



Università degli Studi di Verona  
Dipartimento di Informatica  
AA 2016/2017  
Corso di laurea in Informatica

# Elaborato ASM

Laboratorio di Architettura degli Elaboratori

Agresti Nicola VR407685  
Piccinelli Marco VR407621

# Indice

Specifiche.....	3
Descrizione etichette ASM.....	4
Diagramma di flusso.....	6
Esecuzione.....	7
Scelte progettuali.....	8

# Specifiche

Elaborazione che effettua il monitoraggio di un impianto chimico industriale mediante l'uso di Assembly inline. Riceve come input il pH di una soluzione contenuta in un serbatoio e, una volta impostate le soglie minima e massima per il funzionamento ottimale, il programma fornisce in uscita lo stato della soluzione in termini di acido (*A*), basico (*B*) e neutro (*N*). 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.

Il programma deve leggere il contenuto di una stringa (*bufferin*) contenente in ogni riga i seguenti valori:

INIT,RESET,PH

- **INIT** [1]: valore binario, quando vale **1** il sistema è acceso; quando vale **0** il sistema è spento e deve restituire una linea composta da soli **0**.
- **RESET** [1]: quando posto a **1** il controllore deve essere resettato, ovvero tutte le uscite devono essere poste a **0** e il sistema riparte.
- **PH** [3]: valore del pH misurato dal rilevatore. Il range di misura è compreso tra **0** e **14** con risoluzione di **0,1**. Il valore è espresso in decimi di pH e sempre riportato in 3 cifre, ad esempio 065 corrisponde a 6,5.

Il programma deve restituire i risultati del calcolo in una stringa (*bufferout\_asm*) in cui ogni riga contiene:

ST,NCK,VLV

- **ST** [1]: indica in quale stato si trova la soluzione al momento corrente (acida - *A*, basica - *B* o neutra - *N*)
- **NCK** [2]: indica il numero di cicli di clock trascorsi nello stato corrente.
- **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*).

Le soglie del pH sono le seguenti:

- $\text{pH} < 6.0$  : Acido
- $6.0 \leq \text{pH} \leq 8.0$  : Neutro
- $\text{pH} > 8.0$  : Basico

# Descrizione etichette ASM

Nel file controller.c abbiamo la funzione `__asm__` che contiene il codice assembly inline che esegue l'elaborazione descritta nel paragrafo precedente. Di seguito sono spiegate tutte le etichette presenti.

**INIZIO:** salvo l'indirizzo di bufferin nel registro ESI (source index) e quello di bufferout nel registro EDI (destination index). Utilizzo la xor per azzerare il contenuto di EBX in modo da non avere problemi nel comparare tra lo stato attuale e quello precedente che avviene all'inizio dell'etichetta NCK la prima volta che la macchina è accesa e non è in reset.

**INCBUFFER:** incrementa gli indirizzi di bufferin e bufferout salvati nei registri ESI ed EDI per scorrere le stringhe.

**COMPARE0:** controlla che il primo carattere di ogni "riga" del bufferin sia diverso da `\0` (codice ASCII 0). Se è uguale si va all'etichetta END e l'elaborazione finisce.

**RIEMPI:** viene riempita la prima "riga" del bufferout con la sequenza `-,--,--\n`.

STRINGA	ASCII	BINARIO	ESADECIMALE
-,--	45-45-44-45	00101101001011010010110000101101	0x2D2D2C2D
-,--\n	10-45-45-44	00001010001011010010110100101100	0xA2D2D2C

**INIT\_RESET:** il carattere in posizione zero del bufferin corrisponde a INIT. Il carattere in posizione due corrisponde a RESET. Se INIT è uguale a zero oppure RESET è uguale ad uno ci si muove all'etichetta RESETMACCHINA. Altrimenti, si prosegue verso l'elaborazione del pH.

**RESETMACCHINA:** resetto il registro ebx, contenente lo stato vecchio e mi muovo verso l'etichetta INCBUFFER. Quest'etichetta serve per gestire i reset della macchina quando l'elaborazione è già stata fatta almeno una volta. Se non ci fosse, il contatore dei clock potrebbe non essere resettato correttamente.

**PH:** il carattere della quarta posizione corrisponde alle decine del pH. Se è maggiore di 0 allora il pH è sicuramente basico ( $\text{pH} = 10.0$ ). Se non è stata effettuata la jump precedente passo a controllare l'unità del pH che corrisponde al carattere in quinta posizione. Se questo è maggiore di 8 il pH è sicuramente basico ( $\text{pH} > 8.00$ ). Se è uguale ad 8 passo all'etichetta BASNEU. Altrimenti vado a verificare che l'unità sia minore di 6. Se è minore di 6 il pH è sicuramente acido ( $\text{pH} < 6.0$ ). Se nessuna delle jump precedenti è stata effettuata, il pH è neutro ( $6.0 \leq \text{pH} < 8.0$ ).

**BASNEU:** l'unità del pH è uguale ad 8 quindi potrebbe essere sia basico che neutro. Si procede quindi ad analizzare il decimo, che corrisponde al carattere in sesta posizione. Se è maggiore di 0 il pH è basico altrimenti è neutro e ci si muove nelle etichette opportune.

**BASICO, ACIDO, NEUTRO:** le tre etichette hanno la stessa struttura. Viene salvato il corrispettivo ASCII del carattere dello stato nel registro AL. (65 lettera A, 66 lettere B, 78 lettera N). Viene poi scritto nella prima posizione del bufferout.

**NCK:** l'etichetta inizia comparando i registri EAX ed EBX. Il primo contiene lo stato attuale della sostanza mentre il secondo lo stato misurato nel ciclo precedente. Se gli stati sono diversi all'ora si passa a RESNCK. Altrimenti, si incrementano le unità del contatore dei cicli di clock, salvata nel registro CH (parte alta del registro a 16n bit CX). Una volta incrementata l'unità, si fa un controllo e, se il contatore arriva a 10 ci si muove all'etichetta INCDECINA. Se invece non è dieci, si va direttamente a SCRIVINCK.

**INCDECINA:** incrementa la decina del contatore salvata nella parte bassa del registro CX e si resetta la parte alta del registro, che contiene l'unità.

**RESNCK:** viene resettato il contatore dei cicli di clock.

STRINGA	ASCII	BINARIO	ESADECIMALE
00	48-48	0011000000110000	0x3030

**SCRIVINCK:** scrive il contenuto del registro CX sul bufferout. Esso contiene il numero dei cicli di clock.

**SALVASTATO:** lo stato attuale viene salvato nel registro che contiene lo stato del ciclo precedente. Quest'etichetta servirà per i cicli futuri.

**CHECKCH:** controlla che l'unità dei cicli di clock sia maggiore di 4 ed eventualmente passa all'etichetta con il controllo dell'apertura delle valvole.

**CHECKCL:** controlla che la decina dei cicli di clock sia maggiore di 1. Se non lo è torna a INCBUFFER per iniziare un nuovo ciclo. Se si arriva in questa etichetta si ha un contenuto minore di 4 nel registro dove è salvato il contatore delle unità. Senza quest'ulteriore controllo, in un ipotetico ciclo 13, per esempio, la valvola sarebbe chiusa.

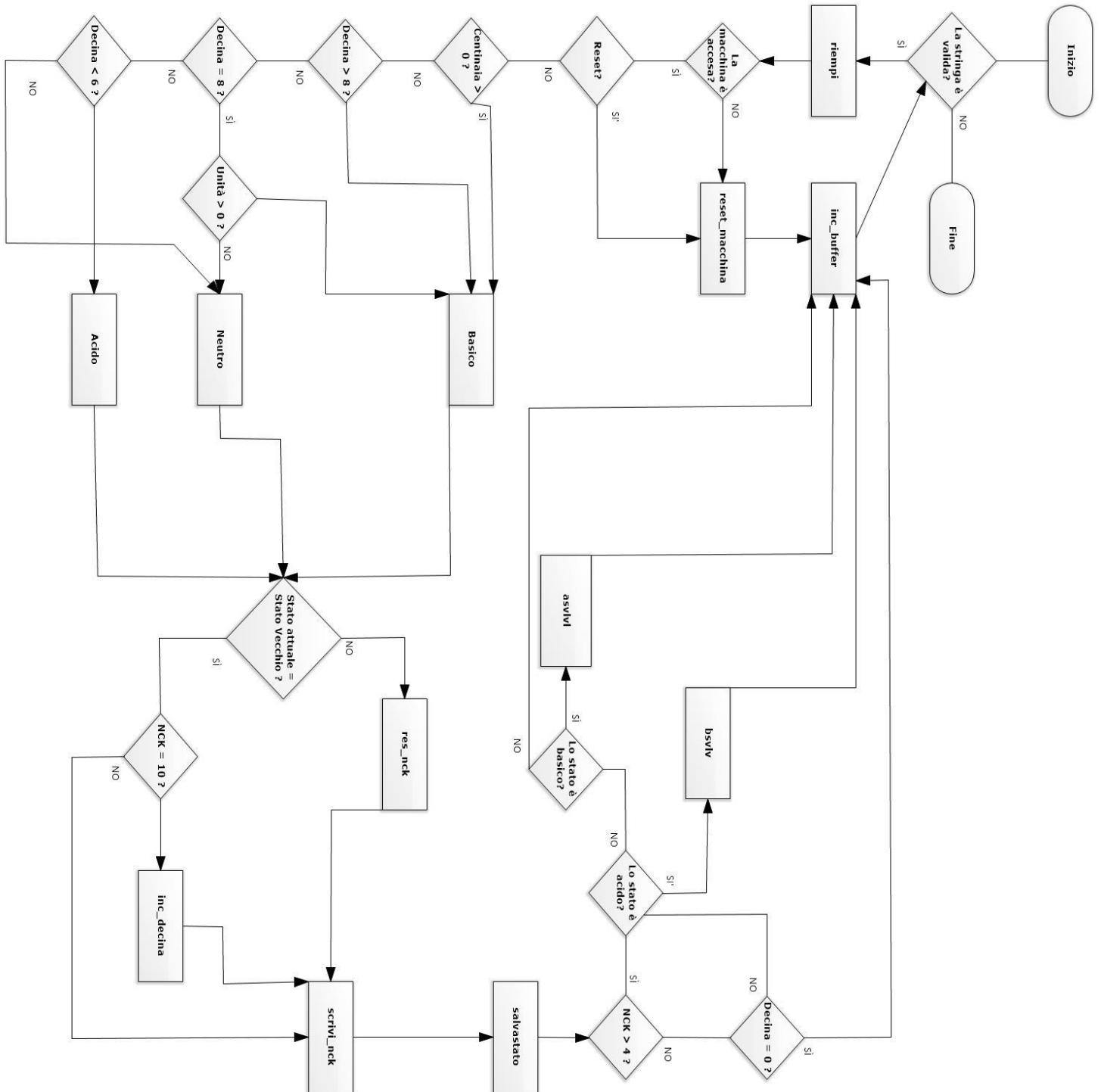
**VLVCHECK:** a questo punto, si torna a controllare il contenuto di AL per vedere lo stato della sostanza. Se lo stato è neutro allora si torna ad INCBUFFER per iniziare un nuovo ciclo, altrimenti, si va ai corrispettivi BSVLV oppure ASVLV in base allo stato.

**BSVLV, ASVLV:** vado a scrivere sul bufferout la stringa "BS" oppure la stringa "AS" in base allo stato misurato.

STRINGA	ASCII	BINARIO	ESADECIMALE
BS	83-66	0101001101000010	0x5342
AS	83-65	0101001101000001	0x5341

**END:** se si arriva a questo punto, l'elaborazione termina.

# Diagramma di flusso

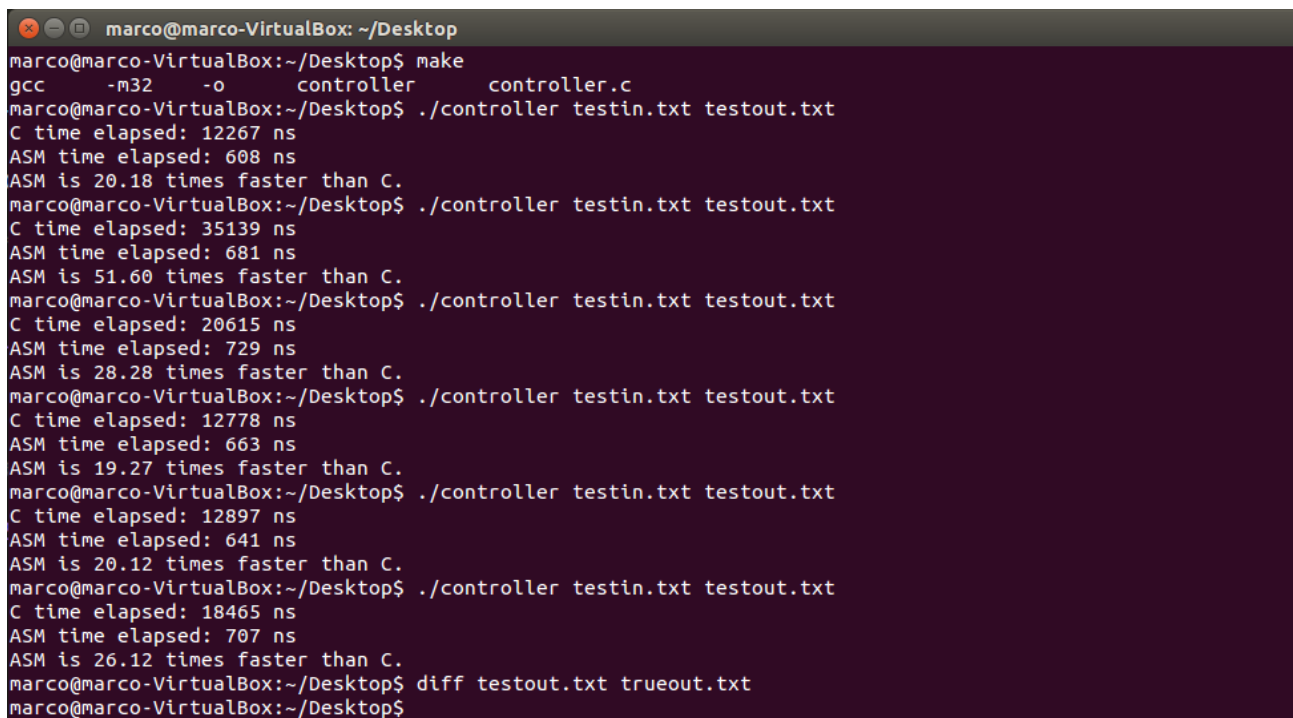


# Esecuzione

Di seguito vi è uno screenshot dell'esecuzione. È stata inserita una printf alla riga 151 del file controller.c privo di modifiche. È utile per avere un confronto immediato delle velocità. Nel file consegnato non sarà ovviamente presente.

La riga inserita è la seguente:

```
printf("ASM is %.2f times faster than C.\n", ((float)c_time_in_nanos/(float)asm_time_in_nanos));
```



```
marco@marco-VirtualBox: ~/Desktop
marco@marco-VirtualBox:~/Desktop$ make
gcc -m32 -o controller controller.c
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 12267 ns
ASM time elapsed: 608 ns
ASM is 20.18 times faster than C.
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 35139 ns
ASM time elapsed: 681 ns
ASM is 51.60 times faster than C.
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 20615 ns
ASM time elapsed: 729 ns
ASM is 28.28 times faster than C.
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 12778 ns
ASM time elapsed: 663 ns
ASM is 19.27 times faster than C.
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 12897 ns
ASM time elapsed: 641 ns
ASM is 20.12 times faster than C.
marco@marco-VirtualBox:~/Desktop$ ./controller testin.txt testout.txt
C time elapsed: 18465 ns
ASM time elapsed: 707 ns
ASM is 26.12 times faster than C.
marco@marco-VirtualBox:~/Desktop$ diff testout.txt trueout.txt
marco@marco-VirtualBox:~/Desktop$
```

Come possiamo vedere l'elaborazione di ASM è mediamente 20 volte più veloce a quella di C. Abbiamo escluso le situazioni in cui la parte in C ha registrato tempi di molto superiori alla media.

# Scelte progettuali

È stato utilizzato il registro CX a 16 bit per il contatore dei cicli di clock ma durante l'elaborazione lo abbiamo modificato anche tramite la parte alta e quella bassa del registro (registri CH e CL a 8 bit). Per andare ad isolare i singoli caratteri del bufferin abbiamo utilizzato dei registri a 8 bit. In EBX è salvato lo stato del ciclo precedente mentre EAX è usato per le varie istruzioni e come stato del ciclo attuale. Inoltre nei registri source index (ESI) e destination index (EDI) vi sono memorizzati gli indirizzi di bufferin e bufferout.

Abbiamo impostato il contatore dei cicli di clock con i caratteri ASCII corrispondenti ai numeri (lo 0 è 48). In questo modo ci evitiamo istruzioni inutili durante la fase di lettura di bufferin e a sua volta durante la fase di scrittura del bufferout. Per questo motivo quando CH è resettato viene utilizzata come istruzione `mov $48,%%ch` e non una `xor` come ad esempio viene fatto nella prima etichetta del codice.

Inoltre, per avere un codice più corto e con meno istruzioni, abbiamo deciso di passare le stringhe in output unendo i vari caratteri anziché passarli uno per uno. Nelle tabelle sono presenti i valori ASCII in binario ed in esadecimale, notazione che abbiamo poi utilizzato nel codice.