UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

ARCHITETTURE DEGLI ELABORATORI

Prova scritta del 24 Gennaio 2024

Cognome e nome Matricola

Esercizio 1. Sia F una funzione che riceve in ingresso un numero intero n senza segno rappresentato su 4 bit. F assume valore 0 se n>0 ed n è multiplo di 3 o di 4, e restituisce indifferentemente 0 o 1 se n vale 2, 5, 7, 11 o 13. F assume valore 1 per gli altri valori di n. Realizzare il circuito che implementa F usando le mappe di Karnaugh, sintetizzando 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 N rappresentato con codifica binaria posizionale, il valore restituito da F.

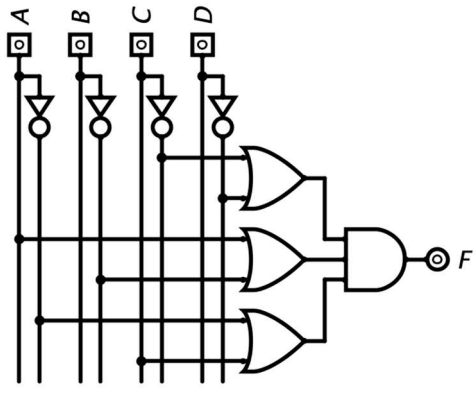
N	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	X
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	X
6	0	1	1	0	0
7	0	1	1	1	X
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	X
12	1	1	0	0	0
13	1	1	0	1	X
14	1	1	1	0	1
15	1	1	1	1	0

Dalla tabella si deriva la seguente mappa di Karnaugh:

AB \ CD	00	01	11	10
00	1	1	0	X
01	0	X	X	0
11	0	X	0	1
10	0	0	X	1

$$F = (\overline{C} + \overline{D})\,(A + \overline{B})\,(\overline{A} + C)$$

Il circuito che implementa F è il seguente:



Esercizio 2. Derivare il valore decimale del numero N rappresentato con la seguente codifica floating point IEEE 754 in singola precisione: 1 10000111 10110101110000000000000

Soluzione. Bit di segno (1) → Segno (-)

Esponente 10000111 in codice eccesso 127: (128+4+2+1)-127= 135-127=8

Il numero è in forma normalizzata, pertanto la mantissa è implicitamente dotata di parte intera valorizzata 1. La mantissa è pertanto: 1,1011010111

N vale (-1)* 1,1011010111 * 2^8. Spostando la posizione della virgola di 8 posizioni verso destra ottengo: (-1)*110110101,11= -437,75

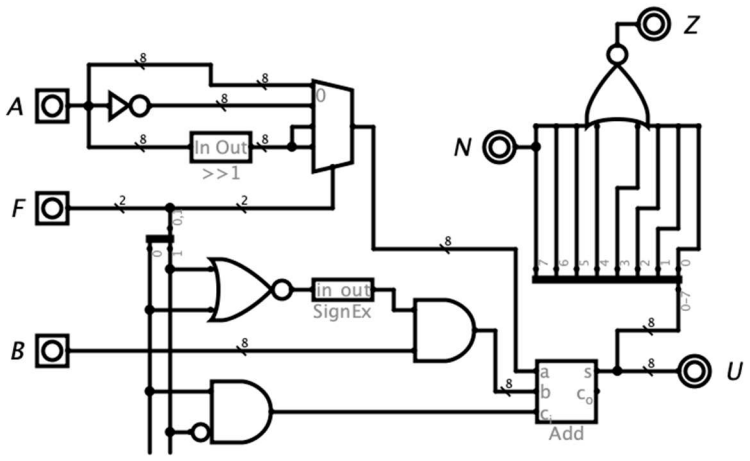
Esercizio 3. Realizzare una ALU dotata degli ingressi A e B, e dell'uscita U, tutti su 8 bit e con rappresentazione cp2, che in base ad un ingresso F gestisce le operazioni: A+B, -A, e A>>1. La ALU genera i bit di esito: i) Z, che notifica e il valore su U vale 0, e ii) N, che notifica se il risultato è negativo. Si trascurino gli overflow.

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

Serve un sommatore per gestire le operazioni A+B (F=00) e -A (F=01), infatti, -A è esprimibile come cp1(A) +1.

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

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), cp1(A) (con F=01), e A>>1 (con F=10 o F=11).



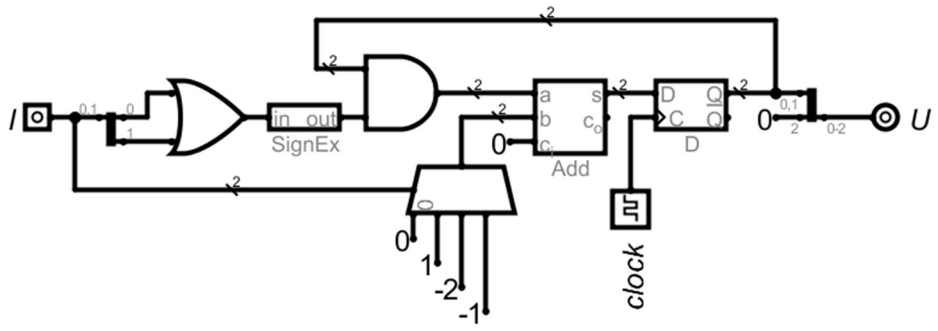
L'ingresso *b* del sommatore assume valore B se F=00, e 0 per gli altri valori di F. Non serve un MUX per regolare il valore in ingresso a *b*, è sufficiente riportare il prodotto logico tra B ed il risultato della valutazione di $\overline{F_1 + F_0}$ esteso in segno su 8 bit. Infine, c_{in} vale 1 solo quando F=01, ovvero $c_{in} = \overline{F_1}\, F_0$

I bit di esito Z richiede una porta NOR che riceve sui suoi 8 ingressi gli 8 bit della rappresentazione di U. Infine, il bit di esito N corrisponde al MSB(U) ovvero a U_7.

Esercizio 4. Progettare un circuito sequenziale sincrono che implementa un contatore modulo 4 dotato di un ingresso su 2 bit. Quando I=01 il circuito conta in avanti di 1, quando I=11 conta all'indietro di 1, e quando I= 10 conta all'indietro di 2. Infine, quando I=00 si azzera il valore memorizzato. Il circuito è dotato di un'uscita U che riporta il valore memorizzato in cp2. Per la realizzazione circuitale è necessario usare Flip-Flop D.

Soluzione. I valori che può assumere il contatore sono 0, 1, 2 o 3, pertanto per rappresentare lo stato servono 2 bit, e 2 flip-flop D per tenerne traccia. Per gestire gli incrementi/decrementi uso un sommatore a 2 bit con uscita *s* connessa all'ingresso D dei 2 flip-flop. Il sommatore riceve: i) sull'ingresso *a* il valore 0 quando I=00 (effettua il reset), o il valore correntemente memorizzato dai flip-flop per tutti gli altri valori di I; ii) sull'ingresso *b* il valore dell'incremento/decremento da effettuare che in base ad I può valere 0 (I=00), 1 (I=01), -2 (I=10) o -1 (I=11); e infine iii) sull'ingresso c_{in} il valore 0.

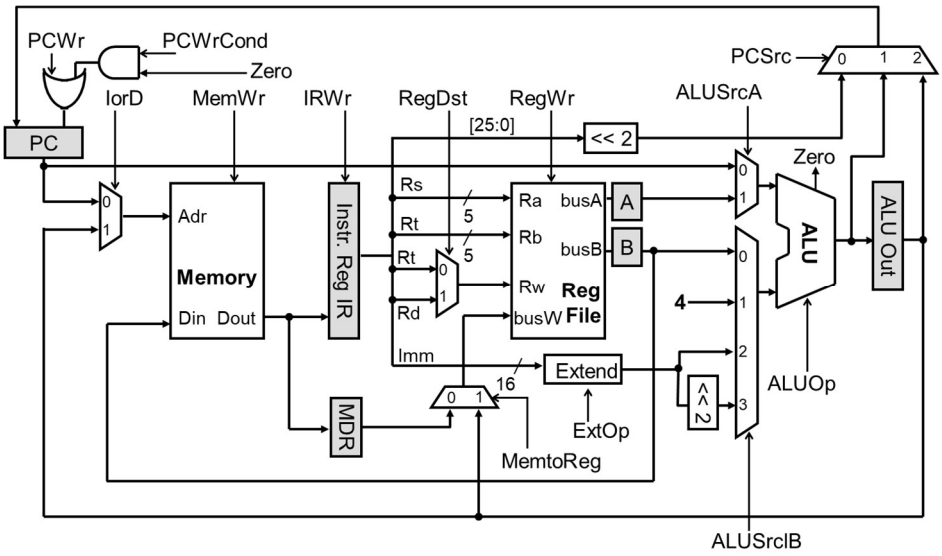
Per regolare il valore sull'ingresso *a* del sommatore effettuo il prodotto logico tra lo stato corrente e l'estensione in segno su 2 bit di $I_1 + I_0$. Quando I=00, abbiamo che $I_1 + I_0 = 0$, che esteso in segno in segno su 2 bit corrisponde al valore 00. Mettendo in AND bitwise 00 e lo stato corrente ottengo il valore 0 sull'ingresso *a*. Per gli altri valori di I abbiamo che $I_1 + I_0 = 1$, che esteso in segno in segno su 2 bit corrisponde al valore 11. Mettendo in AND bitwise 11 e lo stato corrente ottengo sull'ingresso *a* la rappresentazione su 2 bit dello stato corrente.



Infine, poiché il valore su U viene rappresentato in cp2 è necessario estendere a 0 su 3 bit il valore memorizzato.

Esercizio 5. Facendo riferimento al datapath multiciclo in figura, descrivere brevemente le fasi di esecuzione dell'istruzione *bne rt, rs, imm16* (branch on not equal) motivando le risposte date.

bne deriva l'indirizzo della prossima istruzione dalla comparazione del contenuto dei registri *rs* ed *rt*. Se hanno lo stesso contenuto l'indirizzo della prossima istruzione segue quello dell'istruzione corrente a distanza di una parola. Invece, con contenuto differente, l'indirizzo si ottiene sommando all'incremento base del PC, lo spiazzamento, a livello di parola, riferito da *imm16*.



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]] ALUOut ←PC+(SignExt(IR[imm16])<<2)	
3)	if(A!=B) PC← ALUOut	
4)		
5)		

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 ad indirizzamento diretto, inizialmente vuota, che gestisce 8 blocchi. Il sistema gestisce blocchi composti da 128 parole da 32 bit. Considerando la sequenza di richieste alla memoria riportata in tabella, 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 la linea 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)

		Linea 0			**Linea 1**			**Linea 2**			**Linea 3**			**Linea 4**			**Linea 5**			**Linea 6**			**Linea 7**			
	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* 1101 001 110010111	M				1	1101	105																			Carico il blocco 105 in linea 1
2) Richiesta *addr* 1011 101 111010011	M																1	1011	93							Carico il blocco 93 in linea 5
3) Richiesta *addr* 1101 100 101010001	M													1	1101	108										Carico il blocco 108 in linea 4
4) Richiesta *addr* 1011 101 110010111	H																									Accedo al blocco 93 in linea 5

Mem_size: 64KB = 2^{16} byte → indirizzamento su 16bit Word_size: 32 bit = 4 byte Num_word: 128 →Word index su 7 bit

Block_size (nword* word_size)=128* 4byte= 2^7*2^2byte=2^9 byte Byte offset a livello di blocco su 9 bit, di cui 7 bit di indice parola, e 2 di byte offset a livello di word

Cache_size (nblock* block_size): 2^9 byte * 8=2^{12} byte Num_block: 8 → Line index su 3 bit

Tag su 4 bit *(16-(7+2+3))* Schema di un indirizzo: *Tag* 4 bit | *Line index* 3 bit | *Word index* 7 bit | *Byte offset (word)* 2 bit