Università degli studi di Bologna

Anno Accademico 2000/2001

Corso di Architettura degli elaboratori

Esercitazione di Reti Logiche

Realizzazione completa di un'architettura di processore attraverso strumenti di emulazione circuitale

Studenti appartenenti al gruppo di studio:

- Mazzucco Michele 127130
- Pegoraro Alessandro 123121
- Pernici Andrea 121737
- Valenti Roberto 123121

Docente:
Prof. Renzo Davoli
Supporto all'esercitazione:
Dott. Luciano Bononi

Presentazione della rete circuitale

Scopo di questa esercitazione è la realizzazione completa di un'architettura di processore attraverso strumenti (software) di emulazione circuitale. Come suggerito dalle specifiche del progetto, abbiamo analizzato attentamente il comportamento della rete contenuta nel file-esempio di Retro (cpu2.toy) ed interpretato il significato dei valori contenuti nella Ram, quindi abbiamo proceduto ri-adattando l'architettura alle nostre esigenze, con il seguente risultato:

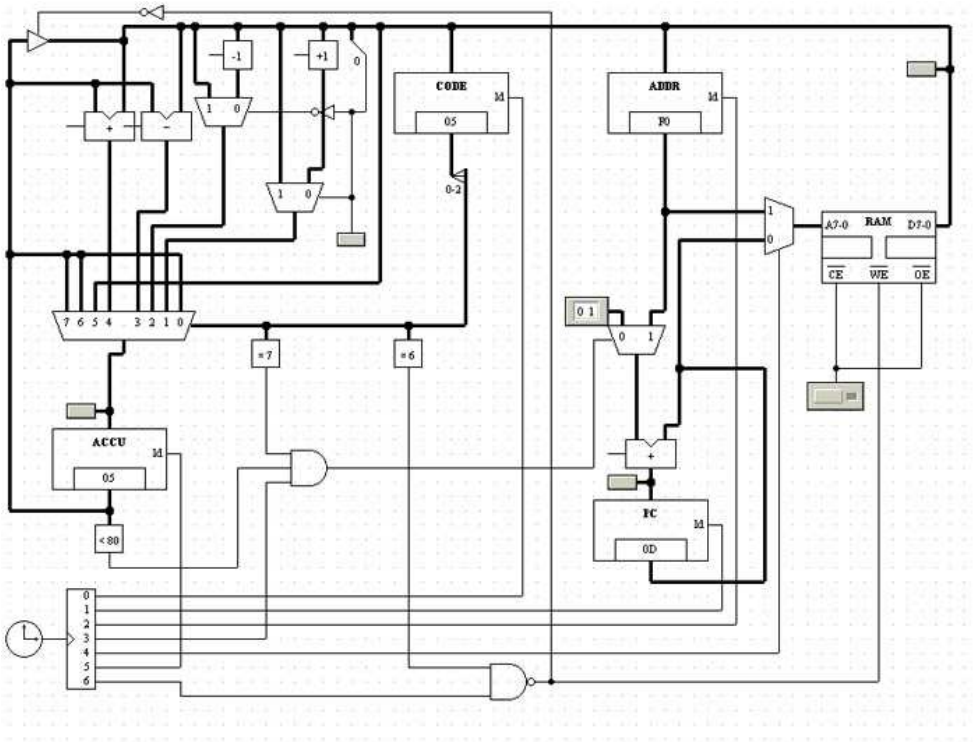

Figura 1

Come si può notare, l'architettura prodotta è molto simile all'esempio dato, ma da questo si discosta per alcuni tratti caratteristici:

- ☞ l'ALU è stata riadattata per implementare correttamente le due funzioni richieste "ODD" ed "EVEN", che agiscono sulla parità dell'input. Infatti il primo bit viene scorporato dagli altri e se risulta essere 1, il dato è dispari e l'ALU decrementa il valore di uno; viceversa, se il primo bit è 0, l'input è pari e al numero sarà sommata un'unità. Due multiplexer a due linee

in ingresso ed un bit di controllo svolgono l'incremento (decremento), e sono controllati da uno wire splitter che "comunica" loro la parità;

- ☞ il valore propagato dalla ALU entra nel registro Accumulatore, quindi subito confrontato con il valore 0x80 (10000000 in binario) : se il valore è minore di tale guardia, si ha l'abilitazione parziale del salto, ottenuto infatti solo quando il codice dell'operazione è "7" e il generatore di impulsi attiva la quarta linea. Quest'ultimo evento si ripete ad ogni ciclo di clock, cioè il microprogramma è definito attraverso la temporizzazione ciclica delle varie componenti. Questo si traduce in una limitazione sui possibili utilizzi della CPU stessa, in quanto è definito a priori il datapath dal programmatore;
- ☞ il contenuto della memoria è stato ovviamente modificato, per far sì che il processore soddisfi le richieste di ordinamento array;
- ☞ il valore della guardia al di sotto del registro ACCUMULATORE che nel file di esempio (cpu2.toy) era uguale a "0", è stato sostituito con "<80", in quanto tale valore in complemento a due è pari a "0";
- ☞ Ordina anche numeri negativi alla spesa di un programma più lungo , in quanto il flag di "avvenuto scambio" viene caricato sempre dalla medesima posizione.

La CPU prodotta, una volta caricato il file "bubblesort.mem" ordina un array di 5 elementi contenuti nell'ultima riga della RAM, vale a dire negli indirizzi da f0 a f4, gestendo anche valori negativi (considerati in complemento a due). L'area di codice occupa quindi le prime 6 "righe" (più precisamente da 00 a 5c).

Funzionamento

Quando viene attivata la CPU la RAM riceve immediatamente il segnale di lettura puntando alla prima cella di memoria e propagando il contenuto di tale posizione. In tal modo si decide qual è l'operazione da compiere, quindi si attiva il clock, al quale è associato un generatore capace di produrre 8 diversi impulsi secondo la temporizzazione definita; questo produce l'evoluzione del microprogramma (come accennato in precedenza), secondo lo schema di un'architettura CISC. Abbiamo programmato il pattern generator nel modo seguente.

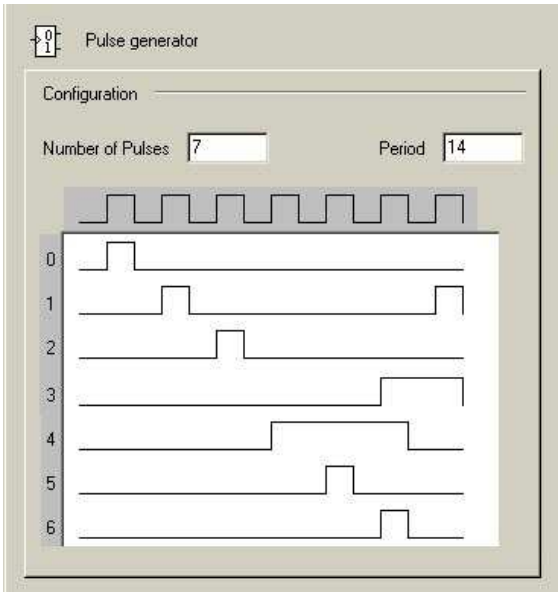

Figura 2

- ☞ al primo (0) è associata l'attivazione del CODE REGISTER , che contiene il codice dell'istruzione da eseguire; i primi 3 bit vengono immessi come linea di controllo nel multiplexer dell'ALU: rappresentano insomma l' "opcode" dell'istruzione;
- ☞ al secondo (1) corrisponde il segnale per l'attivazione del program counter (PC), che causa l'incremento di quest'ultimo di un'unità (o di un offset, nel caso in cui si debba saltare).
- ☞ al terzo (2) è collegato l'ADDRESS REGISTER, destinato a contenere un valore che a seconda dell'istruzione da eseguire rappresenta un indirizzo di memoria oppure un offset;
- ☞ il quarto segnale (3) è collegato ad una porta and a tre ingressi; quest'ultima da l'abilitazione al salto attivato effettivamente solo se anche gli altri due ingressi della porta lo consentono: l'opcode deve essere "7" ed il valore contenuto nel registro ACCUMULATORE deve risultare essere positivo (istruzione if (ACC >= 0) then PC :=PC + Offset);
- ☞ al quinto segnale (4) decide qualora la Ram debba venir indirizzata dal PC oppure dall'Address Register: nel secondo caso il PC risulta "escluso". In questo modo l'indirizzamento della RAM avviene direttamente tramite ADD. Register e questo permette, per esempio, di memorizzare un dato dall'Accumulatore ad un indirizzo specificato;
- ☞ il sesto impulso (5) controlla il registro ACCUMULATORE, attivandolo per immettere in quest'ultimo il dato elaborato

dalla ALU;

- ☞ il settimo segnale (6) abilita tramite una porta nand a due ingressi la scrittura sulla RAM: questa può avvenire comunque solo se anche l'altro input della porta nand vale 1, cioè se l'op-code è pari a "06".

Il ciclo di clock

Come già detto in precedenza, il microprogramma non è ottimizzato, in quanto ad ogni ciclo di clock (14 periodi) vengono eseguiti tutti i passi sufficienti a poter compiere qualsiasi operazione prevista sotto forma di hardware. Tutto ciò comporta evidentemente un vistoso decadimento delle prestazioni della CPU: per esempio, se si ha l'istruzione di caricamento, vengono inutilmente attivate anche le porte necessarie al salto ,o quelle per la scrittura in memoria, ecc. .
I primi 6 periodi del ciclo sono occupati dalla definizione dell'istruzione da effettuare: qualunque essa sia, viene caricato il primo dei due dati nel CODE REGISTER, quindi si ha l'incremento del program counter (quarto periodo del ciclo, vedi anche figura 2); infine il secondo valore viene immesso nell'ADDRESS REGISTER: da tutto ciò si può concludere che i dati nella RAM devono essere programmati a "coppie", cioè sapendo a priori che il primo valore rappresenta un codice operativo, mentre il secondo un indirizzo. A questo punto la CPU è pronta ad eseguire qualsiasi istruzione, in quanto i registri sono caricati in maniera opportuna e saranno proprio tali valori a determinare il comportamento dell'automa.
A partire dall'ottavo periodo, terminata la fase di fetch, inizia l'indirizzamento della memoria sulla base dell'indirizzo contenuto nell'ADDRESS per una durata complessiva di 5 periodi. Quindi, a seconda a seconda dell'operazione (op-code 5-6-7), si ottiene rispettivamente il caricamento di un dato contenuto nell'indirizzo specificato, il salvataggio all'interno della cella specificata, oppure il salto di un offset pari a tale valore.
Infine (periodo 14) si ha l'aggiornamento del program counter al fine di farlo puntare al codice operativo successivo.

Istruzioni Supportate

Istruzione	Opcode	Commento
ACC := ACC	00	Questa istruzione non provoca alcun cambiamento nello stato del sistema
ACC := mem(Address)	05	Il registro ACC viene caricato con il valore (Byte) contenuto alla locazione di RAM di indirizzo Address. Il tipo di indirizzamento è immediato, e non è possibile alcun indirizzamento indiretto
ACC := ACC - mem(Address)	03	Il registro ACC assume il valore ottenuto dal valore attuale al quale si sottrae il valore contenuto in RAM all'indirizzo Address (indirizzo immediato). Non vengono gestiti i casi di overflow (negativo)
ACC := ACC + mem(Address)	04	Il registro ACC viene ad assumere il valore ottenuto dal valore attuale al quale si somma il valore contenuto in RAM all'indirizzo Address (indirizzo immediato). Non vengono gestiti i casi di overflow (positivo)
ACC := ODD	01	Se il valore preso in esame è pari, questo comando causa l'incremento di un'unità del valore in ACC
ACC := EVEN	02	Questa istruzione causa il decremento di un'unità del valore in ACC solamente se il valore iniziale risulta essere dispari
mem(Address) := ACC	06	Con questo comando si ha l'abilitazione a scrivere in RAM alla locazione Address (indirizzo immediato) il valore contenuto nel registro ACC
if (ACC >= 0) then PC :=PC + Offset	07	Questa istruzione causa l'aggiornamento condizionale (se ACC >= 0) del program counter (PC), eventualmente sommando il valore immediato Offset (8 bit in complemento a due).

L'Arithmetic Logic Unit

L'ALU è stata in parte riprogettata per permettere di eseguire alcune operazioni ("odd" ed "even"), ed è sostanzialmente un multiplexer ad 8 ingressi (collegati alle varie porte che materialmente computano) e 3 bit di controllo. Gli ingressi sono i seguenti:

0) il dato che entra da questo bus non subisce alcuna modifica;
1) questo ingresso calcola se il valore in ingresso è pari: se sì lo incrementa di un'unità;
2) questo ingresso calcola se il valore in ingresso è dispari: se sì lo decrementa di un'unità;
3) la quarta linea di input calcola la differenza fra il valore contenuto nel registro accumulatore e quello propagato dalla memoria;
4) il quinto ingresso fa la somma fra il fra il dato memorizzato nell'accumulatore e quello propagato dalla RAM;
5) qui viene caricato il valore proveniente dalla memoria (load);
6) l'ALU non fa niente in quanto si dovrebbe salvare in memoria (store);
7) l'ALU non compie nessuna operazione in quanto si dovrebbe avere un salto all'interno della RAM;

La programmazione della memoria

I requisiti minimali del progetto prevedono che la cpu ordini un vettore di 5 elementi. Sulla base delle nostre conoscenze, abbiamo compilato la memoria come segue:

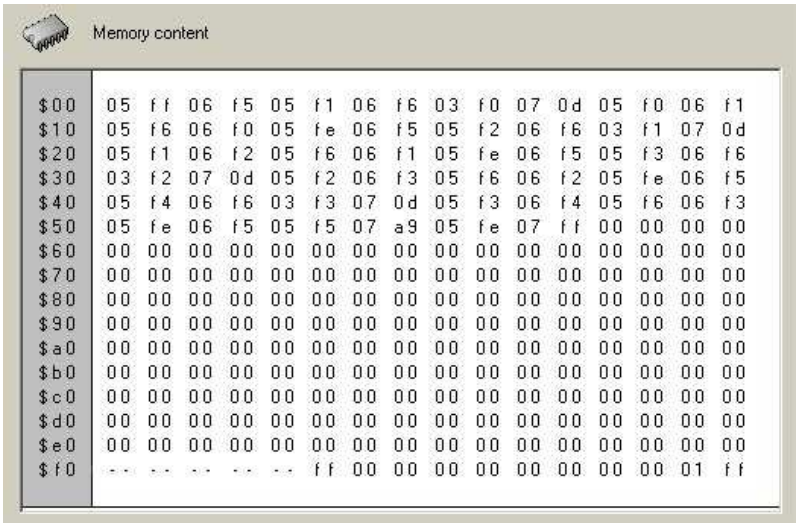

Figura 3

01= odd;02= even;03= sottrazione;04= addizione;05= carica;06= memorizza;07= salto (offset)

Come si può notare, un certo numero di operazioni sono ripetute: quelle che caricano il flag da “FE” e quelle che successivamente lo modificano. Avremmo in realtà potuto mantenere il codice più breve, memorizzando nella posizione del flag il valore stesso dell’elemento scambiato, ma abbiamo preso questa decisione per avere la certezza del contenuto del flag stesso. L’halt dinamico è ottenuto saltando in avanti di 255 posizioni.

OPCODE	ADDRESS	INDIRIZZI	OPERAZIONE	COMMENTO
05	FF	$00	Carica valore di ff	(CARICA IL VALORE DEL FLAG)
06	F5	$02	Scrivilo in f5	(MEMORIZZA IL VALORE DEL FLAG)
05	F1	$04	Carica valore di f1	(2°VALORE)
06	F6	$06	Scrivilo in f6	(ACC)
03	F0	$08	Sottrai valore di f0	(f1-f0)
07	0D	$0A	Salta di 14	(SE FLAG=00)
05	F0	$0C	Carica valore di f0	(1°VALORE)
06	F1	$0E	Scrivilo in f1	(2°VALORE)
05	F6	$10	Carica valore di f6	(ACC)(f6=2°VAL)
06	F0	$12	Scrivilo in f0	(1°VALORE)
05	FE	$14	Carica valore di fe	(01)
06	F5	$16	Scrivilo in f5	(FLAG)
05	F2	$18	Carica valore di f2	(3°VALORE)
06	F6	$1A	Scrivilo in f6	(ACC)
03	F1	$1C	Sottrai valore di f1	(f2-f1)
07	0D	$1E	Salta di 14	(SE FLAG=00)
05	F1	$20	Carica valore di f1	(2°VALORE)
06	f2	$22	Scrivilo in f2	(3°VALORE)
05	f6	$24	Carica valore di f6	(ACC)
06	f1	$26	Scrivilo in f1	(2°VALORE)
05	fe	$28	Carica valore di fe	(01)
06	f5	$2a	Scrivilo in f5	(FLAG)
05	f3	$2c	Carica valore di f3	(4°VALORE)
06	f6	$2e	Scrivilo in f6	(ACC)
03	f2	$30	Sottrai f2	(f3-f2)
07	0d	$32	Salta di 14	(SE FLAG=00)
05	f2	$34	Carica valore di f2	(3°VALORE)
06	f3	$36	Scrivilo in f3	(4°VALORE)
05	f6	$38	Carica valore di f6	(ACC)
06	f2	$3a	Scrivilo in f2	(3°VALORE)
05	fe	$3c	Carica valore di fe	(01)
06	f5	$3e	Scrivilo in f5	(FLAG)

05	f4	$40	Carica valore di f4	(5°VALORE)
06	f6	$42	Scrivilo in f6	(ACC)
03	f3	$44	Sottrai f3	(f4-f3)
07	0d	$46	Salta di 14	(SE FLAG=00)
05	f3	$48	Carica valore di f3	(4°VALORE)
06	f4	$4a	Scrivilo in f4	(5°VALORE)
05	f6	$4c	Carica valore di f6	(ACC)
06	f3	$4e	Scrivilo in f3	(4°VALORE)
05	fe	$50	Carica valore di fe	(01)
06	f5	$52	Scrivilo in f5	(FLAG)
05	f5	$54	Carica valore di f5	(FLAG)
07	a9	$56	Salta di 169	(SE FLAG=00)
05	fe	$58	Carica valore di fe	(01)
07	ff	$5a	Salta di 255	(HALT DINAMICO)

Mazzucco Michele
Pegoraro Alessandro
Pernici Andrea
Valenti Roberto