Architettura degli elaboratori
Esercitazione

UniVR - Dipartimento di Informatica

Fabio Irimie

2° Semestre 2023/2024

Indice

1 Microoperazioni

I seguenti esercizi fanno riferimento all'architettura con 1 bus.

1.1 Esercizio 1 (Fetch)

Esercizio 1.1
Scrivere le microistruzioni per effettuare il fetch di un'istruzione, commentando ogni passo.

`Fetch`

F
1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
   Estraggo dal program counter l'indirizzo dell'istruzione da eseguire e lo metto nel MAR per ottenere l'istruzione. Successivamente metto imposto $SELECT_4$ che seleziona la costante 4 dal multiplexer per sommarla al program counter che si trova già nella ALU, questo equivale ad andare all'istruzione successiva, cioè PC + 1 word. Il risultato della somma viene salvato nel registro Z.
2. Z_{out}, PC_{in}, $WMFC$
   Estraggo l'indirizzo dell'istruzione successiva dall'registro Z e lo inserisco nel program counter mentre aspetto che la funzione di lettura del MAR venga completata.
3. MDR_{out}, IR_{in}
   Una volta che viene letto il dato all'indirizzo inserito nel MAR il risultato viene messo in MDR e successivamente trasferito nell'IR completando così il fetch dell'istruzione.

1.2 Esercizio 2 (Inc %reg)

Esercizio 1.2
Descrivere le microistruzioni relative alla seguente istruzione:

`INC %EAX`

F
1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
2. Z_{out}, PC_{in}, $WMFC$
3. MDR_{out}, IR_{in},

DE
4. EAX_{out}, $SELECT_0$, CB, ADD, Z_{in}
5. Z_{out}, EAX_{in}, END

1.3 Esercizio 3 (Inc var)

Esercizio 1.3
Descrivere le microistruzioni relative a questa istruzione:

`INC variable`

Si assume che la variabile 'variable' sia costituita da un indirizzo immediato, direttamente codificato nell'istruzione ($IR\_imm\_field_{\_out}$).

F
1. PC_{out}, ; MAR_{in}, $SELECT_4$, ADD, Z_{in}
2. Z_{out}, PC_{in}, $WMFC$
3. MDR_{out}, IR_{in}

DE
4. $IR\_imm\_field_{out}$, MAR_{in}, $READ$, $WMFC$
5. MDR_{out}, $SELECT_0$, CB, ADD, Z_{in}
6. Z_{out}, MDR_{in}, $WRITE$, $WMFC$, END

1.4 Esercizio 4 (CALL rel)

Esercizio 1.4
Si descrivano le microistruzioni relative alla seguente istruzione:

`CALL etichetta`

Assunzioni:

- *La flag 'etichetta' è costituita da un indirizzo immediato, direttamente codificato nell'istruzione;*
- *salto relativo, in quanto l'indirizzo di salto è specificato tramite etichetta simbolica.*

F
1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
2. Z_{out}, PC_{in}, $WMFC$
3. MDR_{out}, IR_{in}

DE
4. ESP_{out}, $SELECT_4$, SUB, Z_{in}
5. Z_{out}, MAR_{in}, ESP_{in}
6. PC_{out}, MDR_{in}, $WRITE$, V_{in}
7. $IR\_imm\_field_{out}$, $SELECT_V$, ADD, Z_{in}, $WMFC$
8. Z_{out}, PC_{in}, END

1.5 Esercizio 5 (CALL abs)

Esercizio 1.5
Descrivere le microistruzioni della seguente istruzione:

```
CALL (%EAX, %EBX)
```

Assunzioni:

- *L'indirizzo a cui saltare è %eax + %ebx (indirizzamento indiretto)*
- *Salto assoluto.*

F 1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
2. Z_{out}, PC_{in}, $WMFC$
3. MDR_{out}, IR_{in}

DE 4. ESP_{out}, $SELECT_4$, SUB, Z_{in}
5. Z_{out}, MAR_{in}, ESP_{in}
6. PC_{out}, MDR_{in}, $WRITE$
7. EAX_{out}, V_{in}, $WMFC$
8. EBX_{out}, $SELECT_V$, ADD, Z_{in}
9. Z_{out}, PC_{in}, END

1.6 Esercizio 6 (RET)

Esercizio 1.6
Descrivere le microistruzioni della seguente istruzione:

```
RET
```

(ritorno di una funzione).

F 1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
2. Z_{out}, PC_{in}, $WMFC$
3. MDR_{out}, IR_{in}

DE 4. ESP_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
5. Z_{out}, ESP_{in}, $WMFC$
6. MDR_{out}, PC_{in}, END

1.7 Esercizio 7 (JMP)

Esercizio 1.7
Descrivere le microistruzioni della seguente istruzione:

`JMP (%eax)`

(Assumo che il salto sia relativo al PC)

F	*1.*	PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
	2.	Z_{out}, PC_{in}, $WMFC$
	3.	MDR_{out}, IR_{in}
DE	*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

1.8 Esercizio 8 (JZ)

Esercizio 1.8
Descrivere le microistruzioni della seguente istruzione:

`JZ (%eax)`

(Assumo che il salto sia relativo al PC)

F	*1.*	PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
	2.	Z_{out}, PC_{in}, $WMFC$
	3.	MDR_{out}, IR_{in}
DE	*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

1.9 Esercizio 9 (Call abs)

Esercizio 1.9
Elencare e commentare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler:

`CALL (%eax, %ebx)`

dove:

- *si assume che l'indirizzo a cui saltare sia memorizzato nella locazione di memoria puntata dal valore %eax+%ebx (indirizzamento indiretto);*
- *si assume salto assoluto (salvo diversamente specificato);*

- *si assume l'utilizzo di architettura Intel dove lo stack cresce verso indirizzi di memoria inferiori rispetto ed %ESP punta alla cima occupata dallo stack.*

F 1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
Estraggo dal program counter l'indirizzo dell'istruzione da eseguire e lo metto nel MAR per ottenere l'istruzione. Successivamente metto imposto $SELECT_4$ che seleziona la costante 4 dal multiplexer per sommarla al program counter che si trova già nella ALU, questo equivale ad andare all'istruzione successiva, cioè PC + 1 word. Il risultato della somma viene salvato nel registro Z.

2. Z_{out}, PC_{in}, $WMFC$
Estraggo l'indirizzo dell'istruzione successiva dall'registro Z e lo inserisco nel program counter mentre aspetto che la funzione di lettura del MAR venga completata.

3. MDR_{out}, IR_{in}
Una volta che viene letto il dato all'indirizzo inserito nel MAR il risultato viene messo in MDR e successivamente trasferito nell'IR completando così il fetch dell'istruzione.

DE 4. ESP_{out}, $SELECT_4$, SUB, Z_{in}
Sottraggo 4 (1 word) dallo stack pointer in modo da creare spazio nello stack per inserire il program counter dell'istruzione successiva da eseguire.

5. Z_{out}, MAR_{in}, ESP_{in}
Estraggo il risultato della somma e lo inserisco all'interno del registro ESP e nel MAR in modo da poter scrivere il PC in quell'indirizzo.

6. PC_{out}, MDR_{in}, $WRITE$
Ottengo il program counter e lo inserisco nel MDR per poter richiamare la funzione WRITE che scrive il dato presente in MDR all'interno dell'indirizzo presente nel registro MAR. Fino a questo punto è stata eseguita una push del program counter nello stack.

7. EAX_{out}, V_{in}, $WMFC$
Metto il valore del registro EAX nel registro V per poter sommare. In questa istruzione richiamo la funzione WMFC per aspettare che la funzione di memoria venga eseguita.

8. EBX_{out}, $SELECT_V$, ADD, Z_{in}
Inserisco EBX nel bus e seleziono il registro V per la somma mettendo il risultato di EAX+EBX nel registro Z

9. Z_{out}, PC_{in}, END

Estraggo il risultato dal registro Z, che sarebbe l'indirizzo a cui effettuare il salto, e lo inserisco direttamente nel program counter visto che questo è stato specificato come salto assoluto.

1.10 Esercizio 10 (XCHG)

Esercizio 1.10
Elencare e commentare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler:

```
XCHG variabile, %eax
```

- *si assume che la variabile sia costituita da un indirizzo immediato direttamente codificato dentro l'istruzione (con il segnale IR_imm_field disponibile).*

F 1. PC_{out}, MAR_{in}, $READ$, $SELECT_4$, ADD, Z_{in}
Estraggo dal program counter l'indirizzo dell'istruzione da eseguire e lo metto nel MAR per ottenere l'istruzione. Successivamente metto imposto $SELECT_4$ che seleziona la costante 4 dal multiplexer per sommarla al program counter che si trova già nella ALU, questo equivale ad andare all'istruzione successiva, cioè PC + 1 word. Il risultato della somma viene salvato nel registro Z.

2. Z_{out}, PC_{in}, $WMFC$
Estraggo l'indirizzo dell'istruzione successiva dall'registro Z e lo inserisco nel program counter mentre aspetto che la funzione di lettura del MAR venga completata.

3. MDR_{out}, IR_{in}
Una volta che viene letto il dato all'indirizzo inserito nel MAR il risultato viene messo in MDR e successivamente trasferito nell'IR completando così il fetch dell'istruzione.

D 4. $IR\_imm\_field_{out}$, MAR_{in}, $READ$
Viene preso l'indirizzo della variabile e viene inserito nel MAR per poter leggere il valore contenuto.

5. EAX_{out}, $SELECT_0$, ADD, Z_{in}, $WMFC$
Il valore di EAX viene sommato a 0 per poter salvare il valore all'interno del registro Z della ALU, in modo da poter effettuare lo scambio. Infine si aspetta che la funzione di lettura venga completata.

E 6. MDR_{out}, EAX_{in}
Viene estratto il valore della variabile dal registro MDR e viene inserito nel registro EAX.

7. Z_{out}, MDR_{in}, $WRITE$
   Viene preso il vecchio valore di EAX e viene messo in MDR per poterlo scrivere all'indirizzo della variabile che si trova ancora nel MAR.
8. $WMFC$, END
   Si aspetta che la funzione di scrittura venga completata e viene interrotta l'istruzione.