Università degli Studi di Verona
A.A. 2017-2018

APPUNTI DI ARCHITETTURA DEGLI ELABORATORI

Creato da: Davide Zampieri

Indice:

MICROISTRUZIONI

Regole ed accorgimenti:
Bisogna tener presente che un solo valore per volta può passare nel bus; per sicurezza, ricordarsi sempre di controllare che non si stiano manipolando due valori differenti in un'unica microistruzione. Ad esempio, se devo utilizzare il valore del registro EAX e lo devo sommare al valore di EBX, sicuramente uno dei due deve essere salvato in un registro ausiliario della ALU (nell'architettura di riferimento si chiama V).

Metodi di indirizzamento:
- *%EAX* Indirizzamento diretto a registro: il valore viene prelevato dal registro;
- *(%EAX)* Indirizzamento indiretto a registro: il valore si trova nell'indirizzo di memoria puntato da EAX, quindi occorre decodificarlo inviando il valore del registro EAX nel registro MAR, aspettare la lettura in memoria (WMFC) e prelevare il valore ottenuto dal registro MDR;
- *4(%EAX)* Come per l'indirizzamento indiretto, con l'aggiunta che il valore ottenuto dal registro MDR deve essere sommato al valore prima della parentesi (in questo caso 4);
- *$4(%EAX)* Indirizzamento indiretto con spiazzamento: come per l'indirizzamento indiretto, con l'aggiunta che, prima di leggere in memoria, al valore del registro EAX va aggiunto lo spiazzamento che è codificato direttamente nell'istruzione e si ottiene dal registro OFFSET(IR).

Esercizi svolti con procedimento:
- *Fetch di un'istruzione*

Il contenuto del registro PC (ossia il puntatore all'istruzione che si andrà ad eseguire dopo il Fetch) viene incrementato di 4 Byte e inserito nel registro MAR che, dato il comando READ, permetterà di leggere il dato (in questo caso l'istruzione che si trova all'indirizzo puntato da PC). Nel frattempo, la somma PC+4 viene completata e il risultato viene rimesso in PC. In questo modo PC punta alla prossima istruzione. A questo punto, la memoria avrà restituito il dato richiesto nel registro MDR, il cui contenuto viene inserito nel registro IR delle istruzioni.

Fetch di un'istruzione
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}
2. WMFC, Z_{OUT}, PC_{IN}
3. MDR_{OUT}, IR_{IN}

- *INC dest*

Il dato contenuto in "dest" (che può essere un registro o un indirizzamento, in questo caso bisognerà leggere in memoria) viene mandato alla ALU selezionando 0 e portando sul bus il CB (bit di riporto) che permette di incrementare di 1 il dato, il quale viene poi salvato nel registro Z. Infine, il contenuto di Z viene reinserito in "dest" (direttamente nel registro o tramite una scrittura in memoria, ricordando che nel registro MAR è ancora presente l'indirizzo).

INC %EAX	**INC variabile**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}
2. WMFC, Z_{OUT}, PC_{IN}	2. WMFC, Z_{OUT}, PC_{IN}
3. MDR_{OUT}, IR_{IN}	3. MDR_{OUT}, IR_{IN}
4. $EAX_{out}, Select0, CB, ADD, Z_{in}$	4. $IR\_imm\_field_{out}, MAR_{in}, READ, WMFC$
5. Z_{out}, EAX_{in}, END	5. $MDR_{out}, Select0, CB, ADD, Z_{in}$
	6. $Z_{out}, MDR_{in}, WRITE, WMFC$
	7. END

- *MOV src, dest*

Il valore di "src" viene spostato in "dest", bisogna quindi prelevare il valore del primo registro "src" e caricarlo nel registro "dest".

- *ADD src, dest*

Viene sommato il valore di "src" a "dest" e il valore ottenuto viene caricato in "dest"; bisogna perciò prelevare il valore del primo registro "src" e caricarlo nel registro ausiliario della ALU, poi prelevare il valore di "dest" e direttamente sommarlo al valore, precedentemente caricato nel registro ausiliario, tramite il comando ADD (segnale di controllo della ALU), per poi salvarlo nel registro di uscita della ALU (chiamato Z). Infine, il valore in Z deve essere inviato in "dest".

MOVL (%EAX), %EBX	**ADDL (%EBX), %EAX**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. EAX_{OUT}, MAR_{IN}, READ 5. WMFC 6. MDR_{OUT}, EBX_{IN}, END	1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.EBX_{OUT}, MAR_{IN}, READ 5.WMFC, EAX_{OUT}, V_{IN} 6.MDR_{OUT}, SELECT[V], ADD, Z_{IN} 7.Z_{OUT}, EAX_{IN}, END
ADDL %EAX, %EBX	**ADD $4(%EAX + %EBX), %ECX**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. EAX_{OUT}, V_{IN} 5. EBX_{OUT}, $SELECT_V$, ADD, Z_{IN} 6. Z_{OUT}, EBX_{IN}, END	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. EAX_{OUT}, V_{IN} 5. EBX_{OUT}, $SELECT_V$, ADD, Z_{IN} 6. Z_{OUT}, V_{IN} 7. $OFFESET(IR)_{OUT}$, $SELECT_V$, ADD, Z_{IN} 8. Z_{OUT}, MAR_{IN}, READ 9. WMFC, ECX_{OUT}, V_{IN} 10. MDR_{OUT}, $SELECT_V$, ADD, Z_{IN} 11. Z_{OUT}, ECX_{IN}, END

- *Jxx value*

Deve essere eseguito un salto verso "value" (può essere un'etichetta, un valore di registro, ...) in base alla situazione descritta dopo J → JZ: Jump if Zero, JE: Jump if Equal, JNE: Jump if Not Equal, JL: Jump if Less, JLE: Jump if Less or Equal, JG: Jump if Great, JGE: Jump if Greater or Equal, JMP: salto incondizionato (si salta direttamente). Bisogna controllare se il valore ottenuto da una precedente CMP (compare) soddisfa la condizione; se non la soddisfa, la microistruzione finisce subito, altrimenti bisogna eseguire il salto. Deve perciò essere decodificato il valore "value" che può provenire da diversi luoghi, in base alla situazione (se è un'etichetta proviene dall'OFFSET, se è un registro proviene da esso, se è un indirizzamento bisognerà leggere dalla memoria). Se il salto è relativo, una volta ottenuto il valore bisogna sommarlo al valore di PC e dopo essere stato sommato, deve essere inserito nel PC. Se il salto è assoluto, il valore "value" deve essere inserito direttamente in PC (senza somma).

JZ (%EAX) con salto relativo	**JZ (%EAX) + %SI con salto relativo**
1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.if (!ZERO) END, EAX_{OUT}, MAR_{IN}, READ 5.WMFC, PC_{OUT}, V_{IN} 6.MDR_{OUT}, SELECT[V], ADD, Z_{IN} 7.Z_{OUT}, PC_{IN}, END	1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.if (!ZERO) END, EAX_{OUT}, MAR_{IN}, READ 5.WMFC, SI_{OUT}, V_{IN} 6.MDR_{OUT}, SELECT[V], ADD, Z_{IN} 7.Z_{OUT}, V_{IN} 8.PC_{OUT}, SELECT[V], ADD, Z_{IN} 9.Z_{OUT}, PC_{IN}, END
JNZ (%EAX) con salto assoluto	**JNZ (%EAX + %EBX) con salto assoluto**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. if (ZERO) END, EAX_{OUT}, MAR_{IN}, READ 5. WMFC 6. MDR_{OUT}, PC_{IN}, END	1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.if (ZERO) END, EAX_{OUT}, V_{IN} 5.EBX_{OUT}, SELECT[V], ADD, Z_{IN} 6.Z_{OUT}, MAR_{IN}, READ 7.WMFC 8.MDR_{OUT},PC_{IN}, END
JMP (%EAX) con salto relativo	**JMP (%EAX) con salto assoluto**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. EAX_{OUT}, MAR_{IN}, READ 5. WMFC, PC_{OUT}, V_{IN} 6. MDR_{OUT}, $SELECT_V$, ADD, Z_{IN} 7. Z_{OUT}, PC_{IN}, END	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. EAX_{OUT}, MAR_{IN}, READ 5. WMFC 6. MDR_{OUT}, PC_{IN}, END
JNE $8(%ESP) con salto relativo	**JNE $8(%EBX) con salto assoluto**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. if (EQUAL) END, ESP_{OUT}, V_{IN} 5. $OFFSET(IR)_{OUT}$, $SELECT_V$, ADD, Z_{IN} 6. Z_{OUT}, MAR_{IN}, READ 7. WMFC, PC_{OUT}, V_{IN} 8. MDR_{OUT}, $SELECT_V$, ADD, Z_{IN} 9. Z_{OUT}, PC_{IN}, END	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. if (EQUAL) END, EBX_{OUT}, V_{IN} 5. $OFFSET(IR)_{OUT}$, $SELECT_V$, ADD, Z_{IN} 6. Z_{OUT}, MAR_{IN}, READ 7. WMFC 8. MDR_{OUT}, PC_{IN}, END

- *PUSH value*

L'operazione di Push consiste nell'inserire un valore “value” nella cima dello Stack; occorre perciò decrementare il registro ESP di 4 per puntare ad una cella vuota dello Stack (superiore a quella attuale), caricare il nuovo valore di ESP sia in ESP sia in MAR, prendere il valore “value” e caricarlo nel registro MDR e dare il comando WRITE, in modo da scrivere il valore nella cella prescelta. Dopo WMFC l'istruzione termina. Il registro ESP viene decrementato di tanti byte quanti ne vengono messi sullo Stack (4 con “L” e 2 con “W”).

- *POP dest*

L'operazione di Pop consiste del prelevare il valore della cima dello Stack e caricarlo in “dest”; è l'esatto opposto della Push. Deve essere incrementato di 4 il valore di ESP in modo da puntare ad una cella sottostante (quindi non vuota) dello Stack, salvare il nuovo valore di ESP in ESP e caricarlo in MAR, poi si dà il comando di lettura READ e, dopo WMFC, si preleva il valore ottenuto da MDR e lo si carica nella

destinazione “dest”. Il registro ESP viene decrementato di tanti byte quanti ne vengono messi sullo Stack (4 con “L” e 2 con “W”). Il recupero di informazioni dello Stack mediante più chiamate dell’istruzione POP deve avvenire nell’ordine inverso del loro inserimento nello Stack mediante le istruzioni PUSH.

PUSH %EAX	**POP %EAX**
1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. $ESP_{out}, Select4, SUB, Z_{in}$ 5. $Z_{out}, MAR_{in}, ESP_{in}$ 6. $EAX_{out}, MDR_{in}, WRITE, WMFC$ 7. END	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. $ESP_{out}, Select4, ADD, Z_{in}, MAR_{in}, READ$ 5. $Z_{out}, ESP_{in}, WMFC$ 6. MDR_{out}, EAX_{in}, END

- *CALL value*

Call indica una chiamata a funzione; la funzione è identificata da “value”, che può essere un indirizzo, un'etichetta, un indirizzamento diretto. Si può dire che prima viene eseguita un'operazione di Push del valore di PC; infatti viene caricato il valore decrementato di 4 di ESP in MAR, poi viene caricato il valore di PC in MDR e si dà il comando di scrittura. Questo serve perché si salva in una cella vuota dello Stack il valore di ritorno della funzione, cioè il valore del PC (in quanto serve al microprocessore per ricordarsi cosa stesse facendo prima di iniziare ad eseguire la funzione chiamata).

CALL (%EAX) con salto relativo	**CALL etichetta con salto relativo**
1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.ESP_{OUT}, SELECT[4], SUB, Z_{IN} 5.Z_{OUT},ESP_{IN},MAR_{IN} 6.PC_{OUT}, MDR_{IN}, WRITE 7.WMFC, EAX_{OUT}, MAR_{IN}, READ 8.WMFC,PC_{OUT},V_{IN} 9.MDR_{OUT}, SELECT[V], ADD, Z_{IN} 10.Z_{OUT}, PC_{IN}, END	1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2. WMFC, Z_{OUT}, PC_{IN} 3. MDR_{OUT}, IR_{IN} 4. $ESP_{out}, Select4, SUB, Z_{in}$ 5. $Z_{out}, MAR_{in}, ESP_{in}$ 6. $PC_{out}, MDR_{in}, WRITE, V_{in}$ 7. $IR\_imm\_field_{out}, SelectV, ADD, Z_{in}, WMFC$ 8. Z_{out}, PC_{in}, END
CALL (%EAX) con salto assoluto	**CALL (%EAX + %EBX) con salto assoluto**
1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.ESP_{OUT}, SELECT[4], SUB, Z_{IN} 5.Z_{OUT}, ESP_{IN}, MAR_{IN} 6.PC_{OUT},MDR_{IN},WRITE 7. WMFC, EAX_{OUT}, MAR_{IN}, READ 8. WMFC 9. MDR_{OUT}, PC_{IN}, END	1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN} 2.WMFC, Z_{OUT}, PC_{IN} 3.MDR_{OUT}, IR_{IN} 4.ESP_{OUT}, SELECT[4], SUB, Z_{IN} 5.Z_{OUT}, ESP_{IN}, MAR_{IN} 6.PC_{OUT},MDR_{IN},WRITE 7.WMFC, EAX_{OUT},V_{IN} 8.EBX_{OUT}, SELECT[V], ADD, Z_{IN} 9. Z_{OUT}, MAR_{IN}, READ 10.WMFC 11.MDR_{OUT},PC_{IN}, END.

CALL (%EAX + $4) con salto assoluto

1.PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}
2.WMFC, Z_{OUT}, PC_{IN}
3.MDR_{OUT}, IR_{IN}

4.ESP_{OUT}, SELECT[4], SUB, Z_{IN}
5.Z_{OUT}, ESP_{IN},MAR_{IN}
6.PC_{OUT}, MDR_{IN}, WRITE
7.WMFC, EAX_{OUT},V_{IN}

8. $OFFSET(IR)_{OUT}$,SELECT[V], ADD, Z_{IN}
9.Z_{OUT},MAR_{IN}, READ
10.WMFC
11.MDR_{OUT}, PC_{IN}, END

- *RET*

La RET estrae dalla cima dello Stack un valore, lo interpreta come indirizzo di un'istruzione e fa saltare il processore a tale istruzione (il valore sullo Stack corrisponde all'indirizzo dell'istruzione successiva all'istruzione CALL). In poche parole, esegue l'esatto opposto di CALL, cioè preleva dallo Stack il valore di PC. Al momento dell'esecuzione della RET in cima allo Stack deve essere presente il valore che era stato messo dalla CALL. Quindi nella funzione il numero di istruzioni PUSH deve essere uguale al numero di istruzioni POP, affinché al momento dell'esecuzione della RET lo Stack sia nelle stesse condizioni in cui si trovava all'inizio della funzione.

RET

1. PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}
2. WMFC, Z_{OUT}, PC_{IN}
3. MDR_{OUT}, IR_{IN}
4. ESP_{OUT}, MAR_{IN}, READ, $SELECT_4$, ADD, Z_{IN}
5. WMFC, Z_{OUT}, ESP_{IN}
6. MDR_{OUT}, PC_{IN}, END

MEMORIE RAM, CACHE E VIRTUALI

Regole ed accorgimenti:
Un esercizio tipico è quello che consiste nel trovare la dimensione dell'indirizzo della RAM e della Cache. Per questo tipo di esercizio, è utile imparare delle piccole formule per identificare la dimensione dei campi che compongono l'indirizzo. I campi sono: parola, blocco, etichetta.

Innanzitutto, bisogna identificare la dimensione dell'indirizzo; per fare ciò, bisogna semplicemente prendere il numero di parole nella RAM e scomporlo in potenza di 2. Si otterrà perciò un numero 2^n, in cui "n" sarà il numero corrispondente alla dimensione dell'indirizzo. Sapere quanto valgono 1KB, 1MB, … in potenze di 2 può agevolare l'operazione: 1KB = 2^{10}, 1MB = 2^{20}, …

Successivamente bisogna scoprire la dimensione del campo parola; per fare ciò, bisogna considerare il numero di parole per blocco e trasformarlo in potenza di 2. Si ottiene perciò il numero 2^n, in cui "n" è il numero corrispondente alla dimensione del campo parola.

Per il campo blocco, è necessario ottenere la dimensione della Cache, dividendo il numero di parole nella Cache per il numero di parole per blocco. Infine, per ottenere la dimensione del campo blocco, bisogna dividere la dimensione della Cache per il tipo di Set, che può essere:

- Ad accesso diretto, in questo caso non teniamo conto del tipo di Set perché vale 1 ($2^0 = 1$);
- n-set associativa, in questo caso il tipo di Set vale 2^n;
- Completamente associativa, in questo caso non esiste il campo Blocco e per calcolare il campo Etichetta basterà sottrarre dai bit dell'indirizzo i bit del campo parola.

Il tutto si può riassumere attraverso delle "formule":

- dimensione Cache = parole Cache / parole Blocco
- blocco = dimensione Cache / tipo Set

Per ottenere il campo etichetta, occorre ottenere la dimensione della RAM, dividendo il numero di parole nella RAM per il numero di parole del blocco. Dopo ciò, si divide il numero appena ottenuto per il "Cache Set" (ovvero la dimensione del campo blocco trovata precedentemente). Il tutto si può riassumere attraverso delle "formule":

- dimensione RAM = parole RAM / parole Blocco
- etichetta = dimensione RAM / Cache Set

Per completare l'esercizio, occorre disegnare l'indirizzo ottenuto e verificare se la somma delle dimensioni dei campi restituisce un risultato uguale alla dimensione dell'indirizzo. Il disegno avrà una forma del genere:

ETICHETTA	**BLOCCO**	**PAROLA**
bit	bit	bit

bit Etichetta + bit Blocco + bit Parola = bit Indirizzo

In alcuni esercizi può capitare che vengano chieste le dimensioni dell'indirizzo della Cache. Per trovare tali dimensioni basta ridurre in potenza di 2 le parole della Cache e sottrarre i bit dei campi blocco e parola (questi due campi rimangono invariati, si va a modificare solo il campo etichetta). Il risultato sarà la dimensione del campo etichetta della Cache:
bit Etichetta Cache = bit Indirizzo Cache - bit Parola - bit Blocco

Un altro tipo di esercizio è quello di considerare quanti "Cache Miss" avvengono durante l'esecuzione di un programma, in base alla grandezza della Cache. In pratica basta calcolare quanto viene occupata la Cache caricando in essa i dati e calcolare quanti e quali dati devono essere sostituiti ciclicamente.

Esempio 1:

In una memoria Cache con blocchi di 4 parole, di dimensione 32 parole, inizialmente vuota e con corrispondenza diretta determinare se i seguenti indirizzi di parola sono "HIT" o "MISS".

1	*4*	*8*	*5*	*33*	*66*	*32*	*56*	*9*	*11*	*4*	*43*	*88*	*6*	*32*

Indirizzo blocco = indirizzo parola / dimensione blocco

0	1	2	1	8	16	8	14	2	2	1	10	22	1	8

Sulle colonne: numero dell'indirizzo di parola considerato
Sulle righe: blocchi della Cache (dimensione Cache / dimensione blocco)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	8	16	8	8	8	8	8	8	8	8	8
1		1	1	1	1	1	1	1	1	1	1	1	1	1	1
2			2	2	2	2	2	2	2	2	2	10	10	10	10
3															
4															
5															
6								14	14	14	14	14	22	22	22
7															
H/M	M	M	M	H	M	M	M	M	H	H	H	M	M	H	H

6 HIT, 9 MISS

Esempio 2:

Si consideri l'esempio precedente con la differenza che la memoria Cache ha corrispondenza associativa (con algoritmo LRU).*

Sulle colonne: numero dell'indirizzo di parola considerato
Sulle righe: blocchi della Cache

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1		1	1	1	1	1	1	1	1	1	1	1	1	1	1
2			2	2	2	2	2	2	2	2	2	2	2	2	2
3					8	8	8	8	8	8	8	8	8	8	8
4						16	16	16	16	16	16	16	16	16	16
5								14	14	14	14	14	14	14	14
6												10	10	10	10
7													22	22	22
H/M	M	M	M	H	M	M	H	M	H	H	H	M	M	H	H

7 HIT, 8 MISS

*: L'algoritmo LRU (Least Recently Used) sostituisce di volta in volta l'elemento usato meno di recente, ovvero la copia di blocco che da più tempo risulta essere inutilizzata; ciò si realizza tramite dei contatori che vengono incrementati o azzerati a seconda che ci sia stato un Hit o un Miss.

PIPELINE

Regole ed accorgimenti:
Questo tipo di esercizi consiste nell'elencare le varie fasi di Prelievo, Decodifica, Elaborazione, Memoria e Scrittura di una istruzione Assembly in base alle situazioni proposte e in base alle dipendenze di altre istruzioni. Innanzitutto, è buona cosa leggere bene ogni istruzione elencata per capire le dipendenze. Per capire quali sono le istruzioni dipendenti tra loro, basta vedere quali registri vengono utilizzati. Se due istruzioni usano lo stesso registro, bisogna sicuramente aspettare la fase S per iniziare la fase E.

Caso ideale:
Nella migliore delle ipotesi, un'istruzione parte con la sequenza P, D, E, M, S e la successiva parte invece con la sua fase P subito dopo la fase P dell'istruzione precedente (ovvero parte nello stesso ciclo della fase D dell'istruzione precedente).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Istruzione 1	P	D	E	M	S				
Istruzione 2		P	D	E	M	S			
Istruzione 3			P	D	E	M	S		

Dipendenze di dato (con 2 istruzioni):
Se un'istruzione dipende da quella precedente, la sua fase P inizia nello stesso istante dell'ultima fase D dell'istruzione precedente, mentre la sua fase D, oltre a durare per tutto il tempo delle altre fasi dell'istruzione precedente, si protrae per un altro ciclo dopo la fase S. Dopodiché l'istruzione prosegue con le altre sue fasi rimanenti.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Istruzione 1	P	D	E	M	S				
Istruzione 2		P	D	D	D	D	E	M	S

Inoltro di operandi:
Per alleviare il problema di stallo causato dalle dipendenze di dato, si usa l'inoltro degli operandi, ovvero si trasferisce il valore di un registro tra due istruzioni mediante l'ALU e i buffer che compongono la pipeline. È possibile fare questo perché il valore desiderato è in realtà disponibile alla fine della fase E dell'istruzione precedente, cioè quando l'ALU completa l'operazione; quindi l'hardware può inoltrare questo valore all'ingresso dell'ALU nella fase E di un'istruzione successiva.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Istruzione 1	P	D	E	M	S				
Istruzione 2		P	D	E	M	S			

In alternativa, le dipendenze di dato possono essere gestite via software mediante l'utilizzo delle NOP (no operation).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Istruzione 1	P	D	E	M	S				
NOP		P	D	E	M	S			
NOP			P	D	E	M	S		
NOP				P	D	E	M	S	
Istruzione 2					P	D	E	M	S

Dipendenze di dato (con 3 istruzioni):
Qualora invece ci siano 3 istruzioni e l'ultima dipende dalla prima ma non dalla seconda, lo stato in più descritto precedentemente deve essere "in più" rispetto alla prima istruzione (cioè da quella che dipende l'istruzione considerata). La terza istruzione, quindi, deve partire solo dopo che la prima istruzione ha finito.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Istruzione 1	P	D	E	M	S				
Istruzione 2		P	D	E	M	S			
Istruzione 3			P	D	D	D	E	M	S

DOMANDE DI TEORIA

Domanda 1:
Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 5 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

Una pipeline ad N stadi è in grado di aumentare le prestazioni idealmente di N volte e di raggiungere lo scopo di riuscire ad eseguire un'istruzione a ogni ciclo di clock. Quindi se da una pipeline a 2 stadi si passa ad una a 5 stadi, essa può aumentare idealmente le prestazioni di 3 volte. Il minimo incremento delle prestazioni che si può ottenere è di 1 volta. Ma l'aumento degli stadi nella pipeline aumenta anche la probabilità di stallo della CPU, e quindi è possibile che non si ottenga alcun incremento.

Domanda 2:
Descrivere il meccanismo e l'utilità della predizione dei salti.

I salti interrompono il flusso della pipeline e per minimizzare il numero di interruzioni è necessario uno schema di brach prediction. La pipeline ora contiene un'istruzione di salto condizionato e altre istruzioni successive, ma non si sa ancora se tali istruzioni serviranno o meno. La predizione dei salti può essere statica o dinamica:

- Predizione statica (compile-time), è realizzata dal compilatore e il risultato della predizione è assumere sempre che il salto non vada effettuato e prelevare la successiva istruzione. Se il salto non verrà effettuato l'esecuzione del programma potrà continuare tranquillamente, se invece verrà effettuato l'istruzione che è stata prelevata dovrà essere scartata.
- Predizione dinamica (run-time), l'hardware del processore valuta la probabilità di effettuare un certo salto tenendo traccia delle decisioni di salto ogni volta che un'istruzione di salto viene eseguita. A questo punto si ipotizza che la decisione da prendere nella presente istanza possa essere la stessa di quella presa in precedenza (se è stato eseguito un salto nella precedente istruzione, si farà l'ipotesi che il salto avvenga nuovamente).

Domanda 3:
Quali sono le motivazioni che fanno preferire la realizzazione di unità di controllo cablate rispetto a quelle micro-programmate. Definire lo schema di un controllore cablato indicando i segnali utilizzati e la funzione dei blocchi presenti.

Le unità di controllo cablate sono pilotate da un segnale di clock. Ogni stato del contatore corrisponde ad uno dei passi dell'istruzione in esecuzione. Le unità di controllo cablate vantano una buona velocità.

Le unità di controllo micro-programmate sono invece molto più lente in quanto, per prelevare le istruzioni successive da eseguire, devono continuamente accedere alla memoria.

Schema del controllore cablato: i segnali di controllo richiesti sono univocamente determinati dal contenuto del contatore dei passi di controllo (PC), dal contenuto del registro delle istruzioni (IR) e dal contenuto del codice di condizione e di altri flag di stato che rappresentano gli stati di diverse parti della CPU e delle linee di controllo legate ad essa.

Domanda 4:
Elencare le ottimizzazioni che devono essere eseguite sull'architettura di un calcolatore per raggiungere l'obiettivo di avere, per la maggioranza delle istruzioni, CPI = 1.

Per raggiungere l'obiettivo di ottenere un CPI medio = 1 ho bisogno di:
- Cache efficiente (separando la cache dei dati dalla cache delle istruzioni);
- Pipeline efficiente e bilanciata (utilizzando un'architettura di tipo CISC/RISC).

Domanda 5:
Si descriva il meccanismo di interrupt.

Un interrupt è un meccanismo che permette, ad un evento esterno al calcolatore, di provocare il trasferimento del controllo da un programma ad un altro, attraverso la ISR (Interrupt Service Routine) e seguendo il "principio di trasparenza", ossia il programma o il processo in esecuzione non deve ricevere modifiche a seguito dell'interrupt. Quindi, quando il processore riceve un interrupt:
- Memorizza il contenuto di PSW e PC sullo Stack;
- Esegue le istruzioni della procedura che ha richiesto l'interrupt;
- Una volta terminato l'interrupt, ripristina il contenuto di PSW e PC.

Domanda 6:
Quali sono le motivazioni che spingono i microprocessori moderni a essere dotati di una pipeline.

Lo scopo di adottare le pipeline nei processori moderni è quello di ridurre il più possibile il CPI medio e di ottenere CPI medio = 1. Per avere dei netti miglioramenti in quantità di tempo è indispensabile cercare di fare entrare la CPU in stallo il meno possibile, di tenerla quindi sempre a regime. Per fare questo, un meccanismo indispensabile è la predizione dei salti (branch prediction). Teoricamente se una pipeline ha N stadi, aumenta l'efficienza di N volte. In realtà non è così poiché aumenta la probabilità di dipendenza fra i dati, motivo per cui i processori moderni cercano di avere molti stadi, ma non eccessivi, ed un sistema di branch prediction il più affidabile possibile.

Domanda 7:
Quali sono i vantaggi e gli svantaggi delle memorie completamente associative rispetto alle memorie non associative.

<u>Memorie completamente associative</u>
- Vantaggi: non ci sono blocchi di cache predefiniti dove memorizzare i blocchi di RAM, quindi la cache è utilizzabile per intero ed i conflitti di blocco compaiono solo quando la cache è piena (questo implica massima libertà nella scelta di dove posizionare in cache il blocco di RAM); lo spazio della cache è usato in modo molto efficiente.

- Svantaggi: è necessario un algoritmo di ricerca per trovare i blocchi ricercati all'interno della cache; costo elevato.

Memorie non associative (ad accesso diretto)

- Vantaggi: facile da realizzare; unico blocco in Cache ove memorizzare uno specifico blocco della RAM, quindi è sufficiente confrontare i bit più significativi dell'indirizzo con l'etichetta associata al blocco della cache per sapere se il blocco è presente oppure no.
- Svantaggi: non molto flessibile.

Domanda 8:
Quali sono i vantaggi di una cache a indirizzamento diretto?

Nella cache ad accesso diretto il set è composto da un solo blocco e ogni blocco della RAM ha un unico posto in cache ove essere memorizzato. I bit più significativi dell'indirizzo vengono confrontati con l'etichetta associata al blocco della cache: se coincidono, la parola cercata si trova in quel blocco della cache, altrimenti non c'è altra posizione ove cercarla e bisogna caricarla dalla RAM. Il metodo di indirizzamento diretto è facile da realizzarsi (anche se non è molto flessibile), inoltre, qualora un programma facesse riferimento sempre e solo a specifiche zone di memoria della RAM, gran parte della cache non verrebbe utilizzata, rallentando notevolmente l'esecuzione del programma.

Domanda 9:
Descrivere il meccanismo della memoria virtuale, specificando quali componenti devono essere presenti nella CPU per realizzarlo.

Nella maggior parte dei calcolatori odierni la memoria centrale fisicamente installata non è grande quanto lo spazio di indirizzamento gestibile da parte del processore. Il meccanismo della memoria virtuale usa il disco rigido per far vedere alla CPU una memoria RAM più grande. Ciò si realizza traducendo l'indirizzo logico (generato dal processore per prelevare l'istruzione) in indirizzo fisico. I componenti necessari per realizzare la memoria virtuale sono il processore, la memoria di massa (disco rigido), la memoria centrale (RAM), la memoria cache e l'unità di gestione di memoria (MMU) che si occupa dei page fault (equivalenti ai cache miss).

Domanda 10:
Si spieghino le proprietà di località spaziale e temporale.

La proprietà di località spaziale dice che se il dato all'indirizzo i viene usato, allora molto probabilmente verrà usato anche il dato all'indirizzo i+q (con q diverso da 0 e piccolo).
La proprietà di località temporale dice che se un'istruzione viene prelevata al ciclo i, allora molto probabilmente verrà prelevata nuovamente anche al ciclo i+p (con p maggiore di 0 e piccolo).
Riassumendo, secondo la località spazio-temporale, se un blocco di parole del programma va in uso da parte del processore, molto probabilmente entro breve tempo e per più volte esso verrà usato nuovamente.

ESERCIZI SU MEMORIE RAM, CACHE E VIRTUALI

Esercizio 1:
Si consideri una CPU dotata di memoria Cache 4-associativa di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 4M parole. Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

Dati:
Cache 4-set associativa, 8k parole, blocco da 64 parole; RAM da 4M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 4M parole = $2^2 * 2^{20} = 2^{22}$ → 22 bit in totale

Dimensione del campo parola:
blocco da 64 parole = 2^6 → 6 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 8k parole / 64 parole = 8k / 64 = $2^{13} / 2^6 = 2^7$
blocco = dimensione Cache / tipo Set = $2^7 / 2^2$ (4-set associativa) = 2^5 → 5 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 4M parole / 64 parole = 4M / 64 = $2^{22} / 2^6 = 2^{16}$
etichetta = dimensione RAM / Cache Set = $2^{16} / 2^5 = 2^{11}$ → 11 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 11 + 5 + 6 = 22

Schema:

ETICHETTA	BLOCCO	PAROLA
11 bit	5 bit	6 bit

Esercizio 2:
Si consideri una memoria Cache 4-set associativa della dimensione di 32 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 1 Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

Dati:
Cache 4-set associativa, 32k parole, blocco da 1024 parole; RAM da 1M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 1M parole = 2^{20} → 20 bit in totale

Dimensione del campo parola:
blocco da 1024 parole = 2^{10} → 10 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 32k parole / 1024 parole = $2^5 * 2^{10} / 2^{10} = 2^5$
blocco = dimensione Cache / tipo Set = $2^5 / 2^2$ (4-set associativa) = 2^3 → 3 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 1M parole / 1024 parole = $2^{20} / 2^{10} = 2^{10}$
etichetta = dimensione RAM / Cache Set = $2^{10} / 2^3 = 2^7$ → 7 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 7 + 3 + 10 = 20

Schema (RAM):

ETICHETTA	BLOCCO	PAROLA
7 bit	3 bit	10 bit

Dimensioni dell'indirizzo per la Cache:
Cache da 32k parole = $2^5 * 2^{10} = 2^{15}$ → 15 bit in totale

Le dimensioni dei campi parola e blocco restano uguali, quindi si riduce la dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola - bit Blocco → 15 - 10 - 3 = 2 bit per il campo etichetta

Schema (Cache):

ETICHETTA	BLOCCO	PAROLA
2 bit	3 bit	10 bit

Esercizio 3:
Si consideri una memoria Cache 4-set associativa della dimensione di 16 Kbyte con 512 byte per blocco. La cache è collegata ad una memoria di 2Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

Dati:
Cache 4-set associativa, 16k parole, blocco da 512 parole; RAM da 2M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 2M parole = $2 * 2^{20} = 2^{21}$ → 21 bit in totale

Dimensione del campo parola:
blocco da 512 parole = 2^9 → 9 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 16k parole / 512 parole = $2^4 * 2^{10} / 2^9 = 2^5$
blocco = dimensione Cache / tipo Set = $2^5 / 2^2$ (4-set associativa) = 2^3 → 3 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 2M parole / 512 parole = $2^{21} / 2^9 = 2^{12}$
etichetta = dimensione RAM / Cache Set = $2^{12} / 2^3 = 2^9$ → 9 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 9 + 3 + 9 = 21

Schema (RAM):

ETICHETTA	**BLOCCO**	**PAROLA**
9 bit	3 bit	9 bit

Dimensioni dell'indirizzo per la Cache:
Cache da 16k parole = $2^4 * 2^{10} = 2^{14}$ → 14 bit in totale

Le dimensioni dei campi parola e blocco restano uguali, quindi si riduce la dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola - bit Blocco → 14 - 9 - 3 = 2 bit per il campo etichetta

Schema (Cache):

ETICHETTA	**BLOCCO**	**PAROLA**
2 bit	3 bit	9 bit

Esercizio 4:
Si consideri una CPU dotata di memoria cache di 128 Kbyte con 64 byte per blocco. Questa CPU è collegata ad una memoria RAM da 16 Mbyte indirizzabile per byte. Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Definire queste dimensioni per una memoria cache di tipo:

a) ad accesso diretto
b) 2-set associativa
c) completamente associativa

Dati (a):
Cache ad accesso diretto, 128k parole, blocco da 64 parole; RAM da 16M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 16M parole = $2^4 * 2^{20} = 2^{24}$ → 24 bit in totale

Dimensione del campo parola:
blocco da 64 parole = 2^6 → 6 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 128k parole / 64 parole = $2^7 * 2^{10} / 2^6 = 2^{11}$
blocco = dimensione Cache / tipo Set = $2^{11} / 2^0$ (accesso diretto) = 2^{11} → 11 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 16M parole / 64 parole = $2^4 * 2^{20} / 2^6 = 2^{18}$
etichetta = dimensione RAM / Cache Set = $2^{18} / 2^{11} = 2^7$ → 7 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 7 + 11 + 6 = 24

Schema:

ETICHETTA	BLOCCO	PAROLA
7 bit	11 bit	6 bit

Dati (b):
Cache 2-set associativa, 128k parole, blocco da 64 parole; RAM da 16M parole.

Le dimensioni dell'indirizzo e del campo parola non cambiano rispetto al punto (a).

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 128k parole / 64 parole = $2^7 * 2^{10} / 2^6 = 2^{11}$
blocco = dimensione Cache / tipo Set = $2^{11} / 2^1$ (2-set associativa) = 2^{10} → 10 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 16M parole / 64 parole = $2^4 * 2^{20} / 2^6 = 2^{18}$
etichetta = dimensione RAM / Cache Set = $2^{18} / 2^{10} = 2^8$ → 8 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 8 + 10 + 6 = 24

Schema:

ETICHETTA	BLOCCO	PAROLA
8 bit	10 bit	6 bit

Dati (c):
Cache completamente associativa, 128k parole, blocco da 64 parole; RAM da 16M parole.

Le dimensioni dell'indirizzo e del campo parola non cambiano rispetto ai punti (a) e (b).

In caso di memoria Cache completamente associativa il campo blocco non esiste.

Dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola → 24 - 6 = 18

Schema:

ETICHETTA	PAROLA
18 bit	6 bit

Esercizio 5:
Si consideri una memoria cache completamente associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 16 Mbyte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

Dati:
Cache completamente associativa, 64k parole, blocco da 1024 parole; RAM da 16M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 16M parole = $2^4 * 2^{20} = 2^{24}$ → 24 bit in totale

Dimensione del campo parola:
blocco da 1024 parole = 2^{10} → 10 bit per il campo parola

In caso di memoria Cache completamente associativa il campo blocco non esiste.

Dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola → 24 - 10 = 14

Schema (RAM):

ETICHETTA	**PAROLA**
14 bit	10 bit

Dimensioni dell'indirizzo per la Cache:
Cache da 64k parole = $2^6 * 2^{10} = 2^{16}$ → 16 bit in totale

Le dimensioni dei campi parola e blocco restano uguali, quindi si riduce la dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola - bit Blocco → 16 - 10 - 0 = 6 bit per il campo etichetta

Schema (Cache):

ETICHETTA	**PAROLA**
6 bit	10 bit

Esercizio 6:
Si consideri una memoria cache 2-set associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

Dati:
Cache 2-set associativa, 64k parole, blocco da 1024 parole; RAM da 4M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 4M parole = $2^2 * 2^{20} = 2^{22}$ → 22 bit in totale

Dimensione del campo parola:
blocco da 1024 parole = 2^{10} → 10 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 64k parole / 1024 parole = $2^6 * 2^{10} / 2^{10} = 2^6$
blocco = dimensione Cache / tipo Set = $2^6 / 2^1$ (2-set associativa) = 2^5 → 5 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 4M parole / 1024 parole = 2^{22} / 2^{10} = 2^{12}
etichetta = dimensione RAM / Cache Set = 2^{12} / 2^5 = 2^7 → 7 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 7 + 5 + 10 = 22

Schema (RAM):

ETICHETTA	BLOCCO	PAROLA
7 bit	5 bit	10 bit

Dimensioni dell'indirizzo per la Cache:
Cache da 64k parole = 2^6 * 2^{10} = 2^{16} → 16 bit in totale

Le dimensioni dei campi parola e blocco restano uguali, quindi si riduce la dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola - bit Blocco → 16 - 10 - 5 = 1 bit per il campo etichetta

Schema (Cache):

ETICHETTA	BLOCCO	PAROLA
1 bit	5 bit	10 bit

Esercizio 7:
Si consideri una memoria cache 4-set associativa della dimensione di 32 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

Dati:
Cache 4-set associativa, 32k parole, blocco da 1024 parole; RAM da 4M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 4M parole = 2^2 * 2^{20} → 22 bit in totale

Dimensione del campo parola:
blocco da 1024 parole = 2^{10} → 10 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 32k parole / 1024 parole = 2^5 * 2^{10} / 2^{10} = 2^5
blocco = dimensione Cache / tipo Set = 2^5 / 2^2 (4-set associativa) = 2^3 → 3 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 4M parole / 1024 parole = 2^{22} / 2^{10} = 2^{12}
etichetta = dimensione RAM / Cache Set = 2^{12} / 2^3 = 2^9 → 9 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 9 + 3 + 10 = 20

Schema (RAM):

ETICHETTA	BLOCCO	PAROLA
9 bit	3 bit	10 bit

Dimensioni dell'indirizzo per la Cache:
Cache da 32k parole = $2^5 * 2^{10} = 2^{15}$ → 15 bit in totale

Le dimensioni dei campi parola e blocco restano uguali, quindi si riduce la dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola - bit Blocco → 15 - 10 - 3 = 2 bit per il campo etichetta

Schema (Cache):

ETICHETTA	BLOCCO	PAROLA
2 bit	3 bit	10 bit

Esercizio 8:
Si consideri una CPU dotata di memoria cache 2-set associativa di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 8M parole. Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

Dati:
Cache 2-set associativa, 8k parole, blocco da 64 parole; RAM da 8M parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 8M parole = $2^3 * 2^{20} = 2^{23}$ → 23 bit in totale

Dimensione del campo parola:
blocco da 64 parole = 2^6 → 6 bit per il campo parola

Dimensione del campo blocco:
dimensione Cache = parole Cache / parole Blocco = 8k parole / 64 parole = 8k / 64 = $2^{13} / 2^6 = 2^7$
blocco = dimensione Cache / tipo Set = $2^7 / 2^1$ (2-set associativa) = 2^6 → 6 bit per il campo blocco

Dimensione del campo etichetta:
dimensione RAM = parole RAM / parole Blocco = 8M parole / 64 parole = 8M / 64 = $2^{23} / 2^6 = 2^{17}$
etichetta = dimensione RAM / Cache Set = $2^{17} / 2^6 = 2^{11}$ → 11 bit per il campo etichetta

Controllo:
bit Etichetta + bit Blocco + bit Parola = bit Indirizzo → 11 + 6 + 6 = 23

Schema:

ETICHETTA	BLOCCO	PAROLA
11 bit	6 bit	6 bit

Esercizio 9:
Identificare le parti dell'indirizzo RAM di una memoria da 8GB, indirizzabile per byte, nel caso sia collegata ad una memoria Cache completamente associativa da 8MB con pagine da 1kB.

Dati:
Cache completamente associativa, 8M parole, blocco da 1k parole; RAM da 8G parole.

Dimensioni dell'indirizzo necessario ad indirizzare tutta la memoria RAM:
RAM da 8G parole = $2^3 * 2^{30} = 2^{33}$ → 33 bit in totale

Dimensione del campo parola:
blocco da 1k parole = 2^{10} → 10 bit per il campo parola

In caso di memoria Cache completamente associativa il campo blocco non esiste.

Dimensione del campo etichetta:
bit Etichetta = bit Indirizzo - bit Parola → 33 - 10 = 23

Schema:

ETICHETTA	**PAROLA**
23 bit	10 bit

ESERCIZI SULLA PIPELINE

Ipotesi:

- Cache separata per istruzioni e dati, così da poter elaborare gli stadi D e M allo stesso ciclo
- Cache hit per tutte le operazioni di memoria, così da gestirle in un singolo ciclo

Esercizi svolti con procedimento:

- *Esercizio 1*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
addl %eax, %ebx	P	D	E	M	S									
movl ind, %ecx		P	D	E	M	S								
subl %ebx, %ecx			P	D	D	D	D	E	M	S				
jz loop							P	D	D	D	D	E	M	S

1. La prima istruzione parte normalmente con P, D, E, M, S
2. La seconda istruzione non ha registri in "comune" con la prima, quindi non dobbiamo aspettare che finisca per iniziare la fase E
3. Nella terza istruzione vediamo che il registro ECX è usato nell'istruzione precedente, quindi dobbiamo aspettare che la seconda istruzione termini e aggiungere una fase D prima di partire con la fase E
4. Nella quarta istruzione si presenta un salto condizionato, in questo caso dobbiamo sempre aspettare che l'istruzione precedente abbia finito e aggiungere una fase D prima di partire con la fase E*

- *Esercizio 2*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
mull %eax, %ebx	P	D	E	M	S									
movl %ecx, ind		P	D	E	M	S								
cmpl %ebx, 0h			P	D	D	D	E	M	S					
jnz inizio						P	D	D	D	D	E	M	S	

1. La prima istruzione parte normalmente con P, D, E, M, S
2. La seconda istruzione non ha registri in "comune" con la prima, quindi non dobbiamo aspettare che finisca per iniziare la fase E
3. Nella terza istruzione vediamo che il registro EBX è usato nella prima istruzione, quindi dobbiamo aspettare che la prima istruzione termini e aggiungere una fase D prima di partire con la fase E

4. Nella quarta istruzione si presenta un salto condizionato, in questo caso dobbiamo sempre aspettare che l'istruzione precedente abbia finito e aggiungere una fase D prima di partire con la fase E*

- *Esercizio 3 (con condizione di salto falsa)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
cmpl %eax, %ebx	P	D	E	M	S									
jz start		P	D	D	D	D	E	M	S					
subl %ebx, %ecx						P	D	D	D	D	E	M	S	
movl %edx, data										P	D	E	M	S

1. La prima istruzione parte normalmente con P, D, E, M, S
2. Nella seconda istruzione si presenta un salto condizionato, in questo caso dobbiamo sempre aspettare che l'istruzione precedente abbia finito e aggiungere una fase D prima di partire con la fase E*
3. Nella terza istruzione vediamo che il registro EBX è usato nella prima istruzione ma possiamo notare che tale istruzione si è completata al passo 5, dovendo noi partire al passo 6 non ci crea nessun problema. Però dobbiamo stare attenti perché nelle specifiche dell'esercizio ci è stato detto che la condizione del salto è falsa, quindi dobbiamo aspettare che l'istruzione del salto finisca, aggiungere una fase D e solo dopo partire con la fase E*
4. Nella quarta istruzione non abbiamo registri in comune con nessuna delle precedenti istruzioni quindi possiamo far partire la fase E prima che si completi la precedente

- *Esercizio 4 (con condizione di salto falsa)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
loop: addl %eax, %ebx	P	D	E	M	S													
movl ind, %ecx		P	D	E	M	S												
subl %ebx, %ecx			P	D	D	D	D	E	M	S								
jz loop							P	D	D	D	D	E	M	S				
movl %ecx, ind											P	D	D	D	D	E	M	S

1. La prima istruzione parte normalmente con P, D, E, M, S
2. La seconda istruzione non ha registri in "comune" con la prima, quindi non dobbiamo aspettare che finisca per iniziare la fase E
3. Nella terza istruzione vediamo che il registro ECX è usato nell'istruzione precedente, quindi dobbiamo aspettare che la seconda istruzione termini e aggiungere una fase D prima di partire con la fase E
4. Nella quarta istruzione si presenta un salto condizionato, in questo caso dobbiamo sempre aspettare che l'istruzione precedente abbia finito e aggiungere una fase D prima di partire con la fase E*

5. Nella quinta istruzione non abbiamo registri in comune con nessuna delle precedenti istruzioni ma, siccome la condizione del salto è falsa, dobbiamo aspettare che l'istruzione del salto finisca, aggiungere una fase D e solo dopo partire con la fase E

- *Esercizio 5*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
addl %eax, %ebx	P	D	E	M	S													
movl %ebx, %ecx		P	D	D	D	D	E	M	S									
subl %eax, %ecx						P	D	D	D	D	E	M	S					
jz loop										P	D	D	D	D	E	M	S	

1. La prima istruzione parte normalmente con P, D, E, M, S
2. Nella seconda istruzione vediamo che il registro EBX è usato nella istruzione precedente, quindi dobbiamo aspettare che la prima istruzione termini e aggiungere una fase D prima di partire con la fase E
3. Nella terza istruzione vediamo che il registro ECX è usato nella istruzione precedente, quindi dobbiamo aspettare che la seconda istruzione termini e aggiungere una fase D prima di partire con la fase E
4. Nella quarta istruzione si presenta un salto condizionato, in questo caso dobbiamo sempre aspettare che l'istruzione precedente abbia finito e aggiungere una fase D prima di partire con la fase E*

- *Esercizio 6*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
addl %eax, %ebx	P	D	E	M	S									
movl ind, %ecx		P	D	E	M	S								
subl %ebx, %ecx			P	D	D	D	D	E	M	S				
jz loop							P	D	E					

1. Parte normalmente
2. Nessuna criticità
3. Il registro ECX è usato nell'istruzione precedente, quindi dobbiamo attendere la sua disponibilità
4. Il salto condizionato non dipende da niente (ignoro le fasi M e S)

- *Esercizio 6 (ottimizzazione)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
addl %eax, %ebx	P	D	E	M	S									
movl ind, %ecx		P	D	E	M	S								
subl %ebx, %ecx			P	D	D	E	M	S						
jz loop					P	D	D							

1. Parte normalmente
2. Nessuna criticità
3. Il registro ECX è usato nell'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 5 dopo la fase M per la presenza di "ind") al ciclo 6 per la fase E tramite la ALU
4. Il salto condizionato non dipende da niente, ma può essere ottimizzato in modo da essere eseguito entro la fase D (che deve comunque attendere un ciclo in più)

- *Esercizio 7*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
mull %eax, %ebx	P	D	E	M	S									
movl %ecx, ind		P	D	E	M	S								
cmpl %ebx, 0h			P	D	D	D	E	M	S					
jnz inizio						P	D	D	D	D	E			

1. Parte normalmente
2. Nessuna criticità
3. Il registro EBX è usato in un'istruzione precedente, quindi dobbiamo attendere la sua disponibilità
4. Il salto condizionato dipende dalla CMPL precedente, quindi devo attendere il risultato del confronto che viene salvato al ciclo 9 (ignoro le fasi M e S)

- *Esercizio 7 (ottimizzazione)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
mull %eax, %ebx	P	D	E	M	S									
movl %ecx, ind		P	D	E	M	S								
cmpl %ebx, 0h			P	D	E	M	S							
jnz inizio				P	D	D								

1. Parte normalmente

2. Nessuna criticità
3. Il registro EBX è usato in un'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 3) al ciclo 5 per la fase E tramite la ALU
4. Il salto condizionato dipende dalla CMPL precedente, ma possiamo inoltrare il suo risultato (disponibile al ciclo 5) al ciclo 6 per la fase D del salto (opportunamente ottimizzato in modo da essere eseguito entro la fase D, che deve comunque attendere un ciclo in più)

- *Esercizio 8*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
subl $2, %ebx	P	D	E	M	S									
movl ind, %ecx		P	D	E	M	S								
addl %eax, %ecx			P	D	D	D	D	E	M	S				
movl ind, %ebx							P	D	E	M	S			
movl %ecx, ind								P	D	D	D	E	M	S

1. Parte normalmente
2. Nessuna criticità
3. Il registro ECX è usato nell'istruzione precedente, quindi dobbiamo attendere la sua disponibilità
4. La fase P deve partire al ciclo 7 (in corrispondenza dell'ultima fase D dell'istruzione precedente)
5. Il registro ECX è usato in un'istruzione precedente, quindi dobbiamo attendere la sua disponibilità

- *Esercizio 8 (ottimizzazione)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
movl ind, %ecx	P	D	E	M	S									
subl $2, %ebx		P	D	E	M	S								
addl %eax, %ecx			P	D	E	M	S							
movl ind, %ebx				P	D	E	M	S						
movl %ecx, ind					P	D	E	M	S					

1. Scambio la seconda istruzione con la prima (per distanziare gli usi di ECX)
2. Nessuna criticità
3. Il registro ECX è usato in un'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 4 dopo la fase M per la presenza di "ind") al ciclo 5 per la fase E tramite la ALU
4. Nessuna criticità perché il registro EBX verrà modificato solo in corrispondenza della fase S dopo che avrò preso "ind" dalla memoria nella fase M
5. Il registro ECX è usato in un'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 5 dopo la fase E) al ciclo 7 per la fase E

- *Esercizio 9 (j1 si riferisce alla prima istruzione)*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
addl %ebx, %ecx	P	D	E	M	S													
movl ind, %ebx		P	D	E	M	S												
cmpl %ebx, %ecx			P	D	D	D	D	E	M	S								
movl ind, %eax							P	D	E	M	S							
jnz j1								P	D	D	D	E						
subl %ebx, %eax											P	D	E	M	S			

1. Parte normalmente
2. Nessuna criticità perché il registro EBX verrà modificato solo in corrispondenza della fase S dopo che avrò preso "ind" dalla memoria nella fase M
3. Il registro EBX è usato nell'istruzione precedente, quindi dobbiamo attendere la sua disponibilità
4. La fase P deve partire al ciclo 7 (in corrispondenza dell'ultima fase D dell'istruzione precedente)
5. Il salto condizionato dipende dalla CMPL precedente, quindi devo attendere il risultato del confronto che viene salvato al ciclo 10 (ignoro le fasi M e S)
6. Il registro EBX è usato in un'istruzione precedente, ma possiamo notare che tale istruzione si è completata al passo 11, dovendo noi partire al passo 11 non ci crea nessun problema.

- *Esercizio 9 (ottimizzazione)*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
addl %ebx, %ecx	P	D	E	M	S													
movl ind, %ebx		P	D	E	M	S												
movl ind, %eax			P	D	E	M	S											
cmpl %ebx, %ecx				P	D	E	M	S										
jnz j1					P	D	D											
subl %ebx, %eax							P	D	E	M	S							

1. Parte normalmente
2. Nessuna criticità perché il registro EBX verrà modificato solo in corrispondenza della fase S dopo che avrò preso "ind" dalla memoria nella fase M
3. Scambio la quarta istruzione con la terza (per distanziare gli usi di EBX)
4. Il registro EBX è usato in un'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 5 dopo la fase M per la presenza di "ind") al ciclo 6 per la fase E. Il registro ECX è usato in un'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 3 dopo la fase E) al ciclo 6 per la fase E

5. Il salto condizionato dipende dalla CMPL precedente, ma possiamo inoltrare il suo risultato (disponibile al ciclo 6) al ciclo 7 per la fase D del salto (opportunamente ottimizzato in modo da essere eseguito entro la fase D, che deve comunque attendere un ciclo in più)
6. Nessuna criticità perché il registro EAX viene modificato alla fine del ciclo 7 e l'istruzione inizia al ciclo 8

- *Esercizio 10 (init si riferisce alla prima istruzione)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
movl %ecx, ind	P	D	E	M	S													
addl $4, %ebx		P	D	E	M	S												
cmpl 0, %ebx			P	D	D	D	D	E	M	S								
jnz init							P	D	D	D	D	E	M	S				
addl %eax, %ecx											P	D	D	D	D	E	M	S

1. Parte normalmente
2. Nessuna criticità
3. Il registro EBX è usato nell'istruzione precedente, quindi dobbiamo attendere la sua disponibilità
4. Il salto condizionato dipende dalla CMPL precedente, quindi devo attendere il risultato del confronto che viene salvato al ciclo 10
5. Nessuna criticità, eseguo l'istruzione dopo aver terminato quella del salto

- *Esercizio 10 (ottimizzazione)*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
movl %ecx, ind	P	D	E	M	S													
addl $4, %ebx		P	D	E	M	S												
cmpl 0, %ebx			P	D	E	M	S											
jnz init				P	D	D												
addl %eax, %ecx						P	D	E	M	S								

1. Parte normalmente
2. Nessuna criticità
3. Il registro EBX è usato nell'istruzione precedente, ma possiamo inoltrare l'operando (disponibile al ciclo 4 dopo la fase E) al ciclo 5 per la fase E
4. Il salto condizionato dipende dalla CMPL precedente, ma possiamo inoltrare il suo risultato (disponibile al ciclo 5) al ciclo 6 per la fase D del salto (opportunamente ottimizzato in modo da essere eseguito entro la fase D, che deve comunque attendere un ciclo in più)
5. Nessuna criticità