ELABORATO ASM
ARCHITETTURA DEGLI ELABORATORI

A.A. 2016/2017 - Corso di Laurea in Informatica

Verona, 15 Luglio 2017

Chiarani Fabio (VR408541)

SOMMARIO

SOMMARIO 2
INTRODUZIONE 3
SPECIFICHE 4
IPOTESI AGGIUNTIVE 5
Valore NCK: 5
Range del pH: 5
Ottimizzazione del codice per prestazioni: 5
Extern Modules o ASM Inline? 5
STRUTTURA GENERALE 6
VARIABILI & REGISTRI 7
STRUTTURA DI BUFFERIN E BUFFEROUT 8
FLUSSO DEL PROGRAMMA 10
FLOW CHART 11
DESCRIZIONE ETICHETTE 12

INTRODUZIONE

La relazione viene fornita assieme a tutti i sorgenti del progetto.
Il progetto è così strutturato:

- *./ (root)*: cartella in cui sono presenti tutti i sorgenti ottimizzati, e per questo, molti potrebbero non essere comprensibili;

I file più importanti contenuti nelle rispettive sottodirectory sono:

- *CONTROLLER.C* : è il controllore in C a cui si fa riferimento come tempo principale;
- *CONTROLLER_ASM.S* : è il controllore C riscritto in assembly (ASM);
- *MAKEFILE* : è il file necessario a compilare tutti i sorgenti in un unico comando;
- *RELAZIONE.PDF* : è la relazione del progetto;

SPECIFICHE

Si ottimizzi un codice in linguaggio C che effettua il monitoraggio di un impianto chimico industriale mediante l'uso di Assembly inline. Ricevendo 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 essere lanciato da riga di comando con due stringhe come parametri, la prima stringa identifica il nome del file .txt da usare come input, la seconda quello da usare come output:

```
$ ./controller testin.txt testout.txt
```

INPUTS:

- ***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.

OUTPUTS:

- ***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).

IPOTESI AGGIUNTIVE

Valore NCK:

Da specifiche il valore del *NCK* e' riportato su due char, questo limita a un range che va da 00 a 99. Per questo e' stata fatta la scelta progettuale di andare in overflow nel caso in cui si arrivi a un valore di *NCK* superiore a 99.

Range del pH:

E' stata fatta una scelta progettuale per cui il controller *ASM* vede come pH basico un qualsiasi valore ricevuto in input che sia maggiore o uguale a 1xx.

Ottimizzazione del codice per prestazioni:

E' stata fatta la scelta progettuale di rendere il codice il più possibile performante ma anche leggibile e comprensibile. Per questo motivo l'ottimizzazione può non essere la migliore tra tutte, ma si e' cercato un compromesso tra ottimizzazione del codice e leggibilità pensata anche a una futura rimodifica e/o manutenzione del codice.

Extern Modules o *ASM* Inline?

E' stato scelto di utilizzare una funzione *ASM* esterna al codice *C* per poter lavorare con file scritti in solo assembly. La scelta progettuale e' stata fatta per rendere inoltre una maggiore utilita' all'utilizzo di un *makefile* all'interno del progetto.

STRUTTURA GENERALE

La scelta di utilizzare un *Extern Module* per la realizzazione del progetto e' stata per differenziare un file *C* da un file *ASM*, e per avere una maggiore dimestichezza nell'uso del linguaggio assembly. Per collegarli e' stato creato un unico file *ASM* chiamato '*controller_ASM.s*' contenente una funzione assembly in seguito richiamata dal file *C.*

La funzione *ASM* e' cosi' dichiarata e collegata al *controller.c* :

[controller_ASM.s]

```
.type controllerASM, @function  #Func Name
```

[controller.c]

```
// Dichiarazione
extern void controllerASM(char * bufferin, char * bufferout);

// Chiamata Funzione
controllerASM(bufferin, bufferout_asm);
```

Per poter utilizzare nel file ASM i due parametri passati alla funzione e' necessario recuperarli dallo stack pointer, ovvero `ESP`.

Il parametro '*bufferin*' contiene il puntatore all'array di char dei dati di ingresso, mentre il parametro '*bufferout_ASM*' contiene il puntatore all'array di char dei dati di uscita. Avendo quindi a disposizione il puntatore e' possibile lavorare in modo diretto con quello senza appoggiarsi ulteriormente allo stack.

Come si puo' vedere dalla *FIGURA 1* e' possibile recuperare dal file ASM i due parametri dallo stack tramite l'istruzione `movl`.

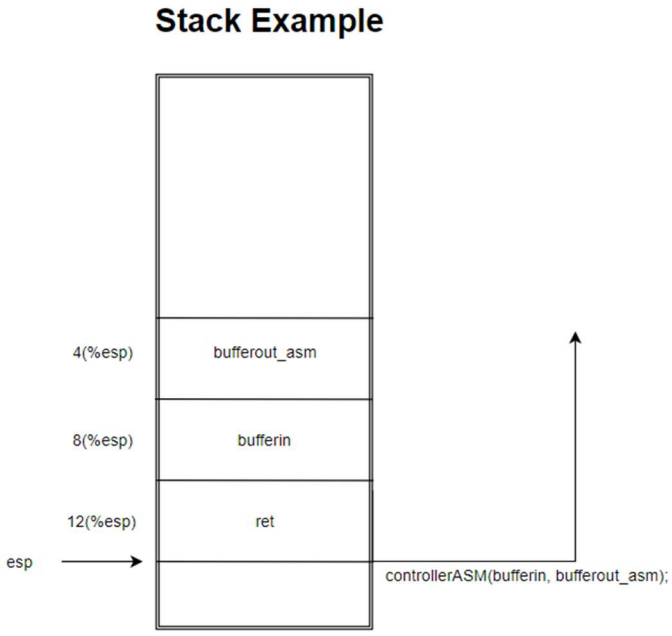


FIGURA 1

VARIABILI & REGISTRI

Per lo sviluppo del progetto non c'e stata la necessita di usare variabili. Invece, sono stati utilizzati tutti e 4 i registri generici `EAX`, `EBX`, `ECX`, `EDX`:

- `EAX` : Il registro `ax` e' stato utilizzato per poter rappresentare il numero di cicli di clock (*NCK*) su due char. Con le divisioni a byte `ax` (da 16 bit) permette di utilizzare i registri `al` e `ah` (da 8 bit) al meglio. Con questo registro e' stato possibile rappresentare *NCK* sui valori di *bufferout_asm* `st[1]` e `st[2]`.

- `EBX` : Il registro `bl` e' stato utilizzato come dividendo per le operazioni di divisione del *NCK*

- `ECX` : Il registro `cl` e' stato utilizzato come contatore per il numero di cicli di clock (*NCK*). Il registro viene azzerato ogni volta che viene cambiato stato, e incrementato ogni volta che, al passo successivo, lo stato rimane invariato. Il contatore non ha limiti di grandezza ma l'output di *NCK* e' rappresentabile solo su due caratteri, quindi, una volta raggiunto il 99 il registro proseguira' a incrementarsi di uno, ma l'output andra' in overflow.

- `EDX` : Il registro `dl` e' stato utilizzato come salvataggio dello stato. Poiche' bisogna verificare se lo stato e' cambiato o meno, il registro dl viene aggiornato ogni volta che lo stato cambia, in modo da aver salvato lo stato del passo precedente per confrontarlo con quello attuale.

- `ESI`: Il registro `esi` e' stato utilizzato come salvataggio e utilizzo del puntatore all'array *bufferin*.

- `EDI`: Il registro `edi` e' stato utilizzato come salvataggio e utilizzo del puntatore all'array *bufferout_asm*.

L'utilizzo dei registri generici senza il bisogno dell'allocazione di variabili rende piu' performante il codice assembly del progetto.

STRUTTURA DI BUFFERIN E BUFFEROUT

Il controllore ASM riceve come input un array di caratteri chiamato *bufferin*, suddiviso in righe, le quali devono essere analizzate, e tramite dei controlli devono produrre un secondo array di caratteri chiamato *bufferout_asm*.

Prima di vedere come vengono analizzate le righe del *bufferin* e' meglio comprendere come sono strutturate nello stack:

Con l'istruzione `movl 8(%esp), %esi` si sposta il puntatore all'array *bufferin* nel registro ESI. In questo modo tutte le letture, scritture, e controlli faranno riferimento al registro ESI. Ogni riga di input e' composta da `INIT,RESET,PHPHPH` quindi se prendiamo in considerazione una singola linea del input, nello stack avremmo:

ESI Stack Example

7(%esi) /n	6(%esi) **PH**	5(%esi) **PH**	4(%esi) **PH**
3(%esi) ,	2(%esi) **RESET**	1(%esi) ,	(%esi) **INIT**

E' importante tenere conto che se si ha un valore del pH come 123, nello stack sara' inserito specchiato, ossia in `4%(esi)` ci sara' il valore 1, in `5%(esi)` ci sara' il valore 2 e in `6%(esi)` ci sara' il valore 3. Se prendiamo come esempio una linea di input come '`0,0,072`' lo schema sara':

7(%esi) /n	6(%esi) **2**	5(%esi) **7**	4(%esi) **0**
3(%esi) ,	2(%esi) **0**	1(%esi) ,	(%esi) **0**

Incrementando il valore di esi, quindi il suo puntatore, grazie all'algebra dei puntatori, e' possibile scorrere tutto il *bufferin*, e quindi scorrere tutte le righe che ha come input il controller ASM. L'incremento avviene grazie all'istruzione `addl`.

Il controllore non sfora la grandezza del *bufferin* sullo lo stack leggendo altri dati perche' prima di eseguire l'incremento del puntatore per passare alla riga successiva controlla l'end of file (*oef*) e in quel caso termina.

La gestione e struttura del *bufferout* e' la medesima a *bufferin* a differenza che il puntatore si ottiene con l'istruzione `movl 12(%esp), %edi` e i dati vengono gestiti nello stack secondo il seguente schema: ogni linea di output e' cosi costruita `ST,NCKNCK,VLVVLV`

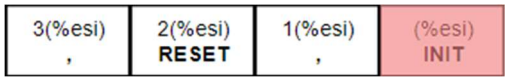
EDI Stack Example

7(%edi) /n	6(%edi) **VLV**	5(%edi) **VLV**	4(%edi) ,
3(%edi) **NCK**	2(%edi) **NCK**	1(%edi) ,	(%edi) **ST**

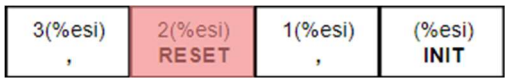
I registri `ESI` e `EDI`, rispettivamente *SourceRegister* e *DestinationRegister*, sono utilizzati per gestire l'input e output del controller. Una scelta poteva essere quella di sovrascrivere il *bufferin* stesso mentre lo si modifica, e in seguito scambiare i puntatori ma e' stato scelto di tenerli suddivisi per tenere un codice piu' pulito e leggibile.

FLUSSO DEL PROGRAMMA

Il controller *ASM* si basa su una serie di controlli al *bufferin* e *jump* ad etichette: parte dalla prima linea di input del *bufferin* e analizza riga per riga i dati producendo una riga di out, sul *bufferout_asm*, per ogni riga di input. Il controller analizza in modo sequenziale i bit della riga ed effettuando dei controlli salta a diverse etichette:

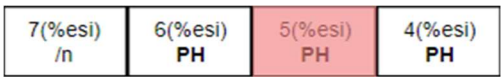


Il primo valore della riga che viene controllato e' il bit di `INIT`, che se posto a 0 il flusso del programma salta all'etichetta `reset.`

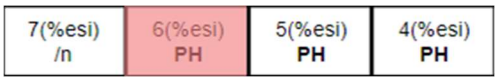


l il secondo valore della riga che viene controllato e' il bit di `RESET`, che se posto a 1 il flusso del programma salta all'etichetta `reset.`

Si controlla in seguito il primo valore del `PH`, che se ha un valore >=1, quindipH >100, vuol dire che non serve controllare i restanti valori del PH perche' e' basico.



Se non era >=1 si controlla il secondo valore. Se e' <6 e' acido, se e' >6 viene effettuato un controllo sul terzo valore, perche' e' neutro solo se e' anche < di 8. Se e' >= 6 e >8 allora e' basico, se e' =8 si effettua un controllo sul terzo valore del pH per confermare se e' basico, o neutro.

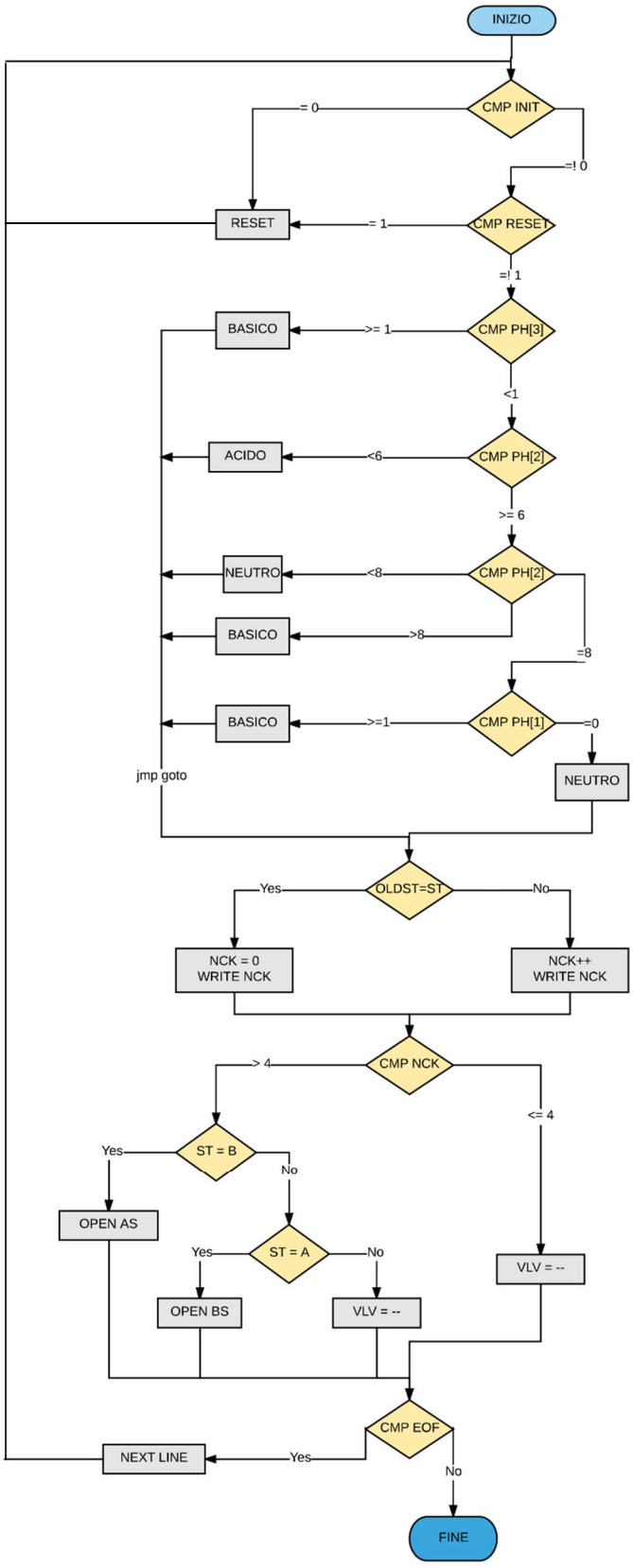


Se l'ultimo valore del pH,(i.e. 201, stiamo analizzando il bit alla pos. 1) e' >=1, vuol dire che il ph ha un valore >= 81, quindi e' basico, se e' =0 allora e' neutro.

Dopo i controlli sul pH il controller procede a effettuare dei controlli sullo stato corrente, che se e' uguale al vecchio stato, viene incrementato il numero di cicli di clock, se invece lo stato cambia il numero di cicli di clock si azzera. Un ultimo controllo viene effettuato sul numero di cicli di clock, che se e' maggiore 5 si apre una determinata valvola.

E' possibile vedere uno schema completo (flow chart) del flusso del controller alla pagina successiva (pagina 11).

FLOW CHART


INIZIO
CMP INIT
= 0
=! 0
RESET
= 1
CMP RESET
=! 1
BASICO
>= 1
CMP PH[3]
<1
ACIDO
<6
CMP PH[2]
>= 6
NEUTRO
<8
CMP PH[2]
BASICO
>8
=8
BASICO
>=1
CMP PH[1]
=0
jmp goto
NEUTRO
OLDST=ST
Yes
No
NCK = 0
WRITE NCK
NCK++
WRITE NCK
CMP NCK
> 4
<= 4
ST = B
Yes
No
OPEN AS
ST = A
Yes
No
OPEN BS
VLV = --
VLV = --
CMP EOF
NEXT LINE
Yes
No
FINE

DESCRIZIONE ETICHETTE

All'interno del controller ASM e' stato fatto ricorso all'uso di diverse etichette per permettere al programma, in determinate condizioni, di eseguire dei salti condizionati e non :

- `controllerASM`: Inizio principale del controller. Vengono preparati tutti i registri che vengono usati successivamente.
- `init`: Etichetta principale di inizio, indica il punto di 'INIZIO' nel flowchart. Vengono effettuati i controlli sui valori di `init`, `reset` e `pH`.
- `goto`: Etichetta necessaria per il ritorno del flusso del programma a quel punto (Dall'etichetta init e' possibile che il flusso del programma sia cambiato, e quindi debba essere ripristinato a questo punto). Viene controllato lo stato attuale con quello precedente per determinare se incrementare o no *NCK*.
- `goto1`: Etichetta simile a goto. Controlla il numero di clicli di clock (*NCK*) attuale. Se e' >4 effettua dei controlli per aprire le valvole.
- `goto2`: Etichetta simile a goto. Riempie il bufferout_asm con i valori fissi (i.e. la virgola ',' e a capo '/n').
- `reset`: Etichetta per resettare il bufferout_asm. Si verifica quando `init` = 0 o `reset` = 1.
- `acido/basico/neutro`: Etichette per settare lo stato di bufferout_asm ad acido (A), basico (*B*) o neutro (*N*).
- `checkvalvole`: Etichetta che determina quale valvola deve essere aperta.
- `openBS/openAS`: Etichette per aprire le valvole *AS*, o *BS* nel caso in cui il *NCK* sia maggiore di 4.
- `end`: ripristina lo stato dei registri sullo stack e ritorna la funzione *ASM*.