UNIVERSITÀ DEGLI STUDI DI VERONA

Elaborato ASM – Laboratorio di Architettura degli Elaboratori

Indice:

Specifiche

Il codice Assembly presentato si occupa di monitorare un impianto chimico industriale. Riceve in input il pH di una sostanza contenuta in un serbatoio e si occupa di definire se tale sostanza sia acida (pH < 6), basica (pH > 8) o neutra (6 ≤ pH ≤ 8). Inoltre il dispositivo si deve occupare di riportare la sostanza allo stato neutro, nell'eventualità in cui si trovi da più di 5 cicli di clock in uno stato diverso, attraverso l'apertura di una valvola che può essere acida (AS), nel caso in cui la sostanza sia nello stato basico, o basica (BS), nel caso contrario. Il programma fornisce in uscita anche il numero di cicli di clock da cui si trova nello stato attuale.

Il programma riceve in **input**:

- INIT [1] : segnala se la macchina è accesa o spenta, 1 equivale a macchina accesa, 0 a macchina spenta;
- RESET [1] : quando viene posto a 1 la macchina deve essere resettata;
- PH [3] : lo stato della sostanza che deve essere controllato. Il range è compreso tra 0 e 14, la precisione richiesta è di 0,1. I valori in ingresso risultano tutti moltiplicati per 10, per cui il range effettivo va da 0 a 140.

E restituisce questi **output**:

- ST [1] : indica lo stato in cui si trova la sostanza nel momento corrente. Acida A, Basica B, Neutra N. Perché la sostanza sia considerata acida il PH deve essere inferiore a 6, perché sia considerata basica deve essere superiore a 8, una sostanza compresa fra questi due valori viene considerata neutra;
- NCK [2] : indica il numero di cicli di clock trascorsi da quando la macchina si trova nello stato corrente;
- VLV [2] : indica le due valvole che devono essere aperte per rendere la sostanza nuovamente neutra, nel caso in cui la soluzione si trovi da più di 5 cicli di clock nello stesso stato.

Variabili utilizzate e loro scopo

Nello sviluppo del nostro progetto abbiamo deciso di non utilizzare né variabili né costanti, preferendo usufruire unicamente dei registri:

il registro a 8 bit **bl** si occupa di memorizzare lo stato attuale (ST)

il registro a 8 bit **cl** si occupa di memorizzare lo stato precedente (OLDST)

il registro a 16 bit **dx** si occupa di memorizzare il numero di cicli di clock (NCK)

in **dl** abbiamo memorizzate le unità (NCK[1])

in **dh** abbiamo memorizzate le decine (NCK[0])

Sono stati usati altri due registri **edi** e **esi** per lo scorrimento delle stringhe bufferin e bufferout_asm.

Modalità di passaggio e restituzione dei valori

Il codice Assembly lavora su due stringhe di 3201 valori massimo ciascuna. Queste due stringhe vengono inizializzate nel codice C sorgente per cui nel momento del passaggio ad ASM gli indirizzi di queste si trovano nello stack.

Le prime istruzioni del codice si occupano di questo:

pushl %ebp : copia il contenuto di **ebp** nello stack, nell'eventualità che a questo siano poste delle modifiche ed evitare che vadano in contrasto con il codice chiamante;

movl %esp, %ebp : assegna al registro **ebp** il contenuto di **esp**;

movl 12(%ebp), %edi : incrementando **ebp** di 12 puntiamo alla cella contenente l'indirizzo di bufferout_asm che salviamo in **edi**;

movl 8(%ebp), %esi : incrementando **ebp** di 8 puntiamo alla cella contenente l'indirizzo di bufferin che salviamo in **esi**.

La nostra funzione Assemly viene richiamata dal C come 'void' infatti lavorando sui puntatori delle due stringhe può modificarle senza dover ritornare valori.

Pseudo – codice ad alto livello

Funzione controller_asm:

NCK[0] = 0
NCK[1] = 0

Etichetta **while**:

Si occupa di scorrere ogni riga di bufferin e creare ogni riga di bufferout_asm.

```
se bufferin [i] == '\0'
        salta all'etichetta fine_ciclo
altrimenti
        se bufferin [i] == '0'
                salta all'etichetta nulla
        altrimenti
                se bufferin [i + 2] == '1'
                        salta all'etichetta nulla
                altrimenti
                        se bufferin [i + 4] != '0'
                                salta all'etichetta basico
                        altrimenti
                                se bufferin [i + 5] < '6'
                                        salta all'etichetta acido
                                altrimenti
                                        se bufferin [i + 5] < '8'
                                                salta all'etichetta neutro
                                        altrimenti se bufferin [i + 5] > '8'
                                                salta all'etichetta basico
                                        altrimenti
                                                se bufferin [i + 6] > '0'
                                                        salta all'etichetta basico
                                                altrimenti
                                                        'N' memorizzato in ST
                                                        salta all'etichetta fine_stato
```

Etichetta **basico**:

Si occupa di assegnare a **ST** il carattere **B**, per basico.

```
'B' memorizzato in ST
salta all'etichetta fine_stato
```

Etichetta **acido**:

Si occupa di assegnare a **ST** il carattere **A**, per basico.

```
'A' memorizzato in ST
salta all'etichetta fine_stato
```

Etichetta **neutro**:

Si occupa di assegnare a **ST** il carattere **N**, per basico.

‘**N**’ memorizzato in **ST**
salta all’etichetta ***fine_stato***

Etichetta **fine_stato**:

Si occupa di salvare il valore di **ST** e la prima virgola in bufferout_asm, di puntare alla riga successiva di bufferin per il ciclo successivo. Inoltre effettua il confronto tra lo stato precedente e quello presente per incrementare o azzerare **NCK**.

i = i + 8
ST memorizzato in bufferout_asm [j]
‘,’ memorizzato in bufferout_asm [j + 1]
se **OLDST** == **ST**
    salta all’etichetta ***incrementa***
altrimenti
    OLDST = **ST**
    NCK[0] = 0
    NCK[1] = 0
    salta all’etichetta ***salvataggio_NCK***

Etichetta **salvataggio_NCK**:

Si occupa di salvare **NCK** in bufferout_asm.

NCK [0] memorizzato in bufferout_asm [j + 2]
NCK [1] memorizzato in bufferout_asm [j + 3]
salta all’etichetta ***no_vlv***

Etichetta **no_vlv**:

Si occupa di salvare in bufferot_asm nessun valore nel caso in cui non si debba aprire alcuna valvola.

‘,’ memorizzato in bufferout_asm [j + 4]
‘-’ memorizzato in bufferout_asm [j + 5]
‘-’ memorizzato in bufferout_asm [j + 6]
salta all’etichetta ***line_feed***

Etichetta **line_feed**:

Si occupa di salvare in bufferout_asm il simbolo per iniziare una nuova riga.

‘\n’ memorizzato in bufferout_asm [j + 7]
j = j + 8
salta all’etichetta ***while***

Etichetta **incrementa**:

Si occupa di incrementare **NCK**. Nel nostro progetto le decine di **NCK** sono salvate nel registro a 8 bit **dh (NCK[0])**, mentre le unità si trovano in **dl (NCK[1])**. Per cui per incrementare **NCK** prima effettuiamo un confronto tra **dl (NCK[1])** e 9, infatti nel caso in cui questo sia uguale a 9 è necessario azzerarlo ed incrementare **dh (NCK[0])** di uno. In più questa etichetta controlla che lo stato non sia uguale ad **N**, in tal caso viene salvato **NCK** e nessuna valvola, altrimenti si passa al confronto di **NCK** per l'apertura della valvola.

```
OLDST = ST
se NCK [1] == 9
        salta all'etichetta incrementa_decine
altrimenti
        NCK[1] = NCK[1] + 1
        se NCK [0] == 0
                salta all'etichetta no_decine
        altrimenti
                se ST == 'N'
                        salta all'etichetta salvataggio_NCK
                altrimenti
                        salta all'etichetta confronto
```

Etichetta **incrementa_decine**:

Si occupa di incrementare **dh (NCK[0])**.

```
NCK [1] = 0
NCK [0] = NCK[0] + 1
salta all'etichetta confronto
```

Etichetta **no_decine**:

Si occupa di controllare, nel caso in cui le decine siano uguali a zero, se le unità sono maggiori di 4, in tal caso procede con l'apertura della valvola, altrimenti salva **NCK**.

```
se NCK [1] <= 4
        salta all'etichetta salvataggio_NCK
altrimenti
        se ST == 'N'
                salta all'etichetta salvataggio_NCK
        altrimenti
                                salta all'etichetta confronto
```

Etichetta **confronto**:

Si occupa di confrontare **ST** per determinare quale valvola aprire, oltre a memorizzare **NCK** e la valvola da aprire in bufferout_asm .

```
NCK [0] memorizzato in bufferout_asm [j + 2]
NCK [1] memorizzato in bufferout_asm [j + 3]
se OLDST != 'A'
```

salta all'etichetta ***vlv_acida***
altrimenti
‘,’ memorizzato in bufferout_asm [j + 4]
‘**B**’ memorizzato in bufferout_asm [j + 5]
salta all'etichetta ***lettera_s***

Etichetta **vlv_acida**:

Si occupa del salvataggio della valvola acida, o di nessuna valvola nel caso in cui lo stato sia **N**.

se **OLDST** != ‘**B**’
salta all'etichetta ***no_vlv***
altrimenti
‘,’ memorizzato in bufferout_asm [j + 4]
‘**A**’ memorizzato in bufferout_asm [j + 5]
salta all'etichetta ***lettera_s***

Etichetta **lettera_s**:

Si occupa del salvataggio della lettera **S** in buffeout_asm.

‘**S**’ memorizzato in bufferout_asm [j +6]
salta all'etichetta ***line_feed***

Etichetta **nulla**:

Si occupa del salvataggio della stringa nulla.

‘-’ memorizzato in **OLDST**
‘-‘ memorizzato in bufferout_asm [j]
‘,’ memorizzato in bufferout_asm [j + 1]
‘-‘ memorizzato in bufferout_asm [j + 2]
‘-‘ memorizzato in bufferout_asm [j + 3]
‘,’ memorizzato in bufferout_asm [j + 4]
‘-’ memorizzato in bufferout_asm [j + 5]
‘-’ memorizzato in bufferout_asm [j + 6]
‘\n’ memorizzato in bufferout_asm [j + 7]
j = j + 8
i = i + 8
salta all'etichetta ***while***

Etichetta **fine_ciclo**:

Si occupa del salvataggio del carattere nullo alla fine della stringa buffeout_asm.

‘\0’ memorizzato in bufferout_asm [j + 7]

Ritorna (al linguaggio C)
* i e j sono delle variabili fittizie che non sono state usate nel codice originale, ma nello pseudo – codice per migliorare la spiegazione.

Scelte progettuali effettuate

Sviluppando il progetto abbiamo effettuato le seguenti decisioni:

- Non convertire il PH in intero, ma di considerare i caratteri passati da bufferin uno ad uno singolarmente;
- Effettuare il conteggio del numero di cicli di clock direttamente in carattere in modo da non dover effettuare la conversione successivamente;
- Incrementare i puntatori alle stringhe solo alla fine di ogni ciclo e non passo passo.