Corso di Laurea in Ingegneria Informatica

Calcolatori — a.a. 2018–2019

Compito del 17 giugno 2019

Cognome e Nome dello studente: _____

La radice quadrata di un intero positivo N può essere calcolata nella forma $\sqrt{N} = (a, b)$, con a (radice intera) e b (residuo) interi tali che $a^2 \leq N < (a+1)^2$, $b = N - a^2$. Un algoritmo per il calcolo della radice di un N espresso in rappresentazione naturale su $2k$ bit (il simbolo $p \gg q$ indica la divisione di p per 2^q tramite lo scorrimento a destra di q posizioni) è il seguente:

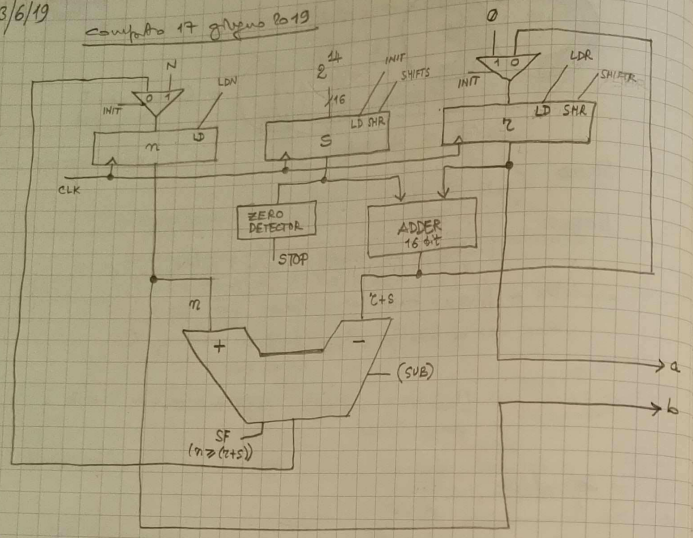
$n \leftarrow N$, $r \leftarrow 0$, $s \leftarrow 2^{2k-2}$ [initialize]
while $s > n$: [adjust]
    $s \leftarrow (s \gg 2)$
while $s \neq 0$: [compute/update]
    if $n \geq (r + s)$: {
        $n \leftarrow n - (r + s)$
        $r \leftarrow (r \gg 1) + s$
    }
    else: $r \leftarrow (r \gg 1)$
    $s \leftarrow (s \gg 2)$
return $a = r$, $b = n$ [terminate]

Reti Logiche Facendo riferimento al metodo di progettazione "parte operativa/parte di controllo": (i) disegnare la parte operativa di una macchina sequenziale sincrona che realizzi l'algoritmo sopra riportato nel caso $2k = 16$; (ii) specificarne il controllo attraverso il diagramma degli stati, facendo attenzione a riportare tutti i necessari segnali di comando (output) e di condizione (input); (iii) simulare il funzionamento temporale della macchina per $N = 3768$; (iv) realizzare la parte di controllo con la tecnica "registro di stato e multiplexer".

Programmazione LM Scrivere un programma in linguaggio Assembly 8086 che, data una variabile di memoria di tipo "word" `N`, calcoli a e b secondo l'algoritmo sopra riportato, ponendoli rispettivamente nei registri di macchina `AX` e `BX`.

23/6/19

completato 17 giugno 2019



NOTE

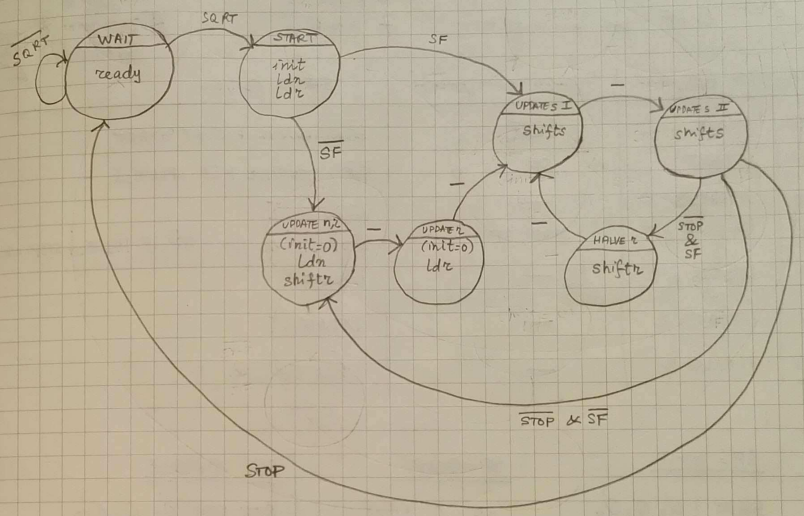
① Dopo l'inizializzazione ($n \leftarrow N$, $r \leftarrow 0$, $s \leftarrow 2^{14}$), durante la fase di aggiustamento di s (ciclo while "adjust") la condizione $s > n$ si può riscrivere come $n - (r+s) < 0$, poiché $r = 0$. Si esce da questa condizione se $SF = 0 \Leftrightarrow n - (r+s) \geq 0$.

② Durante la fase di calcolo ("compute/update"), dopo l'eventuale aggiornamento di n, r va comunque shiftata a destra di 1. Il nuovo valore di r si troverà già all'interno del registro se $SF = 1$, mentre dovrà essere caricato dall'esterno se $SF = 0$.

③ Lo ZERO DETECTOR è una semplice porta NOR con ingresso S:

$$STOP = \overline{S}_{15} \cdot \overline{S}_{14} \cdot \overline{S}_{13} \cdot \cdots \cdot \overline{S}_2 \cdot \overline{S}_1 \cdot \overline{S}_0 = \overline{S_{15} + S_{14} + S_{13} + \cdots + S_2 + S_1 + S_0}$$

$2^{14} = (0100000000000000)_2$ costante di inizializzazione di S.



④ Nella realizzazione del controllo più sopra, ciascuna iterazione della fase di "adjust" richiede tre stati (*) (shifts, shifts, shiftr) anziché due (shifts, shifts), perché in questo modo i tre stati possono essere sfruttati tali e quali nella fase "compute/update" successiva. In alternativa, i due stati di adjust (shifts) dovrebbero essere diversi da quelli di compute/update, con un incremento (tabella) di due stati (9, anziché 7). (*) con $r \gg 1 = 0$, poiché $r = 0$.

⑤ Si ipotizza che l'aggiornamento di n ($n \leftarrow n - (r+s)$) avvenga sul fronte del clock, dunque con ritardamento da non scontare del "contemporaneo" (cioè nello stesso stato) aggiornamento parziale di r ($r \gg 1$). In questo modo, si evita di usare uno stato in più, che sarebbe necessario per $r \leftarrow (r \gg 1) + s$ poiché shiftr e ldr sono mutuamente esclusivi.

Sintesi con registro di stato e multiplexer

3 FF per memorizzare i 7 stati

Funzione di Transizione di stato (f)

Codifica	STATO PRESENTE	ingressi campionati	STATO FUTURO
000	WAIT	SQRT	$\overline{SQRT} \cdot WAIT + SQRT \cdot START$
001	START	SF	$\overline{SF} \cdot UPDATE_{nr} + SF \cdot UPDATE_{SI}$
010	UPDATE s I	—	$UPDATE_{SII}$
011	UPDATE s II	STOP, SF	$\overline{STOP} \cdot \overline{SF} \cdot UPDATE_{nr} + \overline{STOP} \cdot SF \cdot HALVE_r + STOP \cdot WAIT$
100	UPDATE n,r	—	$UPDATE_r$
101	UPDATE r	—	$UPDATE_{SI}$
110	HALVE r	—	$UPDATE_{SI}$

SP = WAIT

SQRT	$Q_2' Q_1' Q_0'$
0	000
1	001
	↓ ↓ ↓
	∅ ∅ SQRT

SP = START

SF	$Q_2' Q_1' Q_0'$
0	100
1	010
	↓ ↓ ↓
	$\overline{SF}$ SF ∅

SP = UPDATE$_{SII}$

STOP SF	$Q_2' Q_1' Q_0'$
00	100
01	110
10	000
11	000
	↓ ↓ ↓
	$\overline{STOP}$ STOP · SF

SP = UPDATE$_{SI}$

$Q_2' Q_1' Q_0' = 011$

SP = UPDATE$_{nr}$

$Q_2' Q_1' Q_0' = 101$

SP = UPDATE$_r$

$Q_2' Q_1' Q_0' = 010$

SP = HALVE$_r$

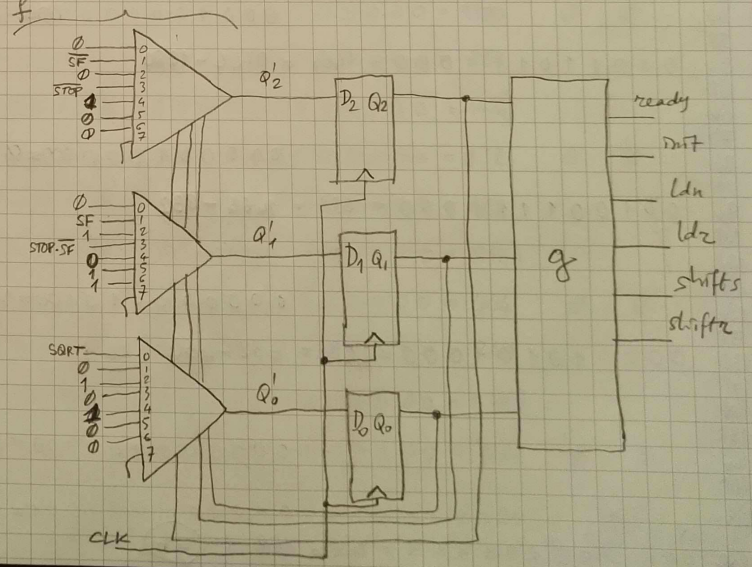
$Q_2' Q_1' Q_0' = 010$

Funzione d'uscita (g)

Q_2	Q_1	Q_0	ready	init	ldn	ldr	shiftS	shiftR
0	0	0	1	0	0	0	0	0
0	0	1	0	1	1	1	0	0
0	1	0	0	0	0	0	1	0
0	1	1	0	0	0	0	1	0
1	0	0	0	0	1	0	0	1
1	0	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	1
1	1	1	X(0)	X(0)	X(0)	X(0)	X(0)	X(0)
			↓	↓	↓	↓	↓	↓
			$\overline{Q_2}\overline{Q_1}\overline{Q_0}$	$Q_2\overline{Q_1}Q_0$		$\overline{Q_1}Q_0$	$\overline{Q_2}Q_1$	$Q_2\overline{Q_0}$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	1	0	0
1	1	0	X	0

$ldn = Q_2\overline{Q_1}\overline{Q_0} + \overline{Q_2}Q_1Q_0$



Funzionamento temporale.

$N = 3768 = (1\,1\,1\,0\,1\,0\,1\,1\,1\,0\,0\,0)_2$ (bit 11 10 9 8 7 6 5 4 3 2 1 0)

La fase adjust termina quando $s = 2^{14}$, diviso via via per 4, risulta più piccolo di N. Questo accade dopo due divisioni per 4 (ciascuna delle quali corrisponde a uno shift a destra di 2 bit): $s = 2^{10} = (0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0)_2 = 1024$

STATO PRESENTE	USCITE generate nello stato	operazioni svolte
011 (UPDATEs II)	shifts	$s \leftarrow s/2$
100 (UPDATE nr)	ldn, shiftr	$n \leftarrow n-(r+s)$, $r \leftarrow r/2$
101 (UPDATE r)	ldr	$r \leftarrow r+s$
010 (UPDATEs I)	shifts	$s \leftarrow s/2$
110 (Halve r)		

Vediamo il funzionamento a partire dallo stato UPDATEs II in cui si ottiene per la prima volta $N \geq s$, ossia SF = 0.

STOP → va a 1 al clock 23↓ ($s \overset{?}{=} 0$); SF → $n \overset{?}{<} r+s$

t	n	s	r	r+s (adder)	STOP	SF	STATO PRESENTE
0↑ 0↓	1 1 1 0 1 0 1 1 1 0 0 0 = 3768	0 1 0 0 0 0 0 0 0 0 0 0 0 1024	0000 0 0 0 0 0 0 0 0 = ∅	0 1 0 0 0 0 0 0 0 0 0 0 0 = 1024	∅	∅	011
1↑ 1↓	1 0 1 0 1 0 1 1 1 0 0 0 = 2744 = 3768 − 1024		0000 0 0 0 0 0 0 0 0 = ∅	0 1 0 0 0 0 0 0 0 0 0 0 0 = 1024			100
2↑ 2↓			0100 0 0 0 0 0 0 0 0 = 1024	(1 0 0 0 0 0 0 0 0 0 0 0 0 = 2048) → non usato			101
3↑ 3↓		0 0 1 0 0 0 0 0 0 0 0 0 0 = 512		(0 1 1 0 0 0 0 0 0 0 0 0 0 = 1536) → non usato			010
4↑ 4↓		0 0 0 1 0 0 0 0 0 0 0 0 0 = 256		0 1 0 1 0 0 0 0 0 0 0 0 0 = 1280	∅	∅	011
5↑ 5↓	0 1 0 1 1 0 1 1 1 0 0 0 = 1464 = 2744 − 1280		0010 0 0 0 0 0 0 0 0 = 512	0 0 1 1 0 0 0 0 0 0 0 0 0 = 768			100
6↑ 6↓			0011 0 0 0 0 0 0 0 0 = 768	(0 1 0 0 0 0 0 0 0 0 0 0 0 = 1024)			101
7↑ 8↓		0 0 0 0 0 1 0 0 0 0 0 0 0 = 64		0 0 1 1 0 1 0 0 0 0 0 0 0 = 832	∅	∅	010 & 011
9↑ 9↓	0 0 1 0 0 1 1 1 1 0 0 0 = 632 = 1464 − 832		0001 1 0 0 0 0 0 0 0 = 384	0 0 0 1 1 1 0 0 0 0 0 0 0 = 448			100
10↑ 10↓			0001 1 1 0 0 0 0 0 0 = 448	(0 0 1 0 0 0 0 0 0 0 0 0 0 = 512)			101
11↑ 12↓		0 0 0 0 0 0 0 0 1 0 0 0 0 = 16		0 0 0 1 1 1 0 1 0 0 0 0 0 = 464	∅	∅	010 & 011
13↑ 13↓	0 0 0 0 1 0 1 0 1 0 0 0 = 168 = 632 − 464		0000 1 1 1 0 0 0 0 0 = 224	0 0 0 0 1 1 1 1 0 0 0 0 0 = 240			100
14↑ 14↓			0000 1 1 1 1 0 0 0 0 = 240	(0 0 0 1 0 0 0 0 0 0 0 0 0 = 256)			101
15↑ 16↓		0 0 0 0 0 0 0 0 0 0 1 0 0 = 4		0 0 0 0 1 1 1 1 0 1 0 0 = 244	∅	1	010 & 011
17↑ 17↓			0000 0 1 1 1 1 0 0 0 = 120				110
18↑ 19↓		0 0 0 0 0 0 0 0 0 0 0 0 1 = 1		0 0 0 0 0 1 1 1 1 0 0 1 = 121			010 & 011
20↑ 20↓	0 0 0 0 0 0 1 0 1 1 1 1 = 47 = 168 − 121 (= b)		0000 0 0 1 1 1 1 0 0 = 60	← 0 0 0 0 0 0 1 1 1 1 0 1 = 61 (= a in r al prossimo clock)		∅	100