link if divisible by 4** (jalf), di **tipo I** e sintassi assembly:

jalf **rd**, **rs1**, **immediate**

L'istruzione deve operare come segue:
a) carica dal banco registri **rs1**;
b) estende di segno il campo immediato immediate (si può usare **Genera cost** per questa operazione). Chiameremo questa word **immediate$_{32}$**;
c) se **immediate$_{32}$** è divisibile per 4, salva il valore di **PC + 4** nel registro **rd**;
d) se **immediate$_{32}$** è divisibile per 4, aggiorna il PC con il valore **rs1** + **immediate$_{32}$**.



Appello Giugno 2025

Esercizio 2.2
Esercizio 2.3

Esercizio 2.2

Indicare il contenuto in bit della word che esprime l'istruzione **jalf t4, s4, 0x203** compilando la tabella sottostante (assumiamo che lo *OpCode* di jalf sia 0x3D, e che porzioni inutilizzate dall'istruzione siano codificate con zeri):

Esercizio 2.3

Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU** e dei **sommatori 150 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa spiegando i calcoli effettuati:

	Memoria	+	Registro	+	ALU	+	Registro		
jalf	200 Insieme al sommatore PC+4	+	50	+	150	+	50	=	**450 ns**

	Memoria	+	Registro	+	ALU	+	Memoria	+	Registro		
lw	200 Insieme al sommatore PC+4	+	50	+	150	+	200	+	50	=	**650 ns**

Il clock è progettato per l'istruzione più lenta (**lw**), che in questo caso impiega **650 ns**, mentre **jalf** richiede solo **450 ns**, poiché **jalf** è più veloce, non necessita di un periodo di clock più lungo.

Appello Giugno 2025

Esercizio 2.4

Indicando con **jalf** il segnale di controllo che viene asserito per eseguire la nuova istruzione, assumiamo che
a) tutti i segnali di tipo ***don't care*** siano pari a **0** e che
b) la **Control Unit** della CPU RISC-V **modificata** per supportare **jalf** sia **difettosa** e sovrascriva il segnale **RegWrite** come segue: **RegWrite** ≔ **MemWrite** (il simbolo ≔ denota che la variabile a sinistra assume il valore dato della variabile a destra)

In tal caso, indicare quale valore sia assegnato a **a7** al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri **a7**, **s0** e **s1** siano inizializzati a **0**. Motivare la propria risposta.

```
addi s0, s0, 0x4
sw t0, 0(s0)
beq s1, s0, Out
jal x0, Exit
Out: add a7, zero, s1
Exit: addi a7, a7, 0x8002
```

```
addi s0, s0, 0x4
```
→ il **MemWrite** in **addi** è **0**, perciò anche **RegWrite** è **0** e quindi questa **addi** non scrive su **s0** (in **s0** rimane **0**).

```
sw t0, 0(s0)
```
→ il **MemWrite** in **sw** è **1**, perciò anche **RegWrite** è **1**, dopo aver scritto **t0** in memoria, verrà scritto il risultato di **ALU** in **x0** (i primi 5 LSB di **immediate**), cioè non scrive niente perché **x0** è sempre **0**.

```
beq s1, s0, Out.
```
→ **s1** e **s0** sono entrambi **0** quindi si salta a **Out**.

```
jal x0, Exit
```
→ saltata.

```
Out: add a7, zero, s1
```
→ non scrive in **a7** per stesso motivo della prima istruzione.

```
Exit: addi a7, a7, 0x8002
```
→ rileva un **eccezione** perché la parte immediata supera i **12 bit**.

Appello Giugno 2025

Esercizio 3

Esercizio 3 (11 punti)

Si consideri l'architettura RISC-V con pipeline. Il programma qui di seguito effettua la somma dei soli valori dispari (dopo averli moltiplicati per due e aver sottratto 1) degli elementi presenti nell'array vettore di lunghezza 10. Infine, viene stampato il valore di tale somma.

```
1   .data
2   vettore: .word 24, 1, 46, 54, 50, 12, 2, 11, 39, 4
3    # 7 pari, 3 dispari
4
5   .text                       # Sommo solo i dispari moltipl. per 2
6   main:   addi t0, zero, 0    # somma parziale
7           la   s0, vettore    # indirizzo base
8           addi t4, s0, 40     # primo indirizzo fuori dall'array
9   ciclo:  lw   t3, 0(s0)
10          andi t2, t3, 1      # è dispari?
11          beq  t2, zero, salta  # se non lo è, salto
12          slli t3, t3, 1      # altrimenti moltiplico per 2 il numero
13          addi t3, t3, -1     # e tolgo uno
14          add  t0, t0, t3     # e lo accumulo
15  salta:  addi s0, s0, 4      # prossimo elemento
16          blt  s0, t4, ciclo  # fine del ciclo?
17
18          addi a7,x0, 1       # stampa…
19          mv a0, t0           # … la somma
20          ecall
```

Si supponga che non si faccia uso di alcuna pseudoistruzione, e che le ecall non richiedano alcuno stallo. Si consideri che la decisione di branch venga presa in fase di ID, e che le unità di forwarding (quando usate) includano il forwarding da EXE a EXE, da MEM a EXE, e (per il branch) da EXE a ID.
Si indichino (ignorando hazard che possano concernere la ecall):

1) Per i primi 24 cicli di clock (**senza unità di forwarding**), le istruzioni tra le quali sono presenti **data hazard** – indicare i numeri di linea N ed M (visibili alla sinistra delle linee di codice in figura) ed il registro coinvolto xY (nel formato N / M / xY, ad esempio 17 / 18 / t0 se c'è un data hazard causato dal registro t0 tra l'istruzione alla linea 17 e quella alla linea 18);

2) Per i primi 15 cicli di clock (**senza unità di forwarding**) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di linea N ed M (nel formato N / M, ad esempio se l'istruzione alla linea 17 causa un control hazard con l'istruzione alla linea 23 scrivere: 17 / 23);

3) quanti **cicli di clock** sono necessari ad eseguire il programma tramite **forwarding**, spiegando il calcolo effettuato;

4) quanti **cicli di clock** sarebbero necessari ad eseguire il programma **senza forwarding**, spiegando il calcolo effettuato;

5) quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **15° ciclo di clock** (**con forwarding**);
   IF:
   ID:
   EX:
   MEM:
   WB:

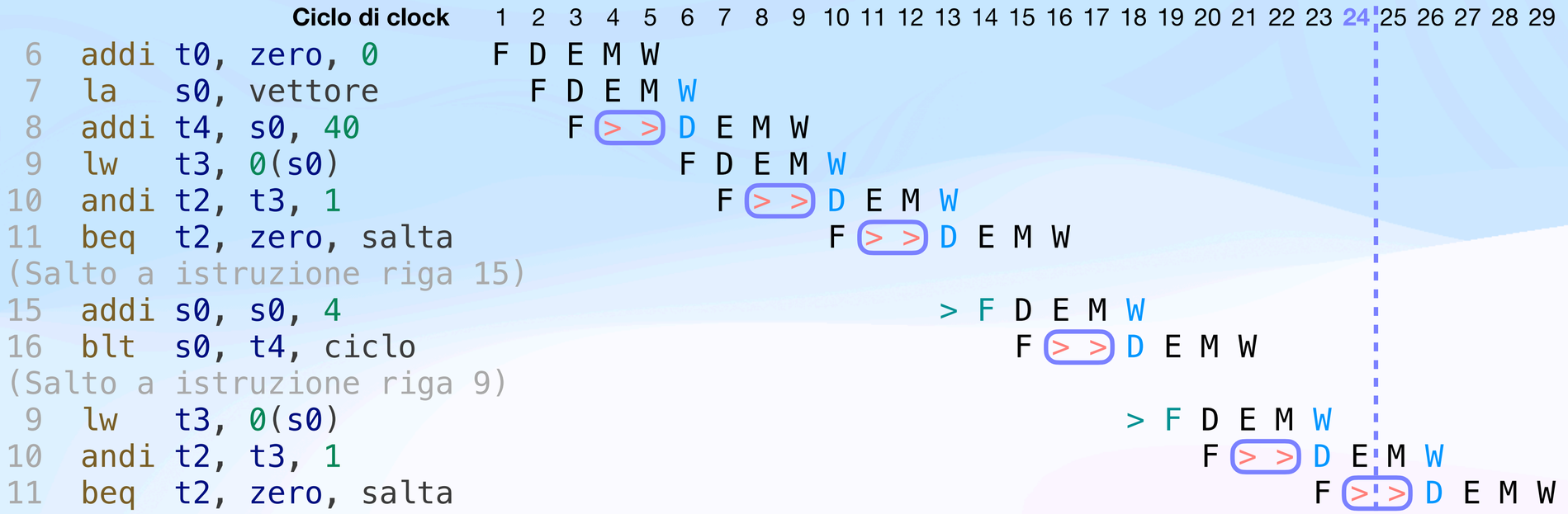
6) Spiegare brevemente il concetto di delay slot o salto ritardato, nel contesto della predizione dei salti.

Appello Giugno 2025

Esercizio 3.1

Si indichino per i primi **24** cicli di clock (**senza unità di forwarding**), le istruzioni tra le quali sono presenti **data hazard** – indicare i numeri di linea **N** ed **M** (visibili alla sinistra delle linee di codice in figura) ed il registro coinvolto **xY** (nel formato **N / M / xY**, ad esempio 17 / 18 / t0 se c'è un data hazard causato dal registro t0 tra l'istruzione alla linea 17 e quella alla linea 18)

> stallo per data hazard
> stallo per control hazard
D, W data hazard risolto con stalli



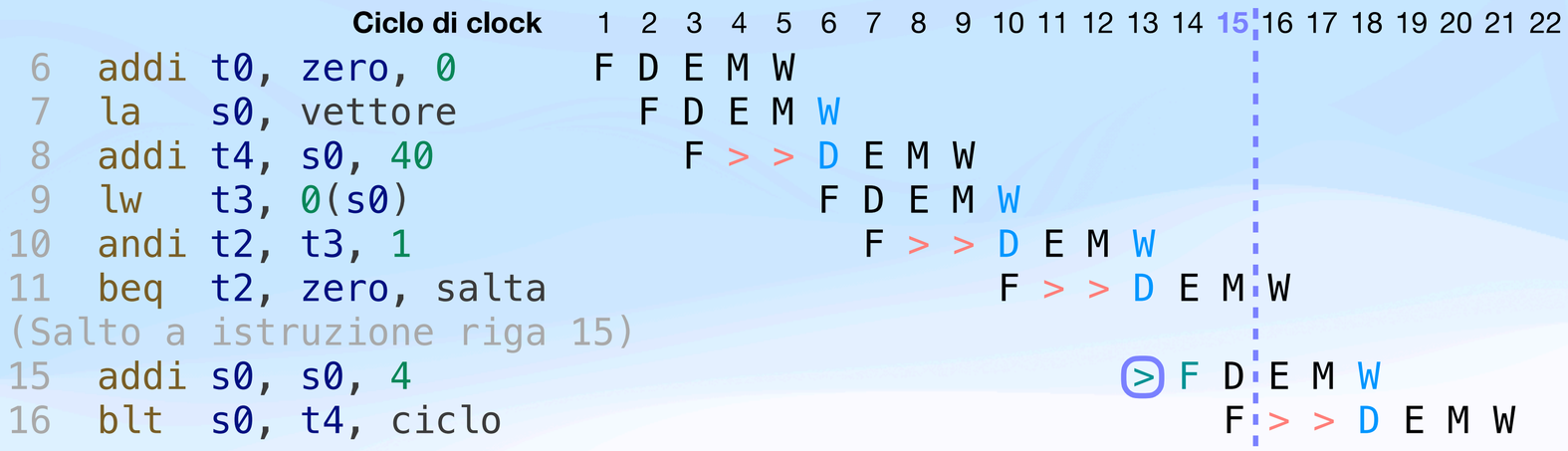
7 / 8 / s0
9 / 10 / t3
10 / 11 / t2
15 / 16 / s0
7 / 8 / s0 (duplicata)
9 / 10 / t3 (duplicata)

Appello Giugno 2025

Esercizio 3.2

Si indichino per i primi **15** cicli di clock (**senza unità di forwarding**) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di linea **N** ed **M** (nel formato **N / M**, ad esempio se l'istruzione alla linea 17 causa un control hazard con l'istruzione alla linea 23 scrivere: 17 / 23)

> stallo per data hazard
> stallo per control hazard
D, W data hazard risolto con stalli



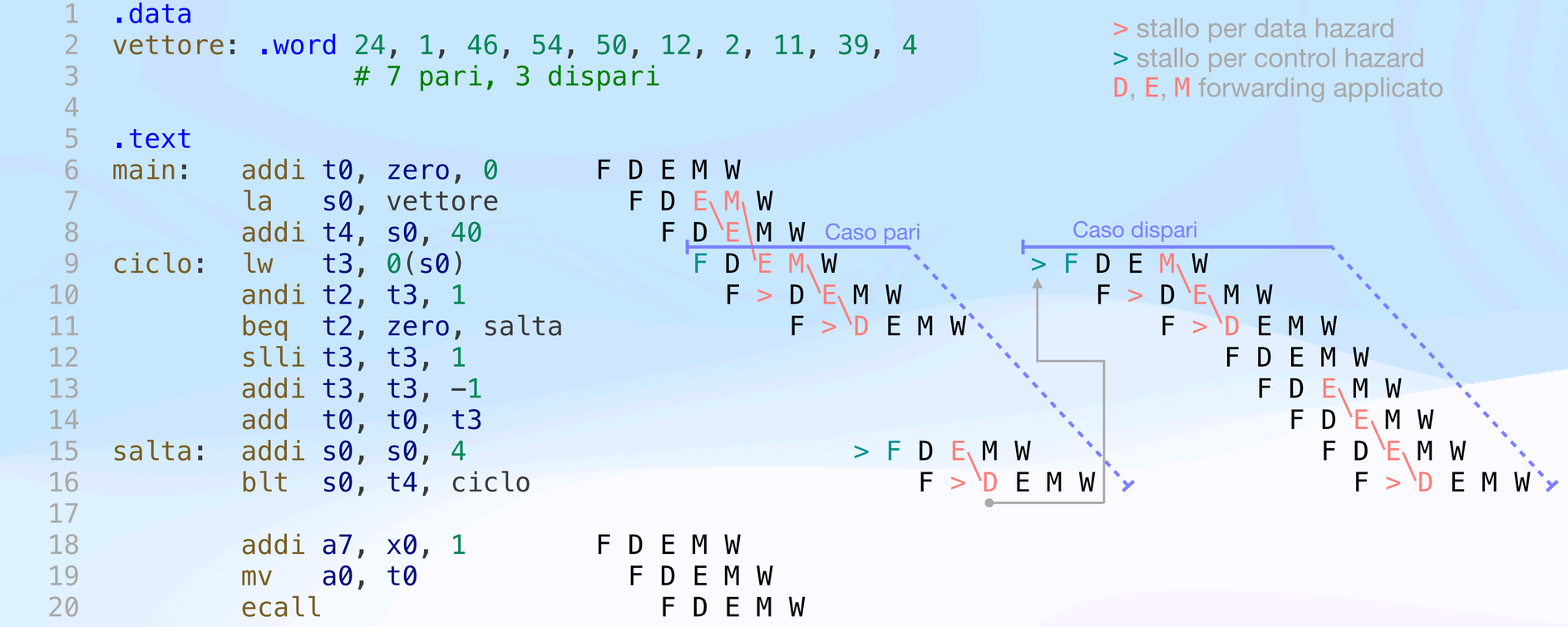
11 / 15

Appello Giugno 2025

Esercizio 3.3

Si indichino quanti **cicli di clock** sono necessari ad eseguire il programma tramite **forwarding**, spiegando il calcolo effettuato

```
1   .data
2   vettore: .word 24, 1, 46, 54, 50, 12, 2, 11, 39, 4
3                  # 7 pari, 3 dispari
4
5   .text
6   main:   addi t0, zero, 0
7           la   s0, vettore
8           addi t4, s0, 40
9   ciclo:  lw   t3, 0(s0)
10          andi t2, t3, 1
11          beq  t2, zero, salta
12          slli t3, t3, 1
13          addi t3, t3, -1
14          add  t0, t0, t3
15  salta:  addi s0, s0, 4
16          blt  s0, t4, ciclo
17
18          addi a7, x0, 1
19          mv   a0, t0
20          ecall
```



Cicli di clock con forwarding

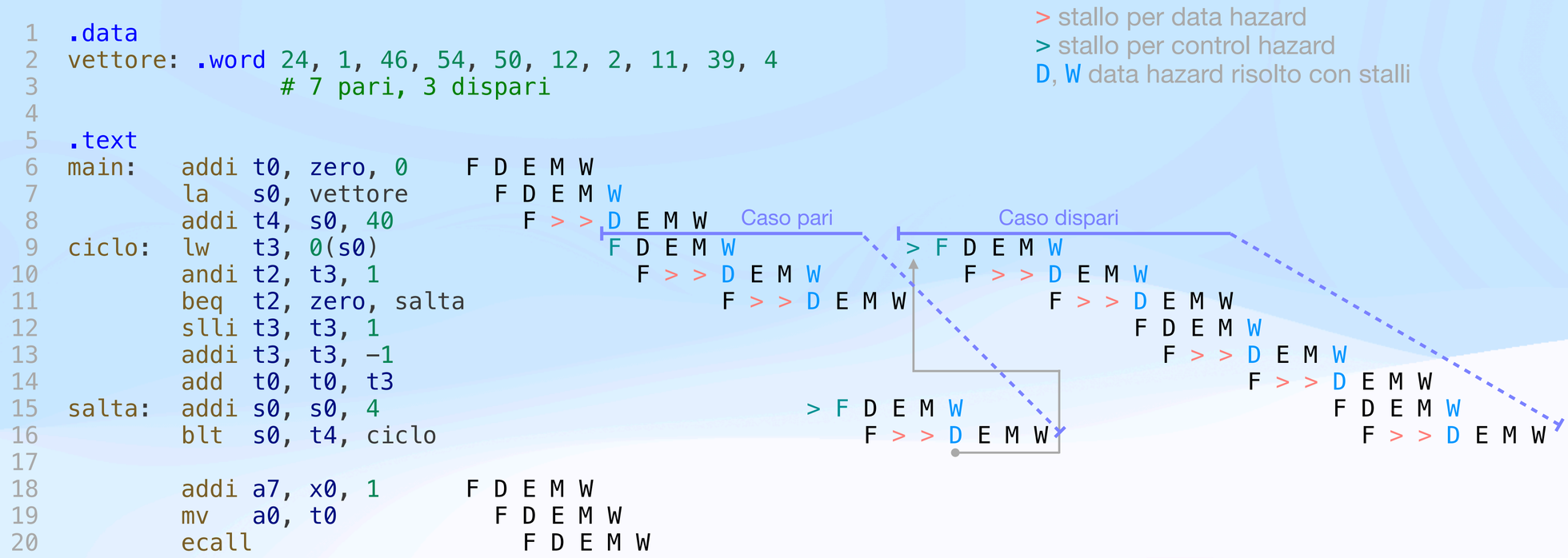
Riempimento pipeline	+	Istruzioni pre-ciclo	-	Entrata ciclo (1 control hazard in meno)	+	Caso pari	×	(	Istruzioni nel ciclo	+	Stalli per data hazard	+	Stalli per control hazard	)	+	Caso dispari	×	(	Istruzioni nel ciclo	+	Stalli per data hazard	+	Stalli per control hazard	)	+	Istruzioni post-ciclo		
4	+	3	-	1	+	7	×	(	5	+	3	+	2	)	+	3	×	(	8	+	3	+	1	)	+	3	=	**115**

Appello Giugno 2025

Esercizio 3.4

Si indichino quanti **cicli di clock** sarebbero necessari ad eseguire il programma **senza forwarding**, spiegando il calcolo effettuato

```
1   .data
2   vettore: .word 24, 1, 46, 54, 50, 12, 2, 11, 39, 4
3                  # 7 pari, 3 dispari
4
5   .text
6   main:   addi t0, zero, 0
7           la   s0, vettore
8           addi t4, s0, 40
9   ciclo:  lw   t3, 0(s0)
10          andi t2, t3, 1
11          beq  t2, zero, salta
12          slli t3, t3, 1
13          addi t3, t3, -1
14          add  t0, t0, t3
15  salta:  addi s0, s0, 4
16          blt  s0, t4, ciclo
17
18          addi a7, x0, 1
19          mv   a0, t0
20          ecall
```



Cicli di clock senza forwarding

Riempimento pipeline	+	Istruzioni pre-ciclo	+	Stalli per data hazard pre-ciclo	-	Entrata ciclo (1 control hazard in meno)	+	Caso pari	×	(	Istruzioni nel ciclo	+	Stalli per data hazard	+	Stalli per control hazard	)	+	Caso dispari	×	(	Istruzioni nel ciclo	+	Stalli per data hazard	+	Stalli per control hazard	)	+	Istruzioni post-ciclo		
4	+	3	+	2	-	1	+	7	×	(	5	+	6	+	2	)	+	3	×	(	8	+	10	+	1	)	+	3	=	**159**

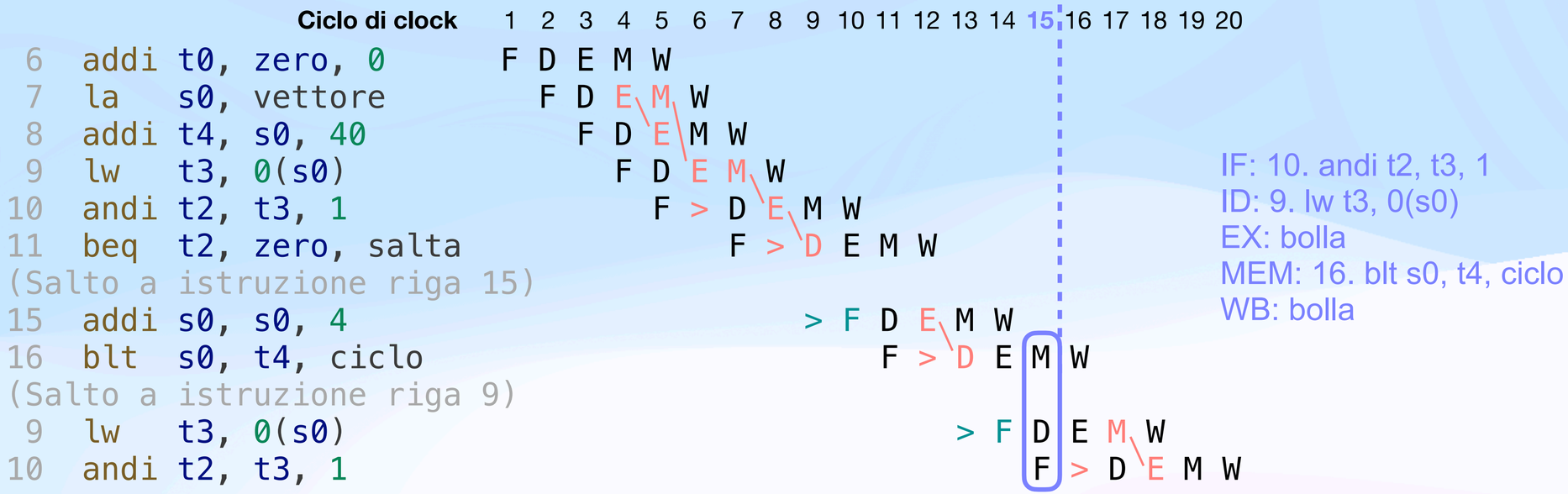
Appello Giugno 2025

Esercizio 3.5
Esercizio 3.6

Esercizio 3.5

Si indichino quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **15° ciclo di clock** (**con forwarding**)

> stallo per data hazard
> stallo per control hazard
D, E, M forwarding applicato



Esercizio 3.6

Spiegare brevemente il concetto di delay slot o salto ritardato, nel contesto della predizione dei salti.

Il delay slot o salto ritardato è una tecnica usata nelle architetture pipeline per ottimizzare l'esecuzione dei salti: l'istruzione immediatamente successiva al branch/jump viene sempre eseguita (prima che il salto abbia effetto), indipendentemente dall'esito del salto stesso.