1 Lezione del 28-11-24

1.1 Descrizione in Verilog del ciclo fetch/execute

1.1.1 Fase di fetch

Abbiamo visto quali erano i formati di indirizzamento degli operandi delle istruzioni, e le sottoliste per le letture e scritture in memoria. Possiamo quindi descrivere in Verilog la fase di fetch del processore:

```
// fasi di fetch
fetch0 : begin
  ADDR <= IP;
  IP <= IP + 1;
  MJR <= fetch1;
  STAR <= readB;
end
fetch1 : begin
  OPCODE <= APP[0];
  STAR <= fetch2;
end
fetch2 : begin
  MJR <= (OPCODE[7:5] == F0) ? fetch4:
         (OPCODE[7:5] == F1) ? fetch4:
         (OPCODE[7:5] == F2) ? F2fetch0:
         (OPCODE[7:5] == F3) ? F3fetch0:
         (OPCODE[7:5] == F4) ? F4fetch0:
         (OPCODE[7:5] == F5) ? F5fetch0:
         (OPCODE[7:5] == F6) ? F6_7fetch0:
       /*(OPCODE[7:5] == F7)?*/F6_7fetch0;
  STAR <= fetch3;
end
fetch3 : begin
  STAR <= MJR;
end

[...]

// formati di fetch
F2fetch0 : begin
  ADDR <= DP;
  MJR <= F2fetch1;
  STAR <= readB;
end
F2fetch1 : begin
  SOURCE <= APP[0];
  STAR <= fetch4;
end

F3fetch0 : begin
  DEST_ADDR <= DP;
  STAR <= fetch4;
end

F4fetch0 : begin
  ADDR <= IP;
  IP = IP + 1;
  MJR <= F4fetch1;
  STAR <= readB;
end
```

```
F4fetch1 : begin
  SOURCE <= APP[0];
  STAR <= fetch4;
end

F5fetch0 : begin
  ADDR <= IP;
  IP = IP + 2;
  MJR <= F5fetch1;
  STAR <= readW;
end
F5fetch1 : begin
  ADDR <= {APP[1], APP[0]};
  MJR <= F5fetch2;
  STAR <= readB;
end
F5fetch2 : begin
  SOURCE <= APP[0];
  STAR <= fetch4;
end

F6_7fetch0 : begin
  ADDR <= IP;
  IP = IP + 2;
  MJR <= F6_7fetch1;
  STAR <= readW;
end
F6_7fetch1 : begin
  DEST_ADDR <= {APP[1], APP[0]};
  STAR <= fetch4;
end
```

Alla fine della fase di fetch saremo riusciti con successo a mettere:

- Il codice operativo dell'istruzione in OPCODE;
- L'operando immediato o in memoria dell'istruzione in SOURCE;
- L'operando destinatario in DEST_ADDR;
- IP sulla prossima istruzione da prelevare.

1.1.2 Fase di esecuzione

Nella fase di esecuzione, avremo quindi tutti gli operandi già inizializzati, e dovremo solo farli passare attraverso apposite reti combinatorie, o scegliere appositi stati di esecuzione del processore:

```
fetch4 : begin
  MJR <= first_exec_state(OPCODE);
  STAR <= fetch5;
end
fetch5 : begin
  STAR <= MJR;
end

[...]

function[STATE_SIZE - 1:0] first_exec_state;
  input[7:0] opcode;
```

```
13   first_exec_state = // istruzioni senza operandi
14                      (opcode == 8'B0000_0000) ? hlt:
15                      (opcode == 8'B0000_1001) ? nop:
16
17                      [...]
18
19                      /*     don't care     */    nvi;
20 endfunction
```

Notiamo che una trattazione più completa di quella fatta in questi appunti sulla struttura del calcolatore è fatta nella directory `verilog/11-24`, dove è disponibile un'implementazione Verilog di un semplice calcolatore, compreso processore e spazio di memoria. Il calcolatore è programmabile secondo l'instruction set riportato in `verilog/11-24/instruction_set.txt`, sfruttando l'assemblatore scritto in Python in `verilog/11-24/assembler/assemble.py`. Sono inoltre disponibili una testbench e un file di impostazione per il pachetto GTKWave che evidenzia il comportamento dei registri interni del processore e delle linee del bus.

Si nota che il calcolatore implementato ha un'architettura con indirizzi a 16 anzichè 24 bit, in quanto dump di memoria da soli 16 KB sono più gestibili.

1.2 Interfacce

Veniamo adesso alla descrizione di interfacce che completano il calcolatore, cioè gli permettono di comunicare col mondo esterno. Le interfacce possono essere di due tipi principali:

- **Parallele**: un byte alla volta (quindi più bit *in parallelo*);
- **Seriali**: un bit alla volta.

Vedremo poi anche le interfacce per la conversione da **analogico a digitale** e viceversa, che trasformano da tensioni a gruppi di bit.

I collegamenti lato bus delle interfacce, come avevamo anticipato sono sempre uguali, mentre cambiano sul lato dispositivo.

1.2.1 Visione funzionale di un interfaccia

La visione funzionale di un interfaccia è quella dal punto di vista di chi deve interagirci, cioè come un insieme di registri su cui opererà il **processore**:

- **Receive Buffer Register (RBR)**: registro dove si vanno a *leggere* informazioni **dall'interfaccia**;
- **Transmit Buffer Register (TBR)**: registro dove si vanno a *scrivere* informazioni **all'interfaccia**.

1.2.2 Sincronizzazione processore-dispositivi

Eseguendo un programma che contiene sequenze di istruzioni `IN` o `OUT`, il processore non può sapere se fra una `IN` e l'altra (o fra una `OUT` e l'altra) il dispositivo ha prodotto nuovi dati (o se ha processato quelli inviati). Dovremo quindi implementare un doppio handshake, sia sul lato processore (*handshake "software"*) che sul lato hardware (*handshake "hardware"*).

Dotiamo quindi le interfacce di registri di stato:

- **Receive Status Register (RSR)**: contiene un bit di interesse, il flag **FI** di **ingresso pieno**;
- **Transmit Status Register (TSR)**: contiene un altro bit di interesse, il flag **FO** di **uscita vuota**.

I due flag FI e FO vengono controllati dall'interfaccia, e quindi impostati a 1 o a 0 quando questa rileva le condizioni opportune.

1.2.3 Ingresso dati a controllo di programma

Vediamo quindi un ciclo di ingresso dati. Si parte con FI a 0. Quando il dispositivo gestito dall'interfaccia scrive in RBR, l'interfaccia mette FI a 1. Questo segnala al processore che c'è un nuovo dato. A questo punto, quando il processore accede in lettura al registro RBR, l'interfaccia riporta FI a 0.

Notiamo che su due letture consecutive il processore è in **attesa attiva** finché non FI non si alza nuovamente. Esistono altri metodi di accesso in memoria che non richiedono l'attesa attiva da parte del processore, fra cui il meccanismo degli *interrupt* e il *DMA (Direct Memory Access))*.

1.2.4 Uscita dati a controllo di programma

Vediamo adesso un ciclo di uscita dati. Il flag FO parte a 0. L'interfaccia lo mette a 0 quando il processore scrive in TBR, per segnalare che il dispositivo non ha ancora elaborato. Quando il dispositivo accede a TBR per la lettura, FO torna a 0.