Architettura degli elaboratori

Appunti ordinati

Lezione 12 - 28/04/2025

Tipi di memoria: RAM

- SRAM: RAM statiche (flip-flop di tipo D), estremamente veloci, utilizzate per realizzare la cache
- DRAM: RAM dinamiche(transistor con condensatore), vanno rinfrescati, offrono grandi capacità ma più lente
    - SDRAM-DDR: è quella più moderna

Macchine a stati finiti

macchina(o automa):

- dispositivo automatico in grado di interagire con l'ambiente esterno;
- a fronte di uno stimolo in ingresso(input), esibisce un comportamento in uscita(output) che dipende anche da informazioni memorizzate in elementi interni(stati)

Ci limitiamo a macchine con *memoria finita.*

Una macchina a stati finiti:

- legge un simbolo in ingresso(che appartiene ad un insieme finito A)
- produce un simbolo in uscita(che appartiene ad un insieme finito B)

- cambia il proprio stato interno

AD esempio:

vogliamo implementare un distributore automatico di biglietti:

accetta 2 tipi di monete, monete grandi(MG) e monete piccole(MP)

un biglietto viene emesso solo quando vengono ricevute una moneta per tipo.

non conta l'ordine di immissione delle monete

le monete non adeguate vengono restituite

(Non basta l'input, abbiamo bisogno della concezione di stato della memoria)

L'insieme finito dei simboli di ingresso è $A = \{MP, MG\}$

L'insieme finito dei simboli in uscita è $B = \{ancora, restituisci, emetti\}$

L'insieme finito degli stati Q è formato da:

- q0: non è stata inserita alcuna moneta
- q1: è stata inserita una MP
- q2: è stata inserita una MG

Quando ho inserito entrambe le monete il biglietto viene stampato tornando allo stato q0.

per rappresentare A(2 elementi) mi serve 1 bit

per rappresentare B(3 elementi) mi servono 2 bit

per rappresentare Q(3 stati) mi servono 2 bit

Ogni flip-flop rappresenta 1 bit

Definendo simboli di input/output e stati posso creare una tabella.

Una macchina(automa)a stati finiti M è composta da una quintupla

$$M =< A, B, Q, o, s >$$

- A: insieme finito di simboli di ingresso
- B: insieme finito di simboli di uscita
- Q: insieme finito di simboli di stato
- o: AxQ→B funzione di uscita
- s: AxQ→Q funzione di cambiamento di stato

Una eventuale “tabella di verità” deve avere come colonne il numero di input, stato corrente, stato futuro, e di output

Abbiamo 3 colonne(1 input e 2 stato) quindi avrò $2^3 = 8$ righe

Analisi di reti sequenziali sincrone

Fare l’analisi vuol dire partire da un circuito e ottenerne la tabella di verità

Esempio delle slides:

Quanti input ho? 2→4 bit

quanti bit in uscita ho? 1

il nuovo stato viene deciso dai 2 circuiti combinatori or ed and.

Dal circuito devo capire i punti in cui “estrarre” le espressioni booleane, sono gli ingressi dei flip-flop:

Otterrò

$$D_E = \bar{A}$$
$$D_F = E(A + B)F$$
$$C = E$$

Le prime due(D) sono input, la terza è output.

Abbiamo 2 colonne per gli input(A,B), 2 per gli stati precedenti(E,F), 2 per gli stati futuri(D_E,D_F), e 1 per le uscite(C).

Così facendo mi basta calcolare le espressioni booleane precedenti per ogni input/stato per costruirmi la tabella di verità(ho fatto l'analisi).

La sintesi consiste nell'opposto(partendo dalla tabella costruire il circuito)

Sintesi di reti sequenziali Sincrone

Esempio:

- voglio costruire una rete sequenziale che riconosca la presenza di una certa sequenza di bit fissata, anche se inclusa in una sequenza più lunga
- vogliamo riconoscere la sequenza 1001

se sono in X e inserisco 0 devo riconoscere 1001, e l'output vale 0

se inserisco 1 mi sposto in y, output è ancora 0, devo riconoscere 001

se inserisco 1 resto in y, perchè la sequenza non è riconosciuta ma il secondo bit può essere l'inizio della sequenza.

se sono in Y e inserisco 0 e mi sposto in W ma emetto ancora 0

a questo punto ho inserito 10, se inserisco 1 torno a Y

se inserisco 0 sono a 3 bit inseriti, quindi mi sposto in Z ed emetto 0

se inserisco 0 a questo punto torno a X e devo ricominciare da capo

se inserisco 1 ho completato la mia sequenza e avrò 1 in output, vuol dire che l'ultimo bit può essere l'inizio della nuova sequenza, quindi mi sposto in Y

ora trovo che ho 4 stati: 2 bit → 2 flip-flop;

ho 2 input(0,1)→ 1 bit

ho 2 output(0,1)→1 bit

ora diamo una configurazione ai 4 stati(assegnamo valore in bit a ciascuno stato, ad esempio X=00...)

La mia tabella avrà forma

A E F | B D_E D_F | B

Dopo aver costruito la tabella devo trovare l'equazione di stati futuri e output

(RICORDO: prendo gli 1 della colonna di cui voglio calcolare l'espressione booleana e metto in AND gli input della riga(negato quando è 0, non negato quando è 1, poi li metto in OR con gli altri 1 della colonna)

D_E → prendo il primo 1 e trovo $\bar{E}F\bar{A} + E\bar{F}\bar{A} = \bar{A}(\bar{E}F + E\bar{F}) = \bar{A}(E\,x + F)$

$D_F = \bar{E}\bar{F}A + \bar{E}FA + E\bar{F}A...$

B ha solo un 1 → EFA

Una tipica domanda d'esame è: dato un circuito in figura e dato valore agli stati di partenza e all'input(non cambia), quanto vale l'output dopo 3 cicli di clock? Devo vedere la tabella(se non c'è la creo io).

Processore RISC-V

Voglio un processore che implementa;

- add
- sub
- lw
- sw
- beq

Inizialmente costruiremo i registri e la ALU collegandoli, poi aggiungeremo la Control unit

Data Path RISC-V semplificato

dobbiamo definire il percorso dei dati che ci servono

Iniziamo dalla parte di fetch:

- Il PC fornire un indirizzo dalla memoria delle istruzioni(in memoria l'area che si chiama Testo)

Decode:

- Si capisce il tipo d'istruzione

Execute:

- eseguire il calcolo

Se ho un istruzione di tipo R devo salvare in un indirizzo li risultato

Se ho un salto posso dover aggiungere un valore diverso da 4 al PC

Lezione 13 - 29/04/2025

Metodologia di temporizzazione

- il segnale di clock determina quando gli elementi di stato scrivono la loro memoria interna
- gli ingressi a un elemento di stato devono raggiungere un valore stabile prima che il fronte attivo del clock provochi l'aggiornamento dello stato
- Supporremo che tutti gli elementi di stato, comprese le memorie, siano sensibili al fronte di salita
- La metodologia sensibile ai fronti permette di leggere e scrivere un elemento di stato nello stesso ciclo di clock
- il ciclo di clock deve avere una durata sufficiente a garantire che gli ingressi siano stabili

Per caricare una istruzione per ciclo di clock usiamo il PC, una ALU che esegue la add e la memoria delle istruzioni(dove ho l'indirizzo della istruzione da caricare), a questo punto l'output della memoria istruzioni sarà l'istruzione(esempio `add x5,x5,x5`)

Lo stesso output del PC diventa sia l'indirizzo dell'istruzione che l'input della ALU (che effettuerà la add) e il risultato sarà salvato di nuovo nel PC.

L'accesso alla memoria istruzioni e l'addizionatore computano CONTEMPORANEAMENTE, ci sarà un piccolo ritardo al massimo.

datapath per le istruzioni di tipo R

- ci servono altri due circuiti digitali
- Register file e ALU

In aggiunta al circuito di prima abbiamo che l'output della memoria istruzioni(l'istruzione), questi 32 bit vengono suddivisi in base alla convenzione di tipo R, userò quindi 5 bit per ingresso dei registri.

dopo un certo ritardo in output avrò i due registri operandi, selezionati grazie a dei multiplexer.

Questi registri operandi entrano nella ALU che esegue l'istruzione, l'output della ALU si ripropone in ingresso al resgister file per essere salvato nel registro indicato dall'istruzione.

Il valore del risultato dell'operazione verrà salvato nel suo registro al prossimo fronte di salita.

Istruzioni Load(tipo I) e Store(tipo S)

- ci servono altri due circuiti digitali
- Memoria dati e unità di estensione del segno

La memoria dati ha due ingressi che sono MemWrite e MemRead che permettono di capire se l'istruzione deve scrivere o leggere in memoria.

I due input(32 bit a testa) sono l'indirizzo del dato e il dato scritto, l'output(se c'è) è il valore letto in memoria.

Il registro di lettura 2 non ci interessa per questa istruzione.

L'unita di estensione del segno prende i 12 bit dell immediato dell'istruzione e estrae il più significativo, facendone l'estensione del segno.

Anche dato letto 2 ad esempio avrà un'output, ma non ha senso, non viene utilizzato per la load

Nella store userò i 2 registri di lettura, , il primo dato letto sarà l'indirizzo sommato con l'immediato(offset) e il secondo il valore dal dato, al fronte di salita successivo verrà salvato il valore nell'indirizzo dato.

Istruzione di salto beq(Tipo SB)

Il genera cost fa prima l'estensione del segno, poi lo shift di 1 bit a sx, ora il valore ottenuto viene sommato al PC per ottenere l'indirizzo del salto.

La ALU fa la sottrazione tra i registri per vedere se i registri sono uguali, se lo sono esegue il salto, altrimenti no.

Abbiamo prodotto data path per istruzioni di diverso tipo, ora li dobbiamo unire

Per mettere insieme tipo R e tipo S uso dei multiplexer, per scegliere tra gli input che mi servono nei punti dove le varie istruzioni richiedono dati diversi:

nel caso delle istruzioni R e S serve un multiplexer prima dell'alu per sapere se voglio scegliere tra dato letto 2 oppure immediato per la store.

serve un altro multiplexer in output per saperre se mi serve l'operazione della ALU oppure se mi serve un indirizzo dove salvare.

I vari segnali di controllo sono azionati dall'unità di controllo.

Al genera costante ricevo in input tutti e 32 i bit anche se gli immediati sono composti tutti da 12 bit perchè questi sono in ordine diverso, quindi mi serve sapere il tipo dell'istruzione(codop) per sapere quali bit prendere.

Unità di controllo

Avrà come input l’istruzione corrente da decodificare, e in output i valori dei bit di controllo visti prima.

Quindi fare un circuito che fa l’unità di controllo è costruire una tavola di verità.

L’idea è di creare l’unità di controllo, dalla quale derivano 2 bit che vanno ad un’altra unità di controllo(ALUOp) che poi comanda la ALU.

- ALUOp=00→somma per istruzioni di load e stor
- ALUOp=01→

Viene fatto questo split per ridurre le dimensioni dell’unità di controllo principale, riducendo la latenza dell’unità di controllo.