Sistemi per la progettazione automatica

Anno Accademico 2008-9

Docente: Tiziano Villa
Laboratorio: Giuseppe Di Guglielmo

31 Marzo 2009

1. (Riscaldamento) I banchi di registri ("register files") sono una componente fondamentale di un'unita' esecutiva. Si considerino varie topologie di un banco di 4 (parole) registri di 4 bit ciascuno.

   - Un banco di registri a porta singola esclusiva puo' leggere o scrivere una sola parola per volta (ma non puo' leggere e scrivere contemporaneamente).
   - Un banco di registri a porta doppia esclusiva puo' leggere due parole per volta, o scrivere una parola per volta, ma non puo' fare entrambe le operazioni contemporaneamente.
   - Un banco di registri a porta doppia in lettura e singola in scrittura permette di leggere da due registri e scrivere in un registro contemporaneamente. Inoltre uno dei registri letti puo' coincidere con il registro in cui si scrive.

   Si progettino tutte e tre le precedenti specifiche di un banco di registri. In particolare si realizzino tutte e tre le specifiche fino al livello di porte logiche e bistabili, includendo tutti i segnali di selezione e controllo necessari. Si mostrino le forme d'onda dei principali segnali durante le operazioni di lettura e scrittura. Si commentino con chiarezza le scelte circuitali e i compromessi risorse/tempo.

2. Si consideri un processore con parole di 16 cifre binarie. La memoria principale $M[0 : 2^{16} - 1]$ contiene 2^{16} parole indirizzabili singolarmente (da x0000 a xFFFF). Il processore ha 8 registri visibili al programmatore di 16 cifre binarie ciascuno, $R0, R1, R2, R3, R4, R5, R6, R7$. Ci sono inoltre dei registri speciali come il contatore di programma, PC, il registro delle istruzioni, IR, il registro degl'indirizzi di memoria, MAR, e il registro dei dati di memoria, MBR.

   Il processore ha cinque modi d'indirizzamento: immediato, a registri, e tre modi d'indirizzamento della memoria (relativo al PC, indiretto, e registro-base + spiazzamento ("Base + offset")).

   Ci sono tre registri a una cifra binaria ciascuno per memorizzare i codici di condizione: N (negativo), Z (zero), P (positivo). Le istruzioni LD, LDI, LDR, LEA, ADD, AND, NOT scrivono il risultato in uno degli 8 registri generali. I codici di condizione sono assegnati in base al valore di tale risultato, considerato come un numero in complemento a due: N=1, Z=P=0 se il risultato e' negativo, Z=1, N=P=0 se il risultato e' zero, P=1, Z=N=0 se il risultato e' positivo. Le altre istruzioni lasciano invariati i codici di condizione.

   Ipotesi semplificatrice: si supponga che il periodo dell'orologio del processore superi il tempo di accesso alla memoria, cioe' che la memoria abbia tempo d'attesa nullo.

   Ogni istruzione e' rappresentata da una parola singola. Sintassi e semantica di ogni istruzione sono descritte di seguito. Il codice operativo di ogni istruzione e' il campo delle quattro cifre binarie piu' significative.

Ogni istruzione, tranne che sia diversamente specificato, include implicitamente nella sua semantica l'azione d'incremento di 1 del PC, cioe' l'esecuzione dell'istruzione prevede di compiere l'assegnamento $PC := PC + 1$ prima delle operazioni richieste dalla sua semantica.

La notazione $sext(imm)$ indica l'estensione dell'operando imm a 16 cifre binarie replicando la cifra piu' significativa tante volte quante sono necessarie.

ADD (Somma)

Istruzione	*Semantica*
`add` DR,SR1,SR2	$if\ (bit[5] = 0)\ DR := SR1 + SR2$
`add` DR,SR1,imm5	$else\ DR := SR1 + sext(imm5)$

```
F E D C | B A 9 | 8 7 6 | 5 | 4 3 | 2 1 0
0 0 0 1 |   DR  |  SR1  | 0 | 0 0 |  SR2

F E D C | B A 9 | 8 7 6 | 5 | 4 3 2 1 0
0 0 0 1 |   DR  |  SR1  | 1 |   imm5
```

Descrizione:
Se bit[5] e' 0, il secondo operando e' ottenuto da SR2. Se bit[5] e' 1, il secondo operando e' ottenuto dal campo imm5 esteso con il segno a 16 cifre binarie. In entrambi i casi, si sommano i due operandi e si memorizza il risultato in DR. Si assegnano i codici di condizione a seconda che il risultato sia negativo, zero o positivo.

AND (Congiunzione di ogni cifra binaria)

Istruzione	*Semantica*
`and` DR,SR1,SR2	$if\ (bit[5] = 0)\ DR := SR1\ and\ SR2$
`and` DR,SR1,imm5	$else\ DR := SR1\ and\ sext(imm5)$

```
F E D C | B A 9 | 8 7 6 | 5 | 4 3 | 2 1 0
0 1 0 1 |   DR  |  SR1  | 0 | 0 0 |  SR2

F E D C | B A 9 | 8 7 6 | 5 | 4 3 2 1 0
0 1 0 1 |   DR  |  SR1  | 1 |   imm5
```

Descrizione:
Se bit[5] e' 0, il secondo operando e' ottenuto da SR2. Se bit[5] e' 1, il secondo operando e' ottenuto dal campo imm5 esteso con il segno a 16 cifre binarie. In entrambi i casi, si calcola la congiunzione logica a cifra a cifra dei due operandi e si memorizza il risultato in DR. Si assegnano i codici di condizione a seconda che il risultato sia negativo, zero o positivo.

BR (Salto condizionato)

Istruzione	*Semantica*
`brn` label	$if\ ((n\ and\ N)\ or\ (z\ and\ Z)\ or\ (p\ and\ P))$
`brz` label	$PC := PC + sext(PC\text{offset9})$
`brp` label	
`brzp` label	
`brnp` label	
`brnz` label	
`brnzp` label	
`br` label	$PC := PC + sext(PC\text{offset9})$

```
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
0 0 0 0 | n z p |     PCoffset9
```

Descrizione:
Si confrontano i valori delle cifre binarie [11:9] con i valori dei registri che memorizzano i codici di condizione N, Z, P. Se il predicato risulta vero (perche' una delle congiunzioni risulta vera), il programma salta all'indirizzo ottenuto sommando al PC incrementato il valore *PCoffset9* con il segno esteso. Per esempio, *brzp label* significa un salto a *label* se l'ultimo risultato e' stato zero o positivo. Si noti che l'istruzione *br label* indica un salto incondizionato.

JMP/RET (Salto/Ritorno da procedura)

Istruzione	*Semantica*
`jmp` BaseR	$PC := BaseR$
`ret`	

```
JMP
F E D C | B A 9 | 8 7 6 | 5 4 3 2 1 0
1 1 0 0 | 0 0 0 | BaseR |   000000

RET
F E D C | B A 9 | 8 7 6 | 5 4 3 2 1 0
1 1 0 0 | 0 0 0 | 1 1 1 |   000000
```

Descrizione:
Si salta senza condizione all'indirizzo specificato dal contenuto del registro di base identificato da bit[8:6]. Per esempio, *jmp R2* significa *PC := R2*; *ret* significa *PC := R7*. L'istruzione *ret* e' un caso speciale dell'istruzione *jmp*, dove si carica il PC con il contenuto di R7, il quale ultimo contiene l'indirizzo di ritorno all'istruzione successiva a quella di chiamata a una procedura.

JSR/JSRR (Salto a procedura)

Istruzione	*Semantica*
	$R7 := PC$
`jsr` label	$if\ (bit[11] = 0)\ PC := BaseR$
`jsrr` BaseR	$else\ PC := PC\ + sext(PC\text{offset11})$

```
JSR
F E D C | B | A 9 8 7 6 5 4 3 2 1 0
0 1 0 0 | 1 |       PCoffset11

JSRR
F E D C | B | A 9 | 8 7 6 | 5 4 3 2 1 0
0 1 0 0 | 0 | 0 0 | BaseR |   000000
```

Descrizione:
Per prima cosa il PC incrementato e' salvato in R7 (per poter ritornare all'indirizzo successivo una volta terminata la chiamata). Poi si carica nel PC l'indirizzo della prima istruzione della procedura chiamata per saltare senza condizione a tale indirizzo. L'indirizzo della procedura chiamata si ottiene dal registro di base (se bit[11]=0), oppure si ottiene sommando il PC incrementato con il valore PCoffset11 esteso di segno (se bit[11]=1).

LD (Lettura da memoria)

Istruzione	*Semantica*
`ld` DR,label	$DR := mem[PC + sext(PC\text{offset9})]$

```
LD
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
0 0 1 0 |   DR  |     PCoffset9
```

Descrizione:
Si calcola un indirizzo sommando il PC incrementato con il valore PCoffset9 esteso di segno e si carica in DR il contenuto di tale indirizzo di memoria. Si assegnano i codici di condizione a seconda che il valore caricato sia negativo, zero o positivo.

LDI (Lettura indiretta da memoria)

Istruzione	*Semantica*
`ldi` DR,label	$DR := mem[mem[PC + sext(PC\text{offset9})]]$

```
LDI
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
1 0 1 0 |   DR  |     PCoffset9
```

Descrizione:
Si calcola un indirizzo sommando il PC incrementato con il valore PCoffset9 esteso di segno e poi prelevando da tale posizione di memoria il contenuto che e' l'indirizzo cercato. Si carica in DR il contenuto di tale ultimo indirizzo di memoria. Si assegnano i codici di condizione a seconda che il valore caricato sia negativo, zero o positivo.

LDR (Lettura da memoria con registro base)

Istruzione	*Semantica*
`ldr` DR,BaseR,offset6	$DR := mem[BaseR + sext(\text{offset6})]$

```
LDR
F E D C | B A 9 | 8 7 6 | 5 4 3 2 1 0
0 1 1 0 |   DR  | BaseR |   offset6
```

Descrizione:
Si calcola un indirizzo sommando il valore nel registro BaseR con il valore offset6 esteso di segno e si carica in DR il contenuto di tale indirizzo di memoria. Si assegnano i codici di condizione a seconda che il valore caricato sia negativo, zero o positivo.

LEA (Caricamento di un indirizzo)

Istruzione	*Semantica*
`lea` DR,label	$DR := PC + sext(PC\text{offset9})$

```
LEA
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
1 1 1 0 |   DR  |     PCoffset9
```

Descrizione:
Si calcola un indirizzo sommando il PC incrementato con il valore PCoffset9 esteso di segno e si carica in DR tale indirizzo. Si assegnano i codici di condizione a seconda che il valore caricato sia negativo, zero o positivo.

NOT (Negazione di ogni cifra binaria)

Istruzione	*Semantica*
`not` DR,SR	$DR := NOT(SR)$

```
F E D C | B A 9 | 8 7 6 | 5 | 4 3 2 1 0
1 0 0 1 |   DR  |  SR   | 1 | 1 1 1 1 1
```

Descrizione:
Si calcola il complemento di ogni cifra binaria di SR e si memorizza il risultato in DR. Si assegnano i codici di condizione a seconda che il risultato sia negativo, zero o positivo.

ST (Scrittura in memoria)

Istruzione	*Semantica*
`st` SR,label	$mem[PC + sext(PC\text{offset9})] := SR$

```
ST
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
0 0 1 1 |   SR  |     PCoffset9
```

Descrizione:
Il contenuto del registro SR e' memorizzato all'indirizzo di memoria ottenuto sommando il PC incrementato con il valore PCoffset9 esteso di segno.

STI (Scrittura indiretta in memoria)

Istruzione	*Semantica*
`sti` SR,label	$mem[mem[PC\ + sext(PC\text{offset9})]] := SR$

```
STI
F E D C | B A 9 | 8 7 6 5 4 3 2 1 0
1 0 1 1 |   SR  |     PCoffset9
```

Descrizione:
Si calcola un indirizzo sommando il PC incrementato con il valore PCoffset9 esteso di segno e poi prelevando da tale posizione di memoria il contenuto che e' l'indirizzo cercato. Si carica il contenuto di SR in tale ultimo indirizzo di memoria.

STR (Scrittura in memoria con registro base)

Istruzione	*Semantica*
`str` SR,BaseR,offset6	$mem[BaseR + sext(\text{offset6})] := SR$

```
STR
F E D C | B A 9 | 8 7 6 | 5 4 3 2 1 0
0 1 1 1 |   SR  | BaseR |   offset6
```

Descrizione:
Si calcola un indirizzo sommando il valore nel registro BaseR con il valore offset6 esteso di segno e si carica il contenuto di SR in tale indirizzo di memoria.

I codici operativi rimanenti sono riservati per usi futuri.

Si progetti un processore che esegue l'insieme d'istruzioni precedente.

I passi del progetto sono i seguenti:

(a) Si progetti l'unita' esecutiva. I suoi componenti principali sono: il banco dei registri a porta duale che permettono o due letture o una scrittura; l'unita' aritmetico-logica che permette addizione e sottrazione in complemento a due, confronto e operazioni logiche; il circuito traslatore che permette lo scorrimento a sinistra o a destra di una posizione.

(b) Si progetti il controllore. Si sviluppi un diagramma degli stati di Moore per il processore. Si indichino le operazioni di trasferimento tra registri associate agli stati.

3. Si progettino integralmente l'unita' esecutiva e il controllore fino ai moduli logici elementari, usando gli strumenti per la specifica, simulazione e sintesi del VHDL introdotti nel laboratorio.

4. Si programmino i seguenti algoritmi nel linguaggio macchina del processore e si eseguano con la realizzazione circuitale proposta. Se necessario, si puo' supporre che la costante 0 e la costante 1 siano memorizzate in locazioni prefissate della memoria principale. Inoltre, come gia' precisato, si assuma che la memoria risponda alle richieste in un solo ciclo di calcolo.

   (a) Programma n. 1

      i. Si scriva un programma nel linguaggio macchina di questo processore per realizzare la seguente procedura in C:

```
int gf(int x) {
   int f,h;
   f = x;
   do {
      f--;
      h = x;
      do {
         h = h -f;
         if (h == 0) return(f);
      } while (h > 0)
   } while (1)
}
```

      Supponendo che il programma si trovi in memoria a partire dall'indirizzo 0, s'indichino a fianco delle istruzioni i loro indirizzi di memoria.

      ii. Si simuli l'esecuzione del precedente programma per $x = 6318$ e si determini il tempo richiesto per calcolare il risultato.

   (b) Programma n. 2

      i. Si scriva un programma nel linguaggio macchina di questo processore per realizzare la procedura che calcola l'ennesimo numero di Fibonacci. Si accluda anche il codice in C della procedura. Supponendo che il programma si trovi in memoria a partire dall'indirizzo 0, s'indichino a fianco delle istruzioni i loro indirizzi di memoria.

      ii. Si simuli l'esecuzione del precedente programma per $n = 10$ e si determini il tempo richiesto per calcolare il risultato.

   (c) Programma n. 3

      i. Si scriva un programma nel linguaggio macchina di questo processore per realizzare la seguente procedura in C:

```
#include <stdio.h>

void Magic(int in);
int Even(int n);

int main() {
   Magic(10);
   return(0);
}

void Magic(int in) {
   if (in == 0)
```

```
      return;
   if (Even(in))
      printf("%i\n", in);
   Magic(in -1);
   if (!Even(in))
      printf("%i\n", in);
   return;
}

int Even(int n) {
   return((n % 2) == 0) ?  1 :  0;
}
```

Supponendo che il programma si trovi in memoria a partire dall'indirizzo 0, s'indichino a fianco delle istruzioni i loro indirizzi di memoria.

ii. Quale uscita produce tale programma ? Si mostri una traccia dettagliata dell'esecuzione sia del programma in C che del vostro programma in linguaggio macchina eseguito dal processore da voi progettato.