1 Lezione del 27-11-24

1.1 Letture e scritture nello spazio di memoria

Durante la fase di fetch, abbiamo eseguito solo **letture** in memoria. Vediamo adesso che nella fase **execute** abbiamo bisogno di effettuare sia letture che scritture. Vediamo come si effettuano a livello di μ-istruzioni queste letture e scritture, in maniera compatibile con le temporizzazioni già definite sulle memorie.

1.1.1 Lettura

Innanzitutto ripassiamo le temporizzazioni nel caso della lettura:

- A indirizzi stabili arriva il comando `/mr`;
- `/s` arriva con ritardo;
- A `/mr` e `/s` bassi si effettua la lettura, cioè le tri-state vanno in conduzione;
- I multiplexer alle uscite vanno a regime dopo gli indirizzi, da qui in puoi i dati sono buoni e si prelevano;
- Quando `/mr` torna a 1 i dati tornano ad alta impedenza, da lì in poi `/s` e gli indirizzi possono tornare instabili.

Definiamo allora un μ-programma per la lettura in memoria:

```
1 mem_r0: begin A23_A0 <= address; DIR <= 0; MR <= 0; STAR <= mem_r1; end
2 mem_r1: begin STAR <= mem_r2; end // stato di wait, da qui in poi omesso
3 mem_r2: begin cpu_register <= d7_d0; MR <= 1; ...; end
```

Notiamo che allo stato R2 si possono cambiare tutte le linee (indirizzo, ecc...) tranne che la DIR, in quanto impostando solo a quel punto MR non si è sicuri che la RAM risponda in tempo.

1.1.2 Scrittura

Ricordiamo che la scrittura è distruttiva. Ricordiamo quindi le temporizzazioni:

- Si abbassa `/s` e ci si assicura che gli indirizzi siano stabili;
- Si abbassa `/mr`;
- I dati dovranno essere corretti fino al fronte di salita di `/mr`.

Definiamo un'altro μ-programma, stavolta per la scrittura in memoria:

```
1 mem_w0: begin A23_A0 <= address; D7_D0 <= new_byte; DIR <= 1; STAR <=
      mem_w1; end
2 mem_w1: begin MW <= 0; STAR <= mem_w2; end
3 mem_w2: begin MW <= 1; STAR <= mem_w3; end
4 mem_w3: begin DIR <= 0; ...; end
```

Notiamo di non poter abbassare DIR o gli indirizzi fino allo stato W3, in quanto non si può essere sicuri che a quel punto la RAM abbia finito di scrivere.

1.2 Letture e scritture nello spazio di I/O

Le letture e le scritture nello spazio di I/O sono diverse, in quanto qui **la lettura è distruttiva**. Inoltre, dobbiamo ricordarci di operare sui registri IOR e IOW anzichè MR e MW.

1.2.1 Lettura

Scriviamo quindi un μ-programma per la lettura nello spazio di I/O dove teniamo conto di dover abbassare IOR **dopo** che gli indirizzi sono stabili, in maniera simile alla lettura:

```
io_r0: begin A23_A0 <= address; DIR <= 0; STAR <= io_r1; end
io_r1: begin IOR <= 0; STAR <= io_r2; end
io_r2: begin STAR <= io_r3; end
io_r3: begin cpu_register <= d7_d0; IOR <= 1; ...; end
```

1.2.2 Scrittura

Ridefiniamo quindi il μ-programma di scrittura:

```
io_w0: begin A23_A0 <= address; D7_D0 <= new_byte; DIR <= 1; STAR <= io_w1
    ; end
io_w1: begin IOW <= 0; STAR <= io_w2; end
io_w2: begin IOW <= 1; STAR <= io_w3; end
io_w3: begin DIR <= 0; ...; end
```

Notiamo che, in questo caso, la scrittura si fa sul fronte di discesa anziché di salita, e quindi l'assegnamento di `D7_D0` al nuovo byte va fatto esclusivamente in W0, e non in W1 com'era possibile nella scrittura nello spazio di memoria.

metti gli address giusti, qui sarebbe 'H00 + offset

1.3 Accessi multipli in memoria

Il processore potrebbe fare accessi non solo ad un byte, ma 2 byte (per operandi su 16 bit) o 3 byte (per indirizzi). Occasionalmente dovremo leggere anche 4 byte, ma questo non è considerato nel corso.

Per fare letture su locazioni multiple, si usano μ-sottoprogrammi di lettura/scrittura modulari. Utilizzeremo:

- Il registro **MJR** per contenere il μ-indirizzo di ritorno;
- Il registro **NUMLOC** come contatore del numero di byte da leggere/scrivere;
- Il registro A23_A0 per contenere l'indirizzo del primo byte da leggere/scrivere;
- I registri APP0, ..., APP3 per contenere i byte letti/da scrivere.

1.3.1 Lettura

Vediamo quindi il μ-programma principale di lettura:

```
s_x: begin ... A23_A0 <= address; MJR <= s_x+1; STAR <= subprogram; end
s_x+1: begin ... /* qui si usa APP0 */ end
```

Definiamo 4 μ-sottoprogrammi:

- readB: legge 1 byte;

- readW: legge 2 byte;
- readM: legge 3 byte;
- readL: legge 4 byte.

I parametri di ingresso saranno l'indirizzo in memoria della prima locazione e la DIR impostata a 0, i parametri di uscita i registri APP da 0 a 3 (che conterranno i byte letti).

Vediamo quindi i μ-sottoprogrammi di lettura:

```
readB: begin MR <= 0; NUMLOC <= 1; STAR <= read0; end;
readW: begin MR <= 0; NUMLOC <= 2; STAR <= read0; end;
readM: begin MR <= 0; NUMLOC <= 3; STAR <= read0; end;
readL: begin MR <= 0; NUMLOC <= 4; STAR <= read0; end;

read0: begin APP0 <= d7_d0; A23_A0 <= A23_A0 + 1; NUMLOC <= NUMLOC - 1;
    STAR <= ( NUMLOC == 1 ) ? read4 : read1; end
read1: begin APP1 <= d7_d0; A23_A0 <= A23_A0 + 1; NUMLOC <= NUMLOC - 1;
    STAR <= ( NUMLOC == 1 ) ? read4 : read2; end
read2: begin APP2 <= d7_d0; A23_A0 <= A23_A0 + 1; NUMLOC <= NUMLOC - 1;
    STAR <= ( NUMLOC == 1 ) ? read4 : read3; end
read3: begin APP3 <= d7_d0; A23_A0 <= A23_A0 + 1; STAR <= read4; end
read4: begin MR <= 1; STAR <= MJR; end
```

non sono sicuro, comunque commentali

1.3.2 Scrittura

Vediamo il μ-programma principale di scrittura:

```
s: begin ... APP1 <= datum(15:8); APP0 <= datum(7:0); A23_A= <= address;
    MJR = s_x+1; STAR <= subprogram; end
```

E definiamo i 4 μ-sottoprogrammi:

- writeB: scrive 1 byte;
- writeW: scrive 2 byte;
- writeM: scrive 3 byte;
- writeL: scrive 4 byte.

I parametri di ingresso saranno l'indirizzo in memoria della prima locazione, la DIR impostata a 0, e i registri APP da 0 a 3 (che conterranno i byte da scrivere).

Implementiamo i μ-sottoprogrammi come:

```
writeB: begin D7_D0 <= APP0; DIR <= 1; NUMLOC <= 1; STAR <= write0; end
writeW: begin D7_D0 <= APP0; DIR <= 2; NUMLOC <= 1; STAR <= write0; end
writeM: begin D7_D0 <= APP0; DIR <= 3; NUMLOC <= 1; STAR <= write0; end
writeL: begin D7_D0 <= APP0; DIR <= 4; NUMLOC <= 1; STAR <= write0; end

write0: begin MW <= 0; STAR <= write1; end;
write1: begin MW <= 1; STAR <= ( NUMLOC == 1) ? write11 : write2; end
```

finisci qua sopra

1.4 Descrizione in Verilog del processore

Iniziamo quindi a definire il nostro processore col linguaggio Verilog. qui per l'amor di cristo riscrivi per benino e metti nella cartella /verilog che così è drammatico