Esercitazione del 19/1/2011 – Costruire un MicroMISP

Descrizione

L'obbiettivo di questo esercizio è creare una CPU semplificata che lavori con word di 4 bit, sia per i dati che per gli indirizzi. Lo spazio di indirizzamento con parole di 4 bit è di $2^4 = 16$ istruzioni e dati. Se usiamo istruzioni a lunghezza fissa, come nel MIPS, e di lunghezza di una 1 word li numero di istruzioni diverse realizzabili è al più $2^4 = 16$ istruzioni[1].

Decidiamo di adottare come idea generale la struttura del MIPS: una memoria istruzioni ed una memoria dati separate, una ALU per eseguire calcoli sui valori contenuti nei registri del register file, in questo caso ridotto a soli due registri a 4 bit, uniche operazioni consentite con la memoria dati sono il trasferimento dati da e verso i registri.

Individuiamo un set minimo di istruzioni.

Occorre una coppia di istruzioni per trasferire dati da e verso la memoria, ad esempio LW e SW.

Occorre anche un'insieme di istruzioni aritmetiche per realizzare qualche operazione sui registri. Se ci limitiamo ad addizioni e sottrazioni possiamo definire le istruzioni ADD e SUB.

Occorre pure una coppia di istruzioni per modificare il flusso delle istruzioni, in maniera condizionata e non condizionata, ad esempio JUMP e BRANCH.

Infine occorre definire un'istruzione per poter settare i registri ad un valore prefissato, ad esempio una costante. Per semplicità, definiamo un'istruzione per settare A in maniera simile a come avviene nel MIPS, ad esempio LI, ed un'altra istruzione per scambiare tra loro il contenuto dei registri A e B, ad esempio SWAP. Come nel MIPS sorge il problema di come indicare perole di 4 bit all'interno di istruzioni di lunghezza fissa anch'esse di 4 bit. Nel MIPS il problema è risolto definendo due istruzioni, una per caricare la parte bassa della word, la pseudo-istruzione LI, ed una per caricare la parte alta del registri, l'istruzione LUI. Nel nostro caso adottiamo una soluzione hardware che ci permette di definire una sola istruzione per entrambi gli scopi: definiamo un'istruzione LI simile al MIPS in grado di caricare la parte bassa della word a 4 bit del registro A, contemporaneamente useremo i due bit bassi che verranno riscritti per completare la parte alta di A. In pratica quello che realizziamo è un'istruzione di caricamento a shift verso sinistra che carica 2 bit alla volta. Un caricamento completo si ottiene con due operazioni LI successive, una per definire la parte alta del registro ed una per definire la parte bassa.

Poichè la lunghezza degli opcode è estremamente ridotta ed il numero di registri limitato decidiamo di dedicare tutto lo spazio disponibile, 4 bit, per l'opcode. In pratica questo ci obbliga a definire istruzioni che manipolano i registri in un modo fissato, al contrario di come avviene nel MIPS in cui è possibile definire quali registri sono interessati dall'istruzione. Decidiamo quindi che le operazioni sulla memoria useranno sempre il registro A per gli indirizzi ed il registro B per i dati in ingresso ed uscita. Per le operazioni aritmetiche il calcolo andrà sempre eseguito considerando come primo operando A e come secondo B ed il risultato finale sarà sempre memorizzato in B. Infine le operazioni di salto useranno sempre A come indirizzo finale ed i salti condizionati saranno decisi dalla ALU considerando direttamente valore del secondo operando, cioè il registro B, e valutando se è uguale a zero, senza eseguire nessun calcolo.

1 Chiaramente una CPU con queste caratteristiche è inutilizzabile in applicazioni reali ed ha il solo scopo di mettere in evidenza alcuni aspetti del funzionamento delle CPU di tipo RISC.

Istruction set

Di seguito è indicato l'istruction set della MicroMISP definita sopra: nella colonna descrizione è descritto formalmente il comportamento di ogni istruzione.

Instruction	Op_3	Op_2	Op_1	Op_0	Descrizione
ADD/SUB SWAP	0	0	~Add/Sub	~Swap/Add-Sub	Se Op_0=1 e Op1=0 → B=A+B Se Op_0=1 e Op1=1 → B=A-B Se Op_0=0 → A↔B PC=PC+1
LI	0	1	b_1	b_0	A=A_1A_0|b_1b_0 , PC=PC+1
LW/SW	1	0	~Lw/Sw	0	Se Op1=0 → B=MEM[A] Se Op1=1 → MEM[A]=B PC=PC+1
JUMP/BRANCH	1	1	~Jump/Branch	0	Se Op1=0 → PC=A Se Op1=1 e B=0→ PC=A Se Op1=1 e B=0→ PC=PC+1

Per le istruzioni multiple, come ADD/SUB, i campi Op_1 e Op_0 vanno presi a 0 (voce indicata con la negazione ~) oppure 1 (voce senza negazione) a seconda della funzione richiesta.

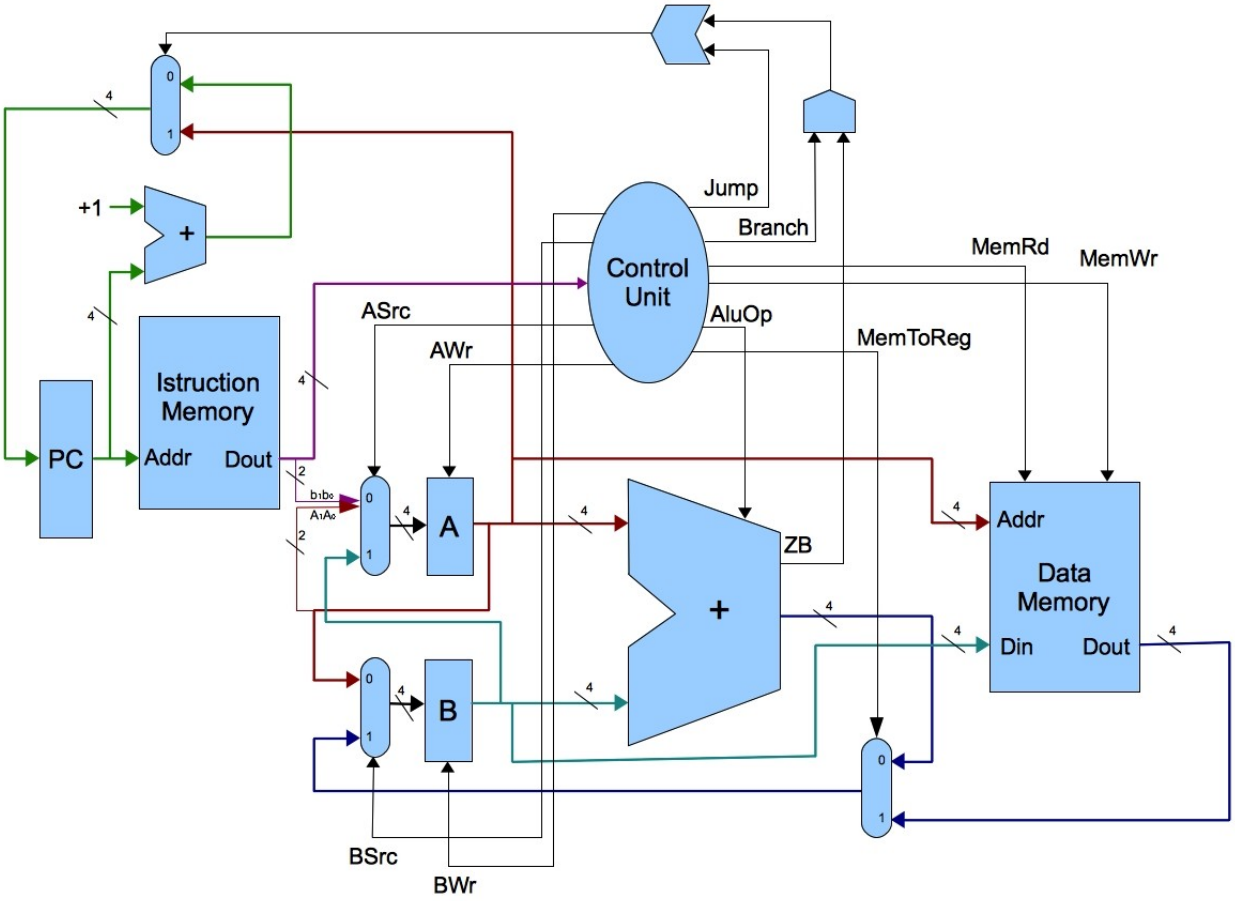
> *Esempio: Per l'istruzione ADD bisogna prendere il bit Op_1 uguale 0 ed il bit Op_0 uguale 1; l'opcode è quindi uguale a 0001.*

Esempi di programmi

```
MEM[1]=4+2:     0x1: 0100₂   LI 00
                0x1: 0110₂   LI 10      ; A := 2
                0x1: 0000₂   SWAP       ; B := 2
                0x1: 0110₂   LI 01
                0x1: 0100₂   LI 00      ; A := 4
                0x1: 0001₂   ADD        ; B := A+B = 4+2 = 6
                0x1: 0100₂   LI 00
                0x1: 0101₂   LI 01      ; A :=1
                0x1: 1010₂   SW         ; MEM[A]=B

IF (MEM[1]=0) A=1   0x1: 0100₂   LI 00
          ELSE A=0  0x2: 0101₂   LI 01      ; A :=1
                    0x3: 1000₂   LW         ; B := MEM[A] = MEM[1]
                    0x4: 0110₂   LI 10
                    0x5: 0110₂   LI 10      ; A=0A
                    0x6: 1110₂   BRANCH     ; Jump to 0xA If B=0
                    0x7: 0100₂   LI 00
                    0x8: 0101₂   LI 01      ; A=1
                    0x9: .....   ......    ; altro codice
                    0xA: 0100₂   LI 00
                    0xB: 0100₂   LI 00      ; A=0
                    0xC: .......            ; altro codice
```

Schema circuitale della MicroMIPS



MicroMIPS

Tabella di verità dell'unità di controllo

Di seguito è implementata la tabella di verità dell'unità di controllo. AluOp vale 0 quando occorre eseguire la somma, vale 1 qundo occorre la sottrazione.

Instr.	Op_3	Op_2	Op_1	Op_0	Asrc	Bsrc	AWr	BWr	MemToReg	Aluop	MemRd	MemWr	Jump	Branch
SWAP	0	0	0	0	1	0	1	1	X	X	0	0	0	0
ADD	0	0	0	1	X	1	0	1	0	0(+)	0	0	0	0
SUB	0	0	1	1	X	1	0	1	0	1(-)	0	0	0	0
LI	0	1	b_1	b_0	0	X	1	0	X	X	0	0	0	0
LW	1	0	0	0	X	1	0	1	1	X	1	0	0	0
SW	1	0	1	0	X	X	0	0	X	X	0	1	0	0
JUMP	1	1	0	0	X	X	0	0	X	X	0	0	1	X
BRANCH	1	1	1	0	X	X	0	0	X	X	0	0	0	1

Tabella completa

Diamo una versione completa della tabella da usare per realizzare lo schema circuitale dell'unità di controllo. I valori indeterminati sono stati presi in modo da semplificare le funzioni dei segnali.

Instr.	Op_3	Op_2	Op_1	Op_0	Asrc	Bsrc	AWr	BWr	MemToReg	Aluop	MemRd	MemWr	Jump	Branch
SWAP	0	0	0	0	1	0	1	1	*0*	*0*	0	0	0	0
ADD	0	0	0	1	*1*	1	0	1	0	0	0	0	0	0
SUB	0	0	1	1	*1*	1	0	1	0	1	0	0	0	0
LI	0	1	b_1	b_0	0	b_0	1	0	*0*	b_1	0	0	0	0
LW	1	0	0	0	*1*	1	0	1	1	*0*	1	0	0	0
SW	1	0	1	0	*1*	*1*	0	0	*1*	*1*	0	1	0	0
JUMP	1	1	0	0	*0*	*1*	0	0	*1*	*0*	0	0	1	*1*
BRANCH	1	1	1	0	*0*	*1*	0	0	*1*	*1*	0	0	0	1

Asrc = $\sim Op_2$

Bsrc = (Op_3+Op_0)

AWr = $(\sim Op_3 \sim Op_2)\sim Op_0+(\sim Op_3 Op_2)$

BWr = $(\sim Op_3 \sim Op_2)+[(Op_3 \sim Op_2)\sim Op_1]$

= $(\sim Op_3 \sim Op_2)+\sim Op_2 \sim Op_1$

MemToReg = Op_3

AluOp = Op_1

MemRd = $[(Op_3 \sim Op_2)\sim Op_1]$

MemWr = $(Op_3 \sim Op_2)Op_1$

Jump = $(Op_3 Op_2)\sim Op_1$

Branch = $(Op_3 Op_2)$