UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

ARCHITETTURE DEGLI ELABORATORI

Prova scritta - 8 Febbraio 2024

Cognome e nome Matricola

Esercizio 1.

Sia F una funzione che riceve in ingresso 2 interi A e B rappresentati in cp2 su 2 bit. Si assuma che non possa verificarsi che A e B riferiscano lo stesso valore. F restituisce 0 quando A<B, e 1 quando A>B. Realizzare un circuito che implementa F usando le mappe di Karnaugh facendo sintesi in forma PoS. Riportare i passaggi e disegnare il circuito derivato.

Soluzione:

La funzione F è definita dalla seguente tabella di verità, che riporta per ogni valore di A e B, rappresentato in cp2, il valore restituito da F. In base all'assunzione riferita, quando A=B è indifferente che F restituisca 1 o 0. Sfruttiamo tale condizione di indifferenza per minimizzare il circuito.

A	B	A_1	A_0	B_1	B_0	F
0	0	0	0	0	0	X
0	1	0	0	0	1	0
0	-2	0	0	1	0	1
0	-1	0	0	1	1	1
1	0	0	1	0	0	1
1	1	0	1	0	1	X
1	-2	0	1	1	0	1
1	-1	0	1	1	1	1
-2	0	1	0	0	0	0
-2	1	1	0	0	1	0
-2	-2	1	0	1	0	X
-2	-1	1	0	1	1	0
-1	0	1	1	0	0	0
-1	1	1	1	0	1	0
-1	-2	1	1	1	0	1
-1	-1	1	1	1	1	X

Dalla tabella si deriva la seguente mappa di Karnaugh:

A_1A_0 \ B_1B_0	00	01	11	10
00	X	0	1	1
01	1	X	1	1
11	0	0	X	1
10	0	0	0	X

La mappa derivata ammette 4 soluzioni:

A_1A_0 \ B_1B_0	00	01	11	10
00	X	0	1	1
01	1	X	1	1
11	0	0	X	1
10	0	0	0	X

a) $F = (B_1 + \overline{B_0})(\overline{A_1} + B_1)(\overline{A_1} + A_0)$

A_1A_0 \ B_1B_0	00	01	11	10
00	X	0	1	1
01	1	X	1	1
11	0	0	X	1
10	0	0	0	X

b) $F = (B_1 + \overline{B_0})(\overline{A_1} + B_1)(\overline{A_1} + \overline{B_0})$

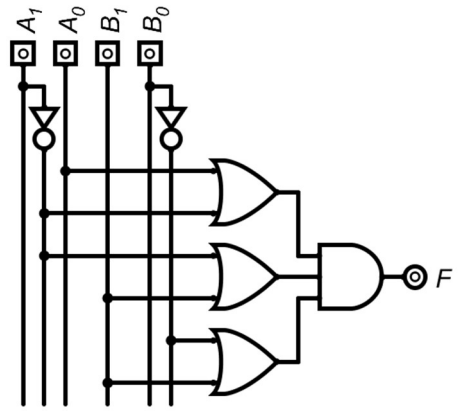
A_1A_0 \ B_1B_0	00	01	11	10
00	X	0	1	1
01	1	X	1	1
11	0	0	X	1
10	0	0	0	X

c) $F = (A_0 + B_1)(\overline{A_1} + B_1)(\overline{A_1} + A_0)$

A_1A_0 \ B_1B_0	00	01	11	10
00	X	0	1	1
01	1	X	1	1
11	0	0	X	1
10	0	0	0	X

d) $F = (A_0 + B_1)(\overline{A_1} + B_1)(\overline{A_1} + \overline{B_0})$

La rete logica che implementa l'espressione *a)* è la seguente:



Esercizio 2.

Derivare la codifica floating point IEEE 754 in singola precisione del numero -721.5625. Si ricorda che la mantissa va rappresentata su 23 bit e l'esponente su 8.

Soluzione:

Bit di segno (1): segno (-)

Parte intera: 721 → 1011010001; Parte frazionaria: .5625 → .1001; da cui: 721.5625=1011010001,1001

Normalizzo spostando la posizione della virgola di 9 posizioni verso sinistra: 721.5625=1,0110100011001 * 2^9

Rappresento l'esponente 9 in codice eccesso con polarizzazione 127. La rappresentazione equivale alla codifica binaria pura di 9+127=136 ovvero 10001000

La mantissa normalizzata ha parte intera implicitamente a 1: è necessario rappresentare su 23 bit solo la parte frazionaria: 0110100011001 000000….0

La codifica IEEE 754 di -721.5625 risulta quindi: 1 100001000 01101000110010000000000

Esercizio 3.

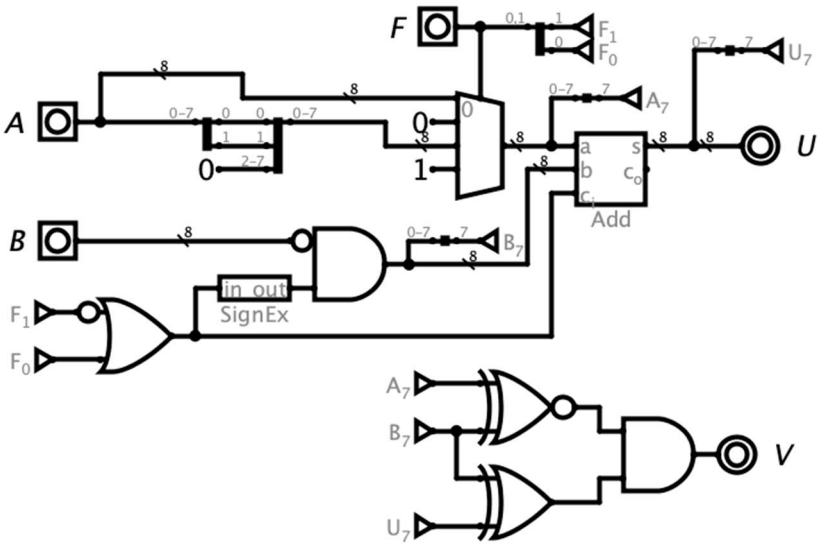
Realizzare una ALU dotata di un ingresso A, e di un'uscita U, entrambi su 8 bit con rappresentazione cp2, e che in base ad un ingresso F gestisce le operazioni: A-B, -B, A mod 4, 1-B. La ALU genera il bit di esito V che notifica eventuali overflow.

Soluzione:

Specifichiamo i valori di F per cui eseguire le varie operazioni, servono 2 bit: i) F=00 → A-B; ii) F=01→ -B; iii) F=10 → A mod 4; iv) F=11: 1-B.

Per usare un solo sommatore esprimo tutte le operazioni come somme: A-B = A+cp1(B)+1, -B = 0+cp1(B)+1, A mod 4 = (A mod 4)+0+0, 1-B = 1+cp1(B)+1.

Per regolare il valore da inoltrare sull'ingresso *a* del sommatore uso un MUX con ingresso di selezione su 2 bit che assume il valore di F, e ingressi dati su 8 bit, che riportano i valori di A (con F=00), 0 (con F=01), A mod 4 (con F=10), e 1 (con F=11).



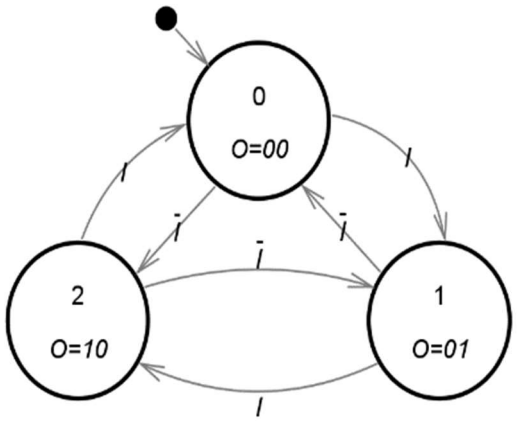
L'ingresso *b* del sommatore assume valore cp1(B) con F=00, 01 e 11, e vale 0 per F=10. Non serve un MUX per regolare il valore in ingresso a *b*. Per ottenere su *b* i valori cp1(B) o 0 calcolo il prodotto logico tra cp1(B) (su 8 bit) e una sequenza di soli 1 o soli 0 che derivo partendo dai segnali F_1 e F_0. Per ottenere una sequenza di soli 0 con F=10 è sufficiente estendere su 8 bit in segno $\overline{F_1}\overline{F_0}$, ma tale espressione è equivalente a $\overline{F_1} + F_0$. Infine, c_{in} vale 1 quando F vale 00, 01 e 11, ovvero se $\overline{F_1} + F_0$. Poiché tutte le operazioni sono espresse come somme si genera overflow (V=1) sse $A_7 = B_7$ e $B_7 != U_7$.

Esercizio 4.

Progettare un circuito sequenziale sincrono che implementa un contatore modulo 3 dotato di un ingresso I di 1 bit. Quando I=1 il circuito conta in avanti di 1, mentre, quando I=0 conta all'indietro di 1. Il circuito è dotato di un'uscita O, su 2 bit, che riferisce in binario puro il valore memorizzato dal contatore. Per la realizzazione circuitale è necessario impiegare la codifica a singolo 1 (un solo bit per stato vale 1) ed usare Flip-Flop D. E' sufficiente mostrare uno schema circuitale ad alto livello di astrazione, rappresentando come blocchi le reti combinatorie che compongono il circuito, ma è necessario specificare le funzioni logiche implementate da tali reti.

Soluzione:

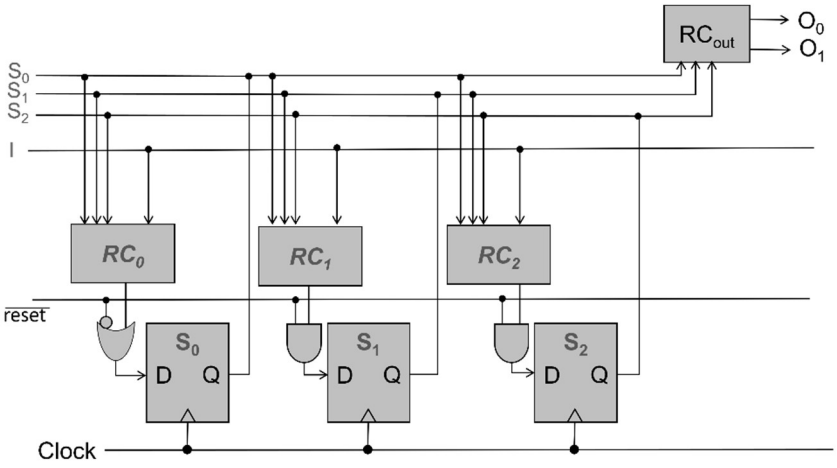
Il comportamento del circuito è modellato dalla FSM in figura. Gli stati denotano il valore memorizzato dal contatore, e riportano il valore assunto dall'uscita O.



Assumiamo che gli stati siano codificati come segue:

0 → 001; 1 →010; e 2 → 100.

Lo schema di riferimento per la realizzazione circuitale è il seguente, dove un segnale di reset attivo basso è usato per impostare lo stato iniziale del circuito.



RC_{out} implementa la funzione specificata dalla seguente tabella di verità.

S_2	S_1	S_0	O_1	O_0
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0

Da cui, sfruttando la codifica a singolo 1, abbiamo che:

$O_1 = S_2 \qquad O_0 = S_1$

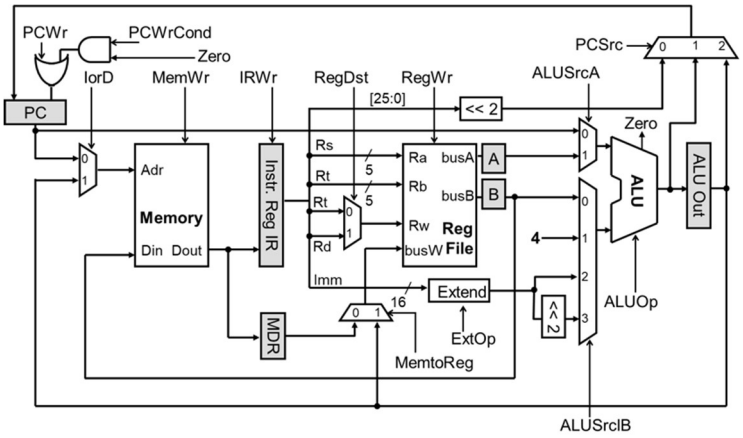
Infine, dall'analisi della FSM deriviamo le reti combinatorie che gestiscono le transizioni di stato. Le reti sono fatte come segue:

RC_0: $S_0' = 1\ sse\ S_2 I + S_1 \bar{I}$ $\qquad$ RC_1: $S_1' = 1\ sse\ S_0 I + S_2 \bar{I}$ $\qquad$ RC_2: $S_2' = 1\ sse\ S_1 I + S_0 \bar{I}$

Esercizio 5.

Facendo riferimento al datapath multiciclo del MIPS in figura, descrivere brevemente le fasi del ciclo di esecuzione dell'istruzione *lw rt, rs, imm16* (load word) motivando le risposte date.

Si ricorda che *lw* carica nel registro *rt* la parola presente in memoria all'indirizzo riferito dalla somma del contenuto di *rs* e *imm16*.



Soluzione:

Fase	**Trasferimenti tra registri**	**Motivo** *(breve spiegazione, max 3 righe)*
1)	IR ← Mem[PC] PC ← PC+4	…
2)	A ← R[IR[rs]] B ← R[IR[rt]]	
3)	ALUOut ←A+(SignExt(IR[imm16]))	
4)	MDR ←Mem[ALUOut]	
5)	R[IR[rt]] ←MDR	

Esercizio 6. Si consideri un sistema di memoria caratterizzato da una memoria di lavoro di 64 KB indirizzata a livello di Byte, e una cache set associativa a 4 vie inizialmente vuota. La cache gestisce 8 blocchi, riferiti A, B, C, D, E, F, G e H, ripartiti in tal ordine tra i set (seguendo l'ordine crescente dell'indice del set). Ogni blocco è composto da 16 parole da 64 bit. Considerando la sequenza di richieste alla memoria riportata in tabella, e assumendo per la sostituzione dei blocchi in cache la politica LRU, si chiede di completare la tabella che illustra il comportamento della cache nel rispetto delle indicazioni seguenti:

- Nella colonna *esito* riportare H (hit) se il blocco richiesto si trova nella cache, M (miss) se invece il blocco deve essere caricato dalla memoria.
- Nelle colonne *dati* deve essere riportato l'indice del blocco della memoria (in decimale), presente nel corrispondente blocco della cache.
- Nella colonna *azione* indicare il blocco a cui si accede in caso di hit, o in cui si caricano i dati della memoria in caso di miss e l'indice del blocco caricato (in decimale).

		Blocco A			**Blocco B**			**Blocco C**			**Blocco D**			**Blocco E**			**Blocco F**			**Blocco G**			**Blocco H**			
	Esito	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Valido**	**Etichetta**	**Dati**	**Azione**
1) Richiesta *addr* 1000000011100010	M													1	10000000	257										Carico in E (set 1) il blocco 257
2) Richiesta *addr* 1100000010001011	M																1	11000000	385							Carico in F (set 1) il blocco 385
3) Richiesta *addr* 1000000011001001	H																									Accedo in E (set 1) al blocco 257
4) Richiesta *addr* 1100000011001001	H																									Accedo in F (set 1) al blocco 385

Mem size: 64KB (2^{16} *byte*) → indirizzamento a livello di byte su 16 bit; Cache size (senza tag): 8 *16*8 byte (*8 blocchi da 16 parole di 8 byte*)= 1KB (2^{10} *byte*);
Block size: 16 word * 64 bit= 128 byte (2^4*2^3 *byte*) → byte offset nel blocco su 7 bit (*4 bit di word index + 3 bit di byte offset a livello di word*);
Set size= 512 Byte (*4 * 128 byte*); Num set: 1KB /512 byte=2 → set index su 1 bit; Tag su 8 bit (*16-(7+1)*); Scomposizione indirizzo memoria: 8bit tag + 1bit set idx + 4 bit di word index + 3 bit di byte offset a livello di word.
NB. La tabella riporta una possibile evoluzione (ad es. al passo 1 al posto di E potrebbe essere scelto il blocco F, G o H…)