

1 Lezione del 28-11-24

1.1 Descrizione in Verilog del ciclo fetch/execute

1.1.1 Fase di fetch

tutto verilog

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.

altro verilog

1.1.3 Formato F1

Abbiamo visto come un'eccezione tra le istruzioni è rappresentata da quelle in formato F1, in quanto il "fetch" effettivo dei loro operandi sorgenti e destinatari va fatto in fase di esecuzione. Queste, ricordiamo, sono le istruzioni di I/O (operandi su indirizzi a 16 bit) e MOV sui registri da 24 bit.

indovina? altro verilog

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.