Università degli Studi di Verona
Dipartimento di Informatica
Corso di laurea in Informatica

Anno accademico 2016/2017

ELABORATO DI ARCHITETTURA DEGLI ELABORATORI

RELAZIONE

Ottimizzazione di un codice in linguaggio C mediante l'uso di Assembly

STUDENTI:
De Marchi Mirco - VR408481
Guadagnini Filippo - VR408802

INDICE:

Descrizione del codice

Il codice riceve come input un file di estensione txt contenente una stringa che descrive per ogni riga la variazione del pH di una soluzione contenuta in un serbatoio e il programma fornisce in uscita su un altro file txt una stringa contenete per ogni riga la corrispondente variazione di stato della soluzione in termini di acido (A), basico (B) e neutro (N).
Le stringa contenuta nel file di input viene elaborata dal programma e riportata all'interno della stringa bufferin, poi il codice si divide in 2 parti: la prima è l'esecuzione in C, che in base alla bufferin elabora la stringa bufferout_c; la seconda è l'esecuzione in Assembly inline, che invece elabora la stringa bufferout_asm.
Il sistema deve portare sempre la soluzione allo stato neutro, accettando un transitorio di 5 cicli di clock; pertanto si richiede che al sesto ciclo di clock in cui il sistema sia allo stato A, venga aperta una valvola con soluzione basica (BS), e analogamente se allo stato B si apra la valvola con soluzione acida (AS). Il sistema deve inoltre fornire in uscita il numero di cicli di clock da cui si trova nello stato attuale.

Ogni riga del file in input contiene i seguenti valori:
INIT , RESET , PH2 PH1 PH0

Ogni riga del file di output deve contenere i seguenti valori:
ST , NCK1 NCK0 , VLV1 VLV0

- INIT[1]: quando è 1 il sistema è acceso, quando è a 0 il sistema deve essere resettato e quindi restituire una stringa composta da soli '-'.
- RESET[1]: quando posto a 1 deve restituire tutti '-'.
- PH [3]: valore del pH misurato dal rilevatore. Il range di misura è compreso tra 0 e 14 con risoluzione di 0,1.

- ST[1]: 'A' (acida) 'B' (basica) 'N' (neutra).
- NCK[2]: indica il numero di cicli di clock trascorsi nello stato corrente.Diagramma degli stati del controllore.
- VLV [2]: indica quale valvola aprire per riportare la soluzione allo stato neutro nel caso in cui la soluzione si trovi da più di 5 cicli di clock in stato acido (BS) o basico (AS).

Di seguito mostro un esempio di stringa contenuto nel file di input (a sinistra) e il corrispondente contenuto del file di output (a destra).

testin.txt	testout.txt
0,0,000	-,--,--
0,0,100	-,--,--
0,0,040	-,--,--
0,0,068	-,--,--
0,1,069	-,--,--
0,0,072	-,--,--
0,0,086	-,--,--
1,0,070	N,00,--
1,0,068	N,01,--
1,0,045	A,00,--
1,0,045	A,01,--
1,0,045	A,02,--
1,0,045	A,03,--
1,0,045	A,04,--
1,0,045	A,05,BS
1,0,045	A,06,BS
1,0,045	A,07,BS
1,1,054	-,--,--
1,0,056	A,00,--
1,0,056	A,01,--
1,0,056	A,02,--
1,0,067	N,00,--
1,0,067	N,01,--
1,0,067	N,02,--
1,0,067	N,03,--
1,0,067	N,04,--
1,0,067	N,05,--
1,0,067	N,06,--
1,0,080	N,07,--
1,0,090	B,00,--
1,0,090	B,01,--
1,0,090	B,02,--
1,0,090	B,03,--
1,0,090	B,04,--
1,0,090	B,05,AS
1,0,090	B,06,AS
1,0,090	B,07,AS
1,0,033	A,00,--
1,0,033	A,01,--

La parte di codice in C e la parte di codice in ASM sostanzialmente producono lo stesso identico risultato, l'unica differenza sta nella ottimizzazione del codice e quindi nella velocità di esecuzione.

Da terminale dopo aver utilizzato il makefile per compilare il programma, l'esecuzione avviene cosi:

./controller testin.txt testout.txt

Assembly: funzioni e passaggio parametri

Le funzioni utilizzate sono le seguenti:

1. ciclocontrollore_asm

Questa funzione si occupa di ciclare la funzione "controllore" fino a quando finisce la stringa bufferin. Aggiorna di volta in volta i registri ESI e EDI che contengono l'indirizzo della stringa rispettivamente da leggere e da modificare, in modo tale che la funzione "controllore" possa agire trovando sempre su ESI una riga del tipo INIT,RESET,PH e su EDI una riga del tipo ST,NCK,VLV.

2. controllore_asm

Questa funzione considera come input il registro ESI contenente l'indirizzo a una riga costruita come INIT,RESET,PH, e modifica la riga indirizzata da EDI, costituita da ST,NCK,VLV, in base ai valori espressi dalla riga in input.
Questa funzione inoltre richiama in sé le funzioni "duecifre2str_asm" e "ph2num_asm".

3. duecifre2str_asm

Converte il numero del clock, contenuto in EBX, in una stringa e la inserisce in NCK[2].
Questa funzione è stata fatta con la limitazione a 2 cifre e non con una qualsiasi cifra in input per ottimizzare la funzione "controllore" per la conversione del clock (contenuto in EBX) in una stringa. Infatti la parte di output NCK può contenere al massimo 2 cifre, di conseguenza è inutile rendere questa funzione compatibile con numeri a 3 cifre o più. Tuttavia nel caso in cui il clock fosse a più di 2 cifre, la funzione inserirà in NCK la stringa "??".

4. ph2num_asm

Converte i 3 caratteri numerici contenuti in PH, in numero e lo salva su EAX.
Questa funzione prevede in ingresso 3 caratteri di PH (PH2 PH1 PH0), che poi devono essere interpretati come un numero.
Questa funzione serve per ottimizzare la conversione da carattere a numero nella funzione "controllore", infatti per questa operazione non serve realizzare una funzione che opera anche con stringhe formate da più di 3 caratteri, perchè il PH in input è dato esattamente da 3 caratteri.

Passaggio di parametri:

Riguardo al passaggio dei parametri invece bisogna specificare che il codice C non contiene codice Assembly inline, ma solo una chiamata a una funzione Assembly di tipo *extern*. Questa scelta è stata fatta per rendere il codice più leggibile.
Infatti abbiamo trovato piuttosto scomoda e poco elegante la sintassi dell'Assebly inline e abbiamo preferito utilizzare la classica sintassi su un file separato richiamato da un *extern modules*.

L'inizializzazione della funzione avviene nel seguente modo:

```
extern void ciclocontrollore_asm( char bufferin[], char bufferout_asm[]);
```

E la funzione viene richiamata così:

```
ciclocontrollore_asm( bufferin, bufferout_asm );
```

Di conseguenza i parametri bufferin e bufferout_asm sono passati tramite lo stack.
Inoltre proprio all'inizio della funzione "ciclocontrollore_asm" eseguo le seguenti istruzioni:

```
pushl %ebp
movl %esp, %ebp
```

In questo modo salvo sullo stack EBP, così da poterlo modificare e fargli assumere proprio il valore attuale di ESP.
A questo punto lo stack si presenta nel seguente modo:

EBP —>	
4 + EBP	EBP
8 + EBP	Bufferin
12 + EBP	Bufferout_asm
16 + EBP	…..

Quindi se voglio salvare bufferin in ESI e bufferout_asm in EDI devo inserire le seguenti istruzioni:

```
movl 8(%ebp), %esi
movl 12(%ebp), %edi
```

Riguardo invece alla chiamata a funzione delle funzioni "controllore_asm", "duecifre2str_asm" e "ph2num_str", non esiste un vero e proprio passaggio dei dati.
Infatti queste funzioni sono state create semplicemente per rendere il codice più leggibile e di immediata comprensione.
Per questo in realtà il codice di queste funzioni potrebbe benissimo essere copiato e incollato al posto della sua rispettiva CALL, e il programma funzionerebbe comunque senza intoppi.
Dunque non esiste un passaggio dei dati per queste specifiche chiamate a funzioni, ma semplicemente utilizza i registri come già sono stati impostati precedentemente per compiere l'operazione desiderata.

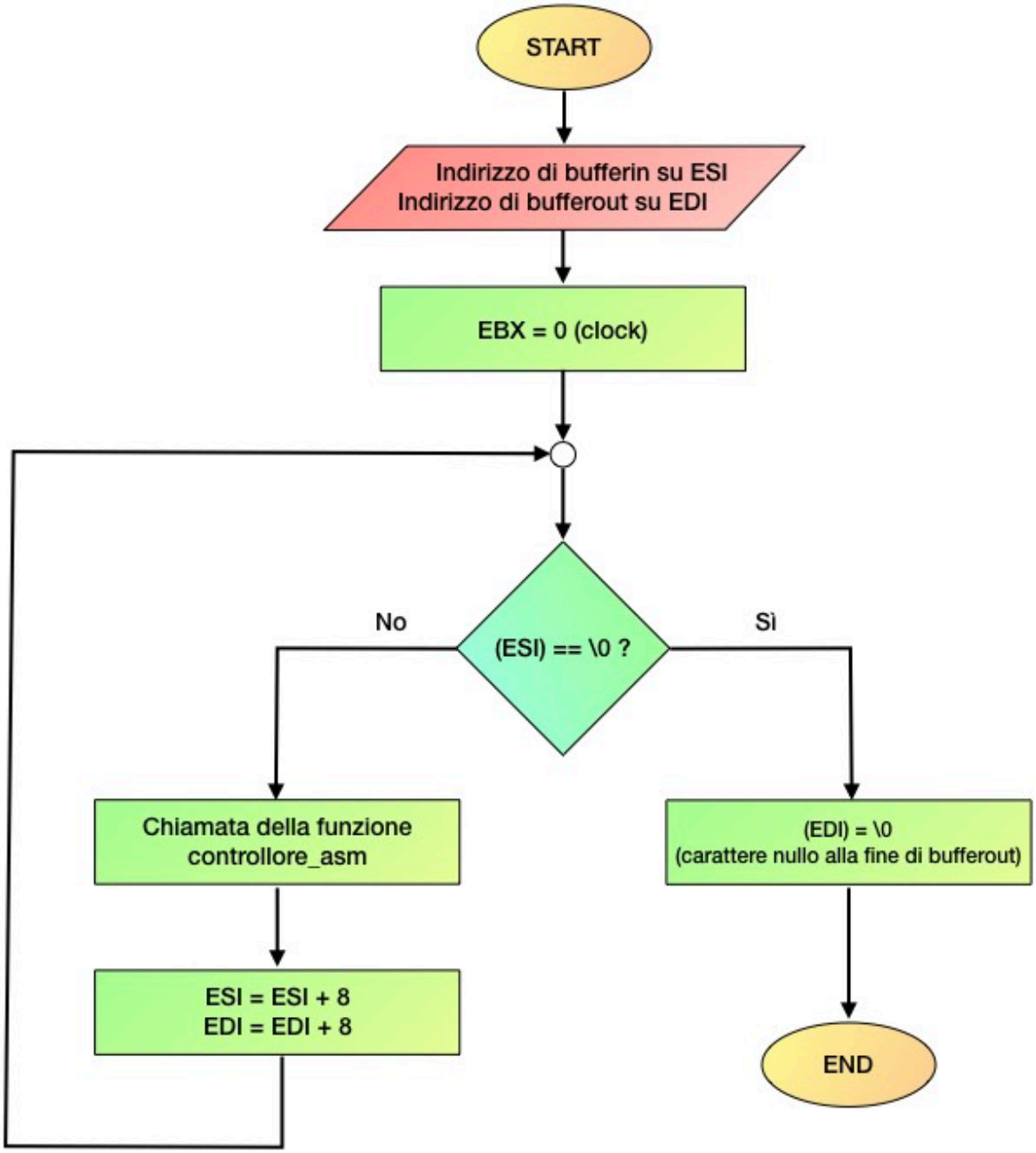
Utilizzo dei registri

Nel codice che abbiamo sviluppato non utilizziamo variabili. Questa scelta ci permette di ottimizzare il codice.
Tuttavia i registri sono fondamentali, perché ognuno ha la propria funzione:

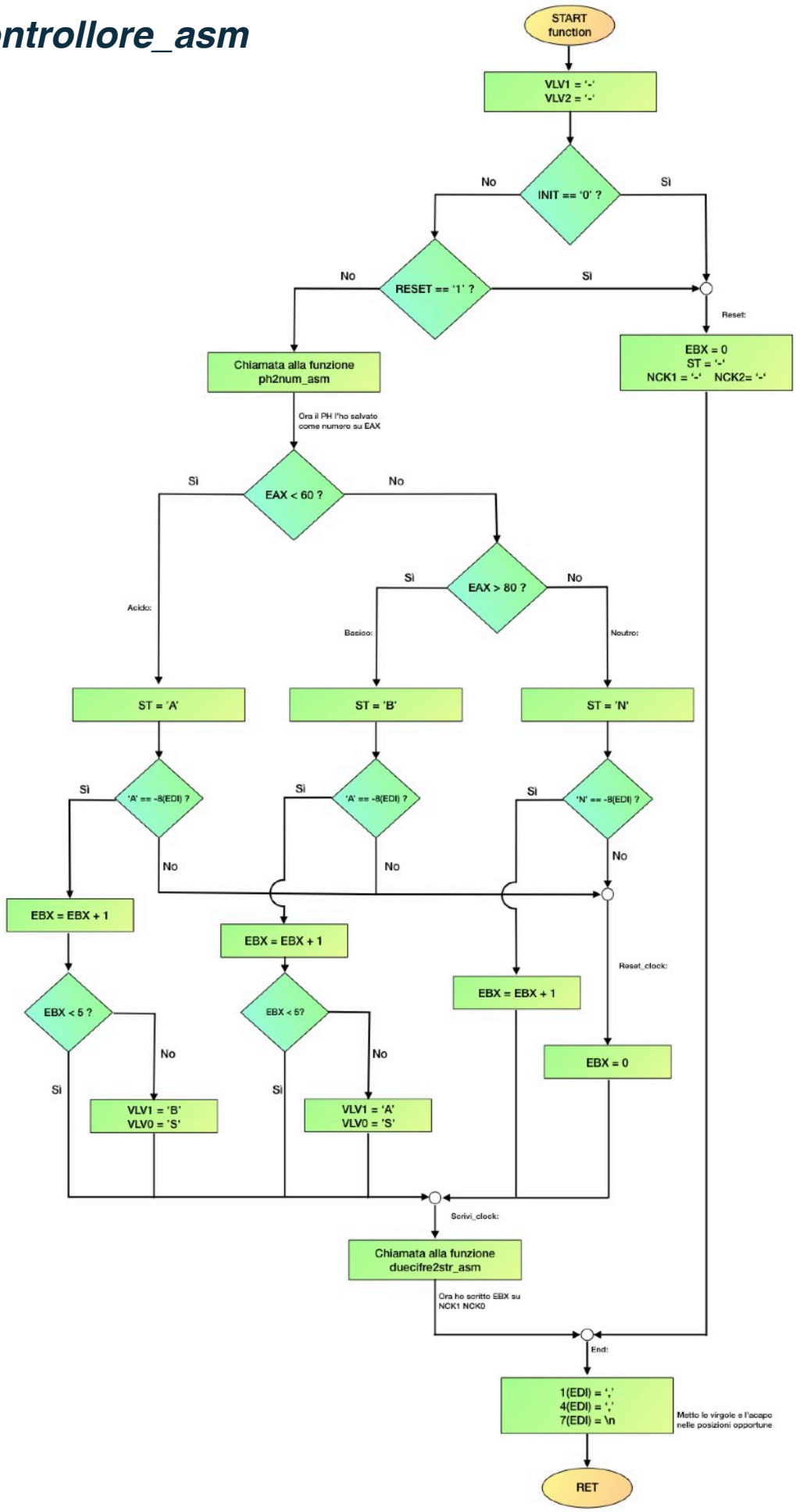
- EAX contiene il valore del ph della soluzione, in modo tale che il confronto per il passaggio di stato avvenga con l'utilizzo di questo registro;
- EBX contiene il numero di cicli di clock passati nello stesso stato per un determinato periodo, in modo che quando questo venga confrontato con il numero 5 (ovvero il numero massimo di volte in cui la soluzione può stare in un determinato stato) si possa aprire una valvola come da consegna;
- ECX non viene utilizzato;
- EDX è un registro libero che serve ad esempio per contenere la costante 10 per fare la moltiplicazione nella conversione da stringa a numero.
- ESI è il registro che contiene l’indirizzo alla stringa “bufferin”.
- EDI è il registro che contiene l’indirizzo alla stringa “bufferout".

Diagramma di Flusso

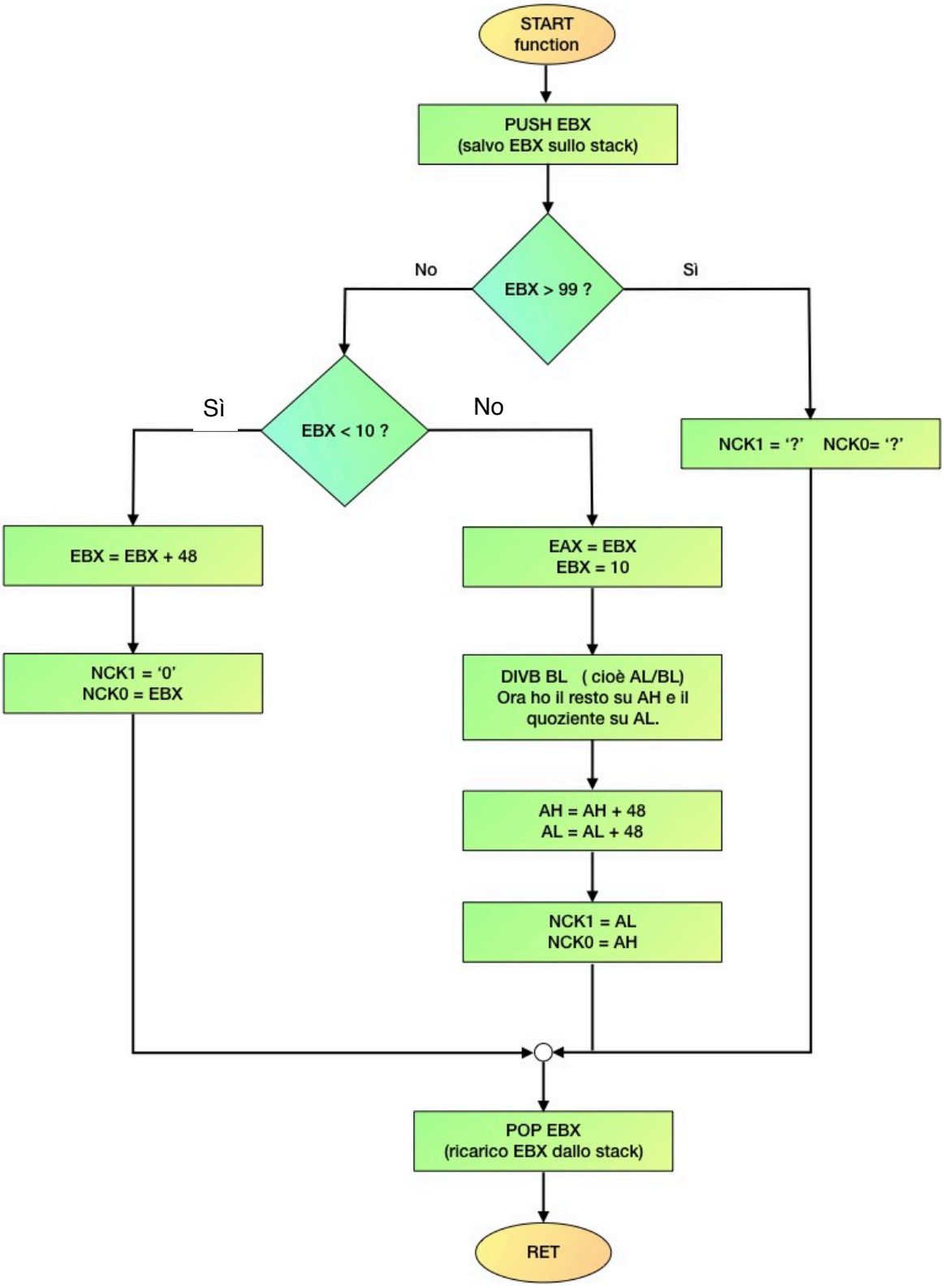
ciclocontrollore_asm



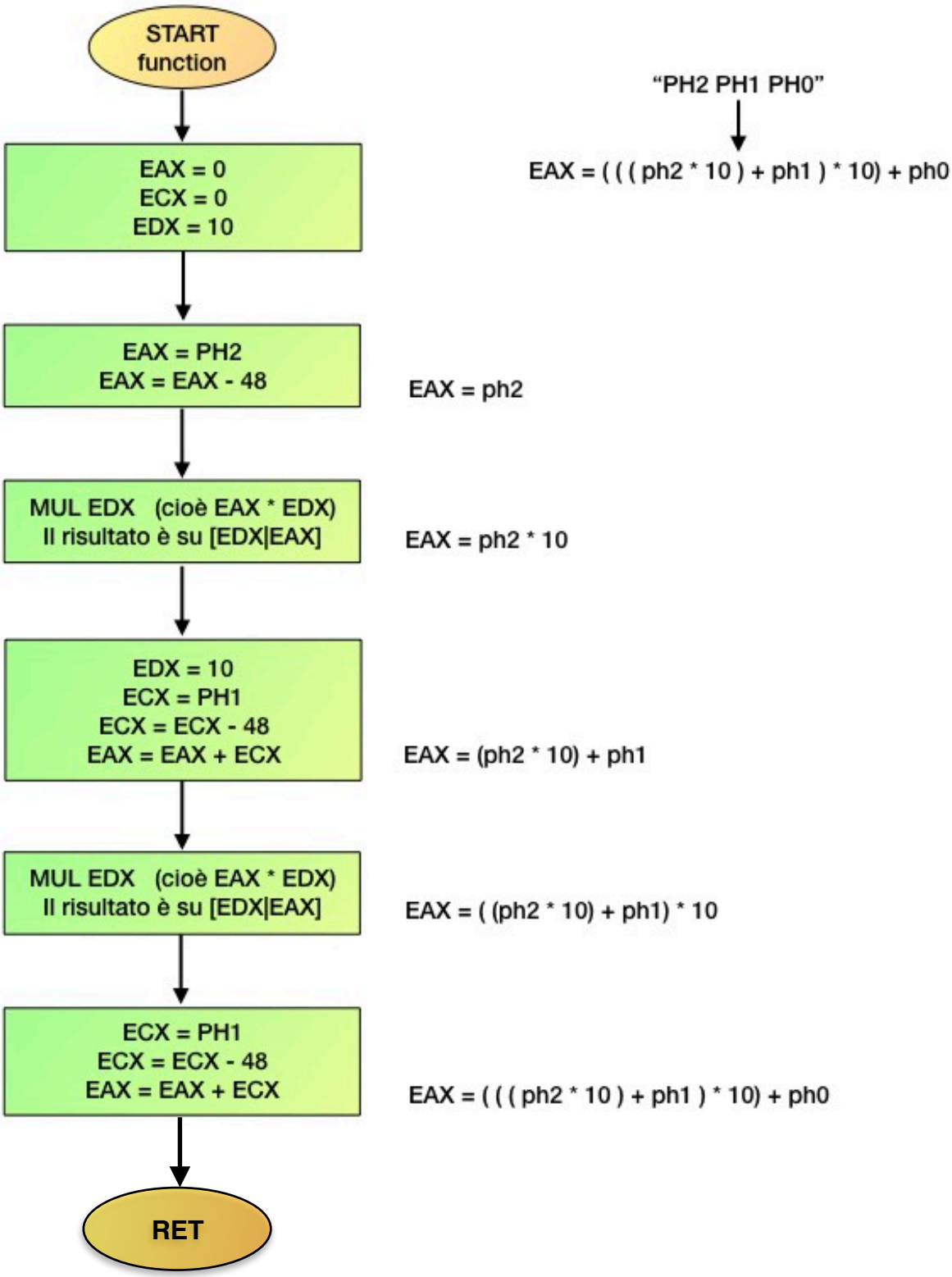
controllore_asm



duecifre2str_asm



ph2num_asm


START
function
EAX = 0
ECX = 0
EDX = 10
EAX = PH2
EAX = EAX - 48
MUL EDX (cioè EAX * EDX)
Il risultato è su [EDX|EAX]
EDX = 10
ECX = PH1
ECX = ECX - 48
EAX = EAX + ECX
MUL EDX (cioè EAX * EDX)
Il risultato è su [EDX|EAX]
ECX = PH1
ECX = ECX - 48
EAX = EAX + ECX
RET
"PH2 PH1 PH0"
EAX = (((ph2 * 10) + ph1) * 10) + ph0
EAX = ph2
EAX = ph2 * 10
EAX = (ph2 * 10) + ph1
EAX = ((ph2 * 10) + ph1) * 10
EAX = (((ph2 * 10) + ph1) * 10) + ph0

N°	Tempo di esecuzione in C	Tempo di esecuzione in ASM
1	19475 ns	829 ns
2	14103 ns	835 ns
3	21890 ns	865 ns
4	18027 ns	870 ns
5	13832 ns	923 ns
6	18722 ns	834 ns
7	17086 ns	837 ns
8	13756 ns	849 ns
9	20784 ns	796 ns
10	17384 ns	972 ns
11	16682 ns	821 ns
12	13900 ns	857 ns
13	18408 ns	914 ns
14	21358 ns	841 ns
15	19715 ns	849 ns
16	22076 ns	855 ns
17	14673 ns	844 ns
18	17677 ns	762 ns
19	14535 ns	784 ns
20	16403 ns	851 ns
21	17951 ns	777 ns
22	14790 ns	757 ns
23	18746 ns	833 ns
24	17550 ns	765 ns
25	18207 ns	765 ns
26	21484 ns	780 ns
27	14790 ns	851 ns
28	18930 ns	769 ns

Tempi di esecuzione C e Assembly

Elaborazione dei dati

Media tempo di esecuzione in C: ***17604***

Media tempo di esecuzione in ASM: ***831***

Miglioramento medio: 17604/831 = ***21,2***

Quindi il programma in Assembly è 21,2 volte più veloce del programma in C.

Facendo un miglioramento percentuale:

100 - ((831/17604) * 100) = ***95,28%***

Il programma in Assembly è del 95,28% più veloce del programma in C.

Scelte progettuali

1. ***Nessuna variabile***

Nel nostro codice abbiamo deciso di non utilizzare alcuna variabile, ma utilizzare solo registri. Questo permette al nostro programma una esecuzione molto più veloce, poiché se avessimo utilizzato qualche variabile, per una qualsiasi operazione fatta su di esse, avremmo fatto un accesso alla memoria RAM e questo avrebbe rallentato di tantissimo l'esecuzione del nostro codice.
Visto che l'accesso ai registri occupa un tempo davvero inferiore rispetto all'accesso a una zona di memoria, questa nostra decisione rende il nostro codice molto veloce e ottimizzato.

2. ***No Assembly inline, solo extern modules***

Abbiamo deciso di non inserire codice Assembly all'interno del codice C e di utilizzare solo funzioni extern per motivi principalmente estetici e di comprensione del codice.
Infatti, facendo ulteriori esercizi, abbiamo notato che l'Assembly inline è utile e di immediato utilizzo solo nel momento in cui dobbiamo utilizzarlo per un codice molto breve. Invece nel caso nostro, avendo un codice molto lungo, abbiamo preferito evitare di scrivere l'intero codice utilizzando la sintassi del Assembly inline, a nostro parere molto ripetitiva e poco elegante, e quindi di richiamare il nostro codice con un extern module, mantenendo dunque la sintassi classica del Assembly.

3. ***Modalità di lettura della stringa***

Per leggere la stringa modifico gli indirizzi salvati su ESI e EDI solo quando devo saltare da una riga all'altra. Infatti ogni volta che finisce il giro di controllo di tutta la riga l'operazione che faccio è la seguente:

ESI = ESI + 8
EDI = EDI + 8

Ogni riga è formata da 8 caratteri, dunque l'indirizzo ESI e EDI vengono incrementati di 8. Invece quando devo accedere ad un specifico carattere della riga uso come metodo di indirizzamento un indirizzamento indiretto con spiazzamento.
La gestione dei caratteri avviene in questo modo:

INIT	,	RESET	,	PH2	PH1	PH0	\n
(ESI)	1(ESI)	2(ESI)	3(ESI)	4(ESI)	5(ESI)	6(ESI)	7(ESI)

ST	,	NCK1	NCK0	,	VLV1	VLV0	\n
(EDI)	1(EDI)	2(EDI)	3(EDI)	4(EDI)	5(EDI)	6(EDI)	7(EDI)

4. ***Clock maggiore di 99***

Se il clock arriva a un valore maggiore di 99, allora verrà inserito su NCK[2] la stringa "??".