

# Relazione progetto MIPS

- Realizzato da Daniel Zanchi - [daniel.zanchi@stud.unifi.it](mailto:daniel.zanchi@stud.unifi.it)
- Data di consegna: 30 Giugno 2015

## - Esercizio 1 - Identificatore di sotto-sequenze

### Testo:

Utilizzando QtSpim, scrivere e provare un programma che realizzi un identificatore di sotto-sequenze di caratteri. Il programma legge in input da file una sequenza di  $N$  caratteri, con  $10 \leq N \leq 100$ , dove ogni carattere può essere o '0' o '1'. Il file di input deve chiamarsi "sequenza.txt" e deve risiedere nella stessa cartella in cui è presente l'eseguibile QtSpim. Il programma permette anche di inserire in input da tastiera un numero intero  $X$  compreso nell'intervallo  $-511 \leq X \leq 511$ .

Sia  $[C_1 \dots C_N]$  la sequenza di caratteri memorizzati su file, dove  $C_1$  è il primo carattere memorizzato nel file stesso, e sia  $Seq(k)$  una sua sotto-sequenza di 10 caratteri, ovvero  $Seq(k) = [C_k \dots C_{k+9}]$  per qualche  $1 \leq k \leq N-9$ .

Il programma:

- Visualizza su console l'insieme degli indici  $k$  tali che  $V([C_k \dots C_{k+9}]_{2, MS}) = X$ , ovvero l'insieme degli indici  $k$  tali che  $Seq(k)$  corrisponde alla codifica in **Modulo e Segno (MS)** su 10 bit del numero intero  $X$  inserito in input da tastiera.
- Visualizza su console l'insieme degli indici  $k$  tali che  $V([C_k \dots C_{k+9}]_{2, C2}) = X$ , ovvero l'insieme degli indici  $k$  tali che  $Seq(k)$  corrisponde alla codifica in **Complemento a Due (C2)** su 10 bit del numero intero  $X$  inserito in input da tastiera.
- Visualizza su console l'insieme degli indici  $k$  tali che  $V([C_k \dots C_{k+9}]_{2, C1}) = X$ , ovvero l'insieme degli indici  $k$  tali che  $Seq(k)$  corrisponde alla codifica in **Complemento a Uno (C1)** su 10 bit del numero intero  $X$  inserito in input da tastiera

## - Soluzione adottata:

L'idea che ho utilizzato per trovare le sotto-sequenze è stata di convertire il mio input  $X$  nelle tre codifiche, salvandole in tre vettori diversi:

MS\_array

C2\_array

C1\_array

Poi confrontare questi tre vettori uno ad uno con la sequenza letta dal file.

Il codice è stato strutturato con le funzioni più ripetute situate prima del main, dove vengono richiamate mediante l'istruzione Jal. Sono state create funzioni come *stampa\_stringa*, *stampa\_intero* perché, anche se funzioni molto corte, vengono richiamate svariate volte all'interno del programma. Anche la funzione *converti\_binario* viene richiamata più volte proprio perché tutte e tre le codifiche utilizzano la conversione in binario puro in molti casi.

main:

```
Apri file
Leggi file -> buffer[ ]
Conta numero caratteri della sequenza -> n
Stampa numero caratteri della sequenza
Stampa sequenza
Chiedi input:
    x := input
    if (-511 > x > 511)
        jump to -> chiedi input
Stampa x
Converto MS:
    if ( x < 0 )
        MS_array [0] = 1
        x = -x
    Converti binario (MS_array)
Converto C2:
    if ( x < 0 )
        x = x + 1024
    Converti binario (C2_array)
Converto C1:
    if ( x < 0 )
        x = x + 1023
    Converti binario (C1_array)
Stampa MS_array
if ( x == 0 )
    Stampa MS_array_alt
Stampa C2_array
Stampa C1_array
Confronta MS_array:
    for1 [ i = 1...n-10 ]
        indice_occ ++
        cont = 0
        for2 [ i = 1...n-10]
            if ( cont == 10 )
                cont = 0
                ii = 0
                Stampa (indice_occ)
                jump to for1
            A =: buffer[i]
            converti_char_int (A)
            B =: MS_array[ii]
            if ( A == B )
                ii ++
                cont ++
                jump to for2
        jump to for1
if ( x == 0 )
    Confronta MS_Array_alt
Confronta C2_array
Confronta C1_array
End
```

Converti binario:

```
for [ i = 10 ... 1 ]
```

```

    array [ i ] = mod x;
    x = x / 2
    i - -
    if ( x == 0 )
        end

```

end

Converti\_char\_int:

```

    if (A == 49 )
        A = 1
    else
        A = 0

```

end

- Approfondimenti sul codice:

Il buffer sul quale verrà messa la sequenza è stata creata di dimensione 101, può contenere 100 caratteri, un carattere in più per indicare la fine del file.

L'input viene controllato che sia compreso tra -511 e 511, se viene inserito un carattere QtSpim prende come input 0.

Dobbiamo considerare le due codifiche che ha il modulo e segno con lo 0. Quindi se X è uguale a 0 viene preso in considerazione anche l'array (precaricato con 1000000000) chiamato MS\_array\_alt.

Altre funzioni come Stampa\_stringa, Stampa\_intero, Stampa\_array non vengono riportate perché sono state viste nello specifico a lezione.

La funzione Converti\_char\_int serve per convertire un carattere in codice ASCII in intero.

Infatti i carattere vengono letti dal file sequenza.txt e messi in un buffer non come interi ma come caratteri. quindi lo 0 corrisponderà al 48, mentre l'1 al 49. Questa funzione converte da char a intero.

Per il complemento a due, nel caso venga inserito un numero negativo si somma al nostro numero 1024 ( $2^{10}$ ) e poi lo convertiamo in binario puro. Se ad esempio inseriamo in input "-5" questo viene convertito come 1019 ( $-5 + 1025$ ), la sua codifica in binario puro equivale alla codifica in complemento a due di "-5".

Per il complemento a uno, nel caso vengo inserito un numero negativo si somma al nostro numero 1023 ( $2^{10}-1$ ) e poi lo convertiamo in binario puro.

Nel programma sono stati usati:

- 4 vettori di tipo .word (per contenere interi) utilizzati per le varie codifiche già scritte sopra.
- 1 vettore di tipo .space, buffer per contenere la sequenza del file
- Varie stringhe di tipo .ascii in modo da stampare messaggi e rendere i codici su console più semplici e leggibili.
- Quasi tutti i registri erano temporanei e i loro valori venivano modificati durante l'esecuzione del codice. \$s0 è stato usato per appoggiare il valore X. \$s7 salva il numero di caratteri presenti sul file.

Le chiamate a procedura per le varie funzioni ripetute non avevano bisogno di salvare il registro \$ra in quanto queste procedure non effettuavano una jal al loro interno. è quindi stato sufficiente effettuare una jr (jump register) direttamente sul registro \$ra. saltando al PC del chiamante + 4.

```

#FUNZIONE PER STAMPARE ARRAY CON INDIRIZZO
stampa_array:
    move $t0, $a0
    li $t5, 40
    li $t1, 0          #la grandezza del vettore
    stampa_int_array:  #ciclo dopo aver stampato l'intero
        lw $a0, 0($t0)  #salvo in $a0
        li $v0, 1       #stampo l'intero contenuto
        syscall
        li $v0, 4
        la $a0, str_spazio #stampo uno spazio
        syscall
        addi $t1, $t1, 4
        addi $t0, $t0, 4  #incremento il puntatore
        blt $t1, $t5, stampa_int_array #finisce quando $t1 > $t5
    jr $ra #se ha finito di leggere tutto

```

1. Quella che ho adottato io converte X nelle tre codifiche e le appoggia in tre vettori, poi confronta una codifica alla volta con la sequenza. (era possibile migliorare questa implementazione confrontando le tre codifiche nello stesso ciclo, invece di fare tre cicli diversi, ma in MIPS era piuttosto complicata la gestione dei registri)
2. In alternativa si poteva leggere i primi 10 caratteri della sequenza, codificarla e confrontarla con X, poi scorrere la sequenza di uno e leggere altri 10 caratteri e così via. Questa implementazione sembrerebbe logicamente più veloce, ma richiedeva molti più calcoli aritmetici, appesantendo il lavoro della CPU, ma risparmiando lo spazio usato dai vettori nel primo caso.

Nel caso di input uguale a 0, vediamo come partecipa al programma anche l'array con la codifica del modulo e segno alternativa. E che è presente l'indice 21 nelle occorrenze trovate:

```
Daniel Zanchi, Esercizio n.1 del progetto

Numero caratteri sul file: 35
Sequenza sul file: 0000000000000000000100000000000000

Inserisci il valore di x da cercare: 0

Valore inserito: 0
Conversione modulo e segno: 0 0 0 0 0 0 0 0 0 0 0
Conversione modulo e segno alt: 1 0 0 0 0 0 0 0 0 0 0
Conversione complemento a due: 0 0 0 0 0 0 0 0 0 0 0
Conversione complemento a uno: 0 0 0 0 0 0 0 0 0 0 0

Caso MS - Elenco indici di inizio occorrenza: 21 1 2 3 4 5 6 7 8 9 10 11 22 23 24 25 26
Caso C2 - Elenco indici di inizio occorrenza: 1 2 3 4 5 6 7 8 9 10 11 22 23 24 25 26
Caso C1 - Elenco indici di inizio occorrenza: 1 2 3 4 5 6 7 8 9 10 11 22 23 24 25 26
-- program is finished running --
```

Se viene inserito un valore minore di -511 vediamo come viene stampato il segnale di errore e viene richiesto di reinserire il valore X:

Daniel Zanchi, Esercizio n.1 del progetto

Numero caratteri sul file: 35  
Sequenza sul file: 0000000000000000000000000100000000000000

Inserisci il valore di x da cercare: -700

E' stato inserito un valore minore di -511

Inserisci il valore di x da cercare:

Qui possiamo vedere il *data segment* che contiene le codifiche, quella non evidenziata è la codifica alternativa dello zero in MS:

[illegible]

## - **Esercizio 2 - Procedure annidate e ricorsive**

### - **Testo:**

Siano  $G$  e  $F$  due procedure definite come segue (in pseudo-linguaggio di alto livello):

Procedure  $G(n)$

```
begin
     $b := 0$ 
    for  $k := 0, 1, 2, \dots, n$  do
        begin
             $u := F(k)$ 
             $b := b^2 + u$ 
        end
    end
    return  $b$ 
end
```

Procedure  $F(n)$

```
begin
    if  $n = 0$ 
        then return 1
        else return  $2 * F(n - 1) + n$ 
    end
end
```

Utilizzando QtSpim, scrivere e provare un programma che legga inizialmente un numero naturale  $n$ , e che visualizzi su console:

- il valore restituito dalla procedura  $G(n)$ , implementando  $G$  e  $F$  come descritto precedentemente. Le chiamate alle due procedure  $G$  ed  $F$  devono essere realizzate utilizzando l'istruzione jal (jump and link).

- la traccia con la sequenza delle chiamate annidate (con argomento fra parentesi) ed i valori restituiti dalle varie chiamate annidate (valore restituito fra parentesi), sia per  $G$  che per  $F$ .

Mostrare e discutere nella relazione l'evoluzione dello stack nel caso specifico in cui  $n=2$ .

### - **Soluzione adottata**

Nelle due procedure erano presenti un ciclo iterativo (for) dove all'interno veniva invocata la procedura  $F(k)$ . Nella funzione  $F(n)$  c'era una chiamata ricorsiva che richiamava se stessa per  $n-1$  volte.

Per realizzare una chiamata ricorsiva è stato necessario utilizzare la pila mediante lo stack frame presente nel MIPS, incrementando lo stack pointer per salvare registri e valori di ritorno e una volta utilizzati questi valori decrementandolo per andare ai precedenti.

Lo stack frame in questo programma svolge un lavoro molto importante, perché salva i valori di ritorno ottenuti dalla funzione  $2 * F(n - 1) + n$  e salva anche ogni volta il valore del registro  $\$ra$ , in modo da eseguire le istruzioni salvando i valori della funzione e poi eseguire tutto a ritroso, avendo salvato i valori che rende la funzione e avendo il valore di dove deve saltare.

Il codice è stato scritto prendendo spunto dall'esercizio visto a lezione "fattoriale" e seguendo lo pseudocodice dato dal testo dell'esercizio, sono state aggiunte le stampe e la gestione dello stack frame che non è visibile in un linguaggio ad alto livello.

La traccia risultante con n=2:

Traccia:

G(2) --> F(0) --> F-return(1) --> F(1) --> F(0) --> F-return(1) --> F-return(3) --> F(2) --> F(1) --> F(0)  
--> F-return(1) --> F-return(3) --> F-return(8) --> G-return(24)

Risultato: G(2)=24

Infatti:

G(2) richiama F(0), che ritorna il valore 1; quindi viene richiamata F(1), che a sua volta richiama F(0); F(0) ritorna il valore 1, quindi F(1) ritorna il valore 3, viene chiamata F(2), che a sua volta richiama F(1) che a sua volta richiama F(0), la quale ritorna 1, successivamente F(1) ritorna 3 e infine F(2) ritorna 8. Giunti a fine del ciclo for la funzione G ritorna 24.

Vediamo la situazione dello stack frame quando viene eseguito il 3° ciclo del for, quindi con k=2:

The screenshot shows the Mars Data Segment window. The table below represents the data shown in the window:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)
2147479520	0	0	4194476	1	4194476	2	4194664
2147479552	0	0	0	0	0	0	0
2147479584	0	0	0	0	0	0	0
2147479616	0	0	0	0	0	0	0
2147479648	0	0	0	0	0	0	0

Below the table, the 'current \$sp' is set to 2147479648. The 'Mars Messages' window shows the following text:

Daniel Zanchi, Esercizio n.2 del progetto

Inserire il valore n: 2

Traccia:

G(2) --> F(0) --> F-return(1) --> F(1) --> F(0) --> F-return(1) --> F-return(3) --> F(2) --> F(1) --> F(0) -->

In verde vediamo come i valori di chiamata vengono salvati nello stack frame relativi a F(2), F(1) e F(0) eseguiti poi a ritroso, in rosso vediamo gli indirizzi di ritorno.

- Osservazioni:

Nel testo richiede che il numero n inserito in input sia naturale quindi viene eseguito un controllo che sia maggiore o uguale a 0. Anche in questo caso se viene inserito un carattere invece che un numero, QtSpim lo interpreta come uno 0.

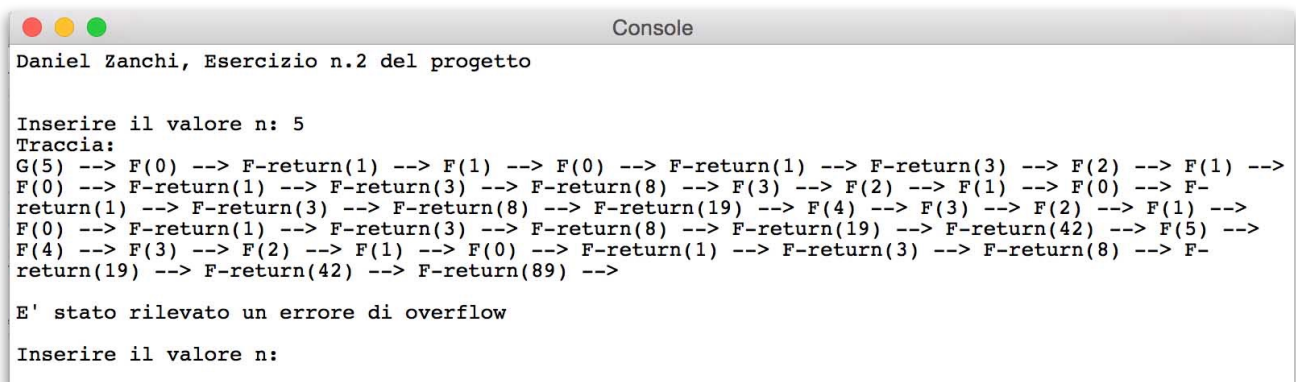
L'esercizio chiedeva anche di stampare una traccia dei valori di ritorno, questo è stato fatto all'interno della ricorsione. E' stato differenziato il caso base e il caso non base:

Il caso base stampava F-return(n) con n <= 1, mentre il caso non base stampava F-return(n) con n > 1.

I registri utilizzati nel MIPS sono di 32 bits, e avendo nella funzione F un'operazione con potenza è molto facile arrivare a un overflow. Questo però poteva succedere senza che l'utente si accorgesse di niente, perché l'operazione mul mette il risultato su due registri *\$hi* e *\$low*, che sono entrambi di 32 bits, quindi se il mio risultato andava sia sul registro *\$low* che sul registro *\$hi* avevo un risultato maggiore ai 32 bits che avrei poi spostato in un registro a 32 bits perdendo quindi parte significativa del mio numero. Per prevenire a questo problema è stato necessario dare un errore nel case che il registro *\$hi* venisse occupato in fase di moltiplicazioni. Infatti se inseriamo un input maggiore o uguale a 5 riceviamo un messaggio di errore di overflow e ci invita a reinserire il valore n.

```
mul $t0, $t0, $t0      #eseguo b = b * b
mfhi $t3               #prendo il secondo registro da 32 bit della moltiplicazione
beqz $t3, ok           # se registro hi diverso da zero c'è stato un overflow, altrimenti salto e vado avanti
la $a0, str_overflow   #stampo errore di overflow
jal stampa_stringa
j chiedo_n             #salto per reinserire n
ok:
```

Infatti vediamo il risultato:



```

Daniel Zanchi, Esercizio n.2 del progetto

Inserire il valore n: 5
Traccia:
G(5) --> F(0) --> F-return(1) --> F(1) --> F(0) --> F-return(1) --> F-return(3) --> F(2) --> F(1) -->
F(0) --> F-return(1) --> F-return(3) --> F-return(8) --> F(3) --> F(2) --> F(1) --> F(0) --> F-
return(1) --> F-return(3) --> F-return(8) --> F-return(19) --> F(4) --> F(3) --> F(2) --> F(1) -->
F(0) --> F-return(1) --> F-return(3) --> F-return(8) --> F-return(19) --> F-return(42) --> F(5) -->
F(4) --> F(3) --> F(2) --> F(1) --> F(0) --> F-return(1) --> F-return(3) --> F-return(8) --> F-
return(19) --> F-return(42) --> F-return(89) -->

E' stato rilevato un errore di overflow
Inserire il valore n:

```

Nel codice non sono stati usati vettori, ma sono state usate delle stringhe .ascii per stampare messaggi sulla console.

Visto che si trattava di due procedure, è stato necessario passare i vari argomenti tramite i registri *\$a0*, *\$a1* ... e i valori di ritorno venivano comunicati tramite *\$v0*

Ad esempio per passare il valore inserito in input alla procedura\_g prima di effettuare la *jal* viene passato il valore tramite *\$a1*.



### - **Esercizio 3 - Operazioni fra matrici**

#### - **Testo:**

Utilizzando QtSpim, scrivere e provare un programma che visualizzi all'utente un menù di scelta con le seguenti quattro opzioni:

a) **Inserimento di matrici.** Il programma richiede di inserire da tastiera un numero naturale  $n > 0$ , e richiede quindi l'inserimento di due matrici quadrate, chiamate A e B, di dimensione  $n \times n$ , contenenti numeri interi. Quindi si ritorna al menù di scelta.

Le matrici A e B dovranno essere allocate dinamicamente in memoria utilizzando la system call 'sbrk' del MIPS, e per loro memorizzazione si consiglia l'utilizzo di una struttura dati a lista.

Ogni volta che si seleziona l'opzione a) del menu, i nuovi valori inseriti di A e B dovranno essere salvati nelle stesse locazioni di memoria in cui erano stati salvati i vecchi valori (per limitare l'utilizzo della memoria), quindi i nuovi valori sovrascriveranno quelli vecchi. Si potrà allocare (con la 'sbrk') uno spazio aggiuntivo di memoria solo se le due nuove matrici dovessero richiedere più spazio di memoria rispetto a quello già allocato in precedenza.

Esempio di interfaccia per l'inserimento delle due matrici:

Dimensione matrici: 3x3

Matrice A:

Riga 1: -2 44 5

Riga 2: 1 1 1

Riga 3: 3 0 1

Matrice B:

Riga 1: 0 0 10

Riga 2: -1 1 -1

Riga 3: 1 0 0

b) **Somma di matrici.** Il programma effettua la somma fra le due matrici A e B, e visualizza su console il risultato  $A+B$ . Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

Risultato di  $A+B$ :

Riga 1: -2 44 15

Riga 2: 0 2 0

Riga 3: 4 0 1

c) **Sottrazione di matrici.** Il programma effettua la sottrazione fra le due matrici A e B, e visualizza su console il risultato  $A-B$ . Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

Risultato di  $A-B$ :

Riga 1: -2 44 -5

Riga 2: 2 0 2

Riga 3: 2 0 1

d) **Prodotto di matrici.** Il programma effettua il prodotto fra le due matrici A e B, e visualizza su console il risultato  $A*B$ . Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

Risultato di  $A*B$ :

Riga 1: -39 44 -64

Riga 2: 0 1 9

Riga 3: 1 0 30

e) Stampa un messaggio di uscita e esce dal programma.

### - Soluzione adottata:

Il testo chiedeva di realizzare le matrici mediante l'uso della chiamata *sbrk*. Per ogni elemento della matrice viene creata una cella di memoria contenente due blocchi, ciascuno di 4 byte, nel primo blocco viene inserito il valore dell'elemento, nel secondo blocco viene inserito l'indirizzo alla cella successiva (0 in caso di ultimo elemento).

Il codice crea le matrici facendo una lista collegata dove ogni n (grandezza della matrice inserita in input) elementi della lista, siamo alla riga successiva.

Quindi se ad esempio inseriamo 3 come grandezza della matrice sono necessarie 9 celle di memoria per ogni matrice, dove: dalla cella 1 alla 3 siamo alla prima riga, dalla cella 4 alla 6 siamo alla seconda riga e infine dalla cella 7 alla 9 siamo alla terza riga.

Per la funzione di inserimento se già avevo allocato memoria in precedenza dovevo riutilizzare questa memoria per l'inserimento di una nuova matrice.

Quindi se ad esempio appena aperto il programma creavo una matrice 2x2 avevo già create mediante la funzione *sbrk* 4 celle di memoria. Nel caso che successivamente avrei voluto creare una matrice 3x3 avevo bisogno di 9 celle di memoria, ma visto che 4 erano già state create in precedenza non dovevo richiamare la syscall *sbrk* 9 volte, ma solamente 5. Quindi avrei sfruttato le 4 celle già create con la prima matrice.

Per fare questo, ogni volta che creavo una matrice andavo prima a controllare se alla testa della mia matrice il blocco che doveva contenere l'indirizzo della cella successiva era uguale a 0. Nel caso ci fosse stato un indirizzo allora avevo già allocato la memoria, quindi potevo, tramite una *sw* salvare il mio numero nel primo blocco e saltare alla cella successiva, senza richiamare la *sbrk*, poi appena trovato nel blocco *elemento successivo* uno 0, allora alla cella successiva avrei dovuto chiamare la funzione *sbrk* per allocare memoria.

Per la stampa è stato molto semplice, in quanto bastava utilizzare un contatore che ogni volte che diventava uguale a n (grandezza matrice) sapevo che dovevo stampare un "\n" per andare a capo. Quindi leggere ogni elemento dalla memoria tramite una *lw* e lo stesso per aggiornare il mio puntatore, leggendo il campo *elemento successivo* sempre tramite una *lw*.

```
stampa_matrice: # a0 = Testa. t0 verra' usato come puntatore per scorrere gli elementi della matrice, $a1 grandezza matrice
move $t0, $a0
nuova_riga:      #stampo un "\n" per andare a capo
la $a0, str_a_capo
li $v0, 4
syscall
move $t2, $zero #setto il contatore delle colonne a 0
loop_print:
beq $t0, $zero, matrice_stampata # se t0 == 0 si e' raggiunta la fine della matrice e si esce
beq $t2, $a1, nuova_riga         #se il contatore t2 == grandezza matrice (n colonne) allora salto a stampare nuova riga
addi $t2, $t2, 1                 #incremento contatore delle colonne stampate
li $v0, 1                        # 1 = syscall per stampare intero
lw $a0, 0($t0)                   # a0 = valore del campo intero dell'elemento corrente (puntato da t0)
syscall                          # stampa valore intero dell'elemento corrente
li $v0, 4                        # stampo uno spazio, per separare un elemento dall'altro
la $a0, str_spazio
syscall
lw $t0, 4($t0)                   # t0 = valore del campo elemento-successivo dell'elemento corrente (puntato da t0)
j loop_print                     #salto e continuo a stampare la matrice
matrice_stampata:
jr $ra                          #salto all'indirizzo contenuto nel registro ra
```

Per la somma e la sottrazione è molto simile alla stampa, infatti è bastato utilizzare due puntatori, (uno per ogni matrice) e caricare i valori degli elementi insieme, sommandoli (o sottraendoli) in un registro per poi stamparli. Questo è stato facile perché lavoriamo con matrici quadrate.

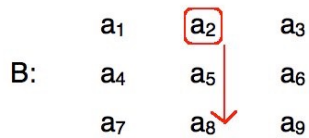
La moltiplicazione è stata un po' più complicata in quanto la moltiplicazione tra matrici non moltiplica solo l'elemento che si trova in posizione x della matrice A per l'elemento che si trova in posizione x della matrice B. Infatti moltiplica la riga in cui si trova l'elemento della matrice A per la colonna in cui si trova l'elemento della matrice B, sommando le varie moltiplicazioni.

Per fare questo nella mia lista collegata è piuttosto macchinoso come procedimento:

Se mi trovo alla posizione x, devo sapere in quale riga e in quale colonna mi trovo. Per sapere la colonna mi è bastato controllare il contatore che contava le colonne per sapere quando andare a capo (così come ho usato nella stampa, somma e sottrazione). Quindi bastava partire dalla testa

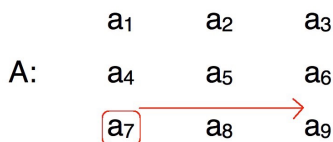
della matrice B e scorrere gli elementi della prima riga finché non arrivavo alla colonna giusta. Una volta arrivati alla colonna giusta sapevo che avrei dovuto moltiplicare da quell'elemento e tutti i sottostanti.

Nel caso che mi trovassi all'elemento  $a_8$ , il puntatore punterebbe a  $a_2$ :



Per trovare la riga giusta invece mi bastava dividere il numero della posizione in cui mi trovavo per la grandezza della matrice. Ad esempio, se ero all' $a_8$  posizione in una matrice  $3 \times 3$ , il contatore (partendo da 0) conta 7,  $7/3 = 2$ , quindi alla terza riga. Trovata la riga puntavo alla testa della matrice A e scorrevo tramite un ciclo doppio (per scorrere alla riga successiva e non al prossimo elemento) e mi fermavo all'inizio della riga in cui si trovava l'elemento.

Nel caso che mi trovassi all'elemento  $a_8$ , il puntatore punterebbe a  $a_7$ :



Una volta che punto agli elementi giusti, mi muovevo passo passo, moltiplicando  $A[i] \times B[i]$  e sommando per i precedenti. Scorrendo la matrice A per riga e la matrice B per colonna. Passando poi all'elemento successivo ripetendo questi passi.

main:

Stampa menu

Input scelta -> x

Scelta menu:

if ( x == 'a' )

    jump Inserisci matrici

if ( x == 'b' )

    jump Somma matrici

if ( x == 'c' )

    jump Sottrai matrici

if ( x == 'd' )

    jump Moltiplica matrici

if ( x == 'e' )

    jump Esci

else

    jump Stampa menu

Inserisci matrici:

Inserisci grandezza matrice -> n

Inserisci matrice A:

cont = 0

Riempi matrice:

if ( cont == n )

    jump Inserisci matrice B

if ( A[next] == 0 )

Crea cella:

if ( cont == n )

    jump Inserisci Matrice B

sbrk ( 8 byte, punt )

Chiedi elemento -> x

A[punt] = x

```

        A[punt-next] = 0
        if ( my-punt != 0 )
            Collega ultimo:
                A[coda] = punt
                coda = punt
        else
            my-punt = punt
            coda = punt
        cont ++
        jump Crea cella
    else
        Chiedi elemento -> x
        my-punt = A[next]
        A[my-punt] = x
        jump Riempi matrice
Inserisci matrice B
...
Stampa matrice A:
    punt = testa A
    Nuova riga:
        Stampa a capo
        cont_colonne = 0
        Prossimo elemento:
            if ( punt == 0 )
                jump Stampa matrice B
            if ( cont_colonne == n )
                jump Nuova riga
            cont_colonne ++
            Stampa ( A[punt] )
            punt = A[next]
            jump Prossimo elemento
Stampa matrice B
Matrice Stampata:
    jump Main
Somma matrici:
    punt_A = testa A
    punt_B = testa B
    Nuova riga:
    Stampa a capo
    cont_colonne = 0
    Prossimo elemento:
        if ( punt_A == 0 )
            jump Sottrai matrici
        if ( cont_colonne == n )
            jump Nuova riga
        cont_colonne ++
        Stampa ( A[punt_A] + B[punt_B] )
        punt_A = A[next]
        punt_B = B[next]
        jump Prossimo elemento
    jump Main
Sottrai matrici
...

```

Moltiplica matrici:

cont\_elemento = 0

Nuova riga:

Stampa a capo

cont\_colonne = 0

Ciclo moltiplica:

if ( punt\_A == 0 )

jump Main

if ( cont\_colonne == n )

jump Nuova riga

num\_riga = cont\_elemento / n

cont\_elemento ++

cont\_colonna ++

Posizionati in cima alla colonna -> punt\_B\_temp

Posizionati in cima alla riga -> punt\_A\_temp

cont\_elementi\_moltiplicati = 0

somma = 0

Scorri:

if (cont\_elementi\_stampati == n )

jump Ciclo moltiplica

somma = somma + (A[punt\_A\_temp] x B[punt\_B\_temp])

if ( conta\_elementi\_moltiplicati == n )

Stampa ( somma )

punt\_A\_temp = A[next]

punt\_B\_temp = Elemento riga successiva

jump Scorri

jump Main

Esci

syscall = 10

### - Simulazione:

Inserimento matrice 2x2:

```
Menu:
a) Inserimento matrici
b) Somma di matrici
c) Sottrazione di matrici
d) Prodotto di matrici
e) Uscita dal programma

Scelta: a

Inserire grandezza matrice quadrata: 2

E' stata scelta una matrice: 2x2

Inserimento matrice A:
Inserire elemento: 1
Inserire elemento: 2
Inserire elemento: 3
Inserire elemento: 4

Inserimento matrice B:
Inserire elemento: 5
Inserire elemento: 6
Inserire elemento: 7
Inserire elemento: 8

Matrice A:

1      2
3      4

Matrice B:

5      6
7      8
```

Somma, sottrazione e moltiplicazione:

```
Menu:
a) Inserimento matrici
b) Somma di matrici
c) Sottrazione di matrici
d) Prodotto di matrici
e) Uscita dal programma

Scelta: b
Risultato di A + B:

6      8
10     12

Menu:
a) Inserimento matrici
b) Somma di matrici
c) Sottrazione di matrici
d) Prodotto di matrici
e) Uscita dal programma

Scelta: c
Risultato di A - B:

-4     -4
-4     -4

Menu:
a) Inserimento matrici
b) Somma di matrici
c) Sottrazione di matrici
d) Prodotto di matrici
e) Uscita dal programma

Scelta: d
Risultato di A X B:

19     22
43     50
```

### - Approfondimenti sul codice:

Anche in questo esercizio alcune funzioni ripetute vengono chiamate tramite la *jal* per questo spesso è stato necessario salvare l'indirizzo contenuto in *\$ra* nello stack, per poi essere ripristinato prima di effettuare la *jr*.

Ogni scelta corrispondeva poi alla chiamata a procedura tramite *jal*. infatti vengono passati gli argomenti alla procedura tramite i registri *\$a0*, *\$a1*, *\$a2* ...

Se viene inserita una grandezza della matrice minore o uguale a zero questa viene richiesta.

Dopo aver inserito entrambe le matrici queste vengono stampate.

Nel programma sono state utilizzate diverse stringhe *.ascii* per stampare messaggi su console. Principalmente è stato utilizzato il registro *\$t0* o *\$t8* come puntatore della matrice. Molti registri sono stati utilizzati come contatori per operazioni di controllo

Esempio nel caso vengano create inizialmente due matrici 2x2 e successivamente due matrici 3x3, vediamo come il programma utilizza le celle già create in precedenza:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268697600	1	268697608	2	268697616	3	268697624	4	0
268697632	5	268697640	6	268697648	7	268697656	8	0
268697664	0	0	0	0	0	0	0	0
268697696	0	0	0	0	0	0	0	0
268697728	0	0	0	0	0	0	0	0
268697760	0	0	0	0	0	0	0	0
268697792	0	0	0	0	0	0	0	0

0x10040000 (heap)

☐ Hexadecimal Addresses
 ☐ Hexadecimal Values
 ☐ ASCII

Mars Messages

Run I/O

Menu:  
a) Inserimento matrici  
b) Somma di matrici  
c) Sottrazione di matrici  
d) Prodotto di matrici  
e) Uscita dal programma  
  
Scelta: a  
  
Inserire grandezza matrice quadrata: 2  
  
E' stata scelta una matrice: 2x2  
  
Inserimento matrice A:  
Inserire elemento: 1  
Inserire elemento: 2  
Inserire elemento: 3  
Inserire elemento: 4  
  
Inserimento matrice B:  
Inserire elemento: 5  
Inserire elemento: 6  
Inserire elemento: 7  
Inserire elemento: 8  
  
Matrice A:  

1	2
3	4

Matrice B:  

5	6
7	8

Successivamente vengono create due matrici 3x3:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268697600	11	268697608	12	268697616	13	268697624	14	268697664
268697632	21	268697640	22	268697648	23	268697656	24	268697704
268697664	15	268697672	16	268697680	17	268697688	18	268697696
268697696	19	0	25	268697712	26	268697720	27	268697728
268697728	28	268697736	29	0	0	0	0	0
268697760	0	0	0	0	0	0	0	0
268697792	0	0	0	0	0	0	0	0

0x10040000 (heap)

☐ Hexadecimal Addresses
 ☐ Hexadecimal Values
 ☐ ASCII

Mars Messages

Run I/O

E' stata scelta una matrice: 3x3

Inserimento matrice A:

Inserire elemento: 11

Inserire elemento: 12

Inserire elemento: 13

Inserire elemento: 14

Inserire elemento: 15

Inserire elemento: 16

Inserire elemento: 17

Inserire elemento: 18

Inserire elemento: 19

Inserimento matrice B:

Inserire elemento: 21

Inserire elemento: 22

Inserire elemento: 23

Inserire elemento: 24

Inserire elemento: 25

Inserire elemento: 26

Inserire elemento: 27

Inserire elemento: 28

Inserire elemento: 29

Matrice A:

11	12	13
14	15	16
17	18	19

Matrice B:

21	22	23
24	25	26
27	28	29

Vediamo come viene riutilizzato lo spazio già allocato in precedenza e vediamo come la fine di ogni matrice ha il campo *elemento successivo* uguale a 0.



## CODICI:

### 1.

```
# Daniel Zanchi - daniel.zanchi@stud.unifi.it - consegna 30 Giugno 2015
# esercizio 1 del progetto: Identificatore di sotto-sequenze
.data
ms_array: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
ms_zero_array: .word 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
c2_array: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
c1_array: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

buffer: .space 101

fileNotFound: .ascii "Il file non è stato trovato: "
file: .asciiiz "/sequenza.txt"

str_benvenuto: .asciiiz "Daniel Zanchi, Esercizio n.1 del progetto \n\n\n"
str_sequenza_sul_file: .asciiiz "Sequenza sul file: "
str_numero_caratteri: .asciiiz "Numero caratteri sul file: "
str_new_line: .asciiiz "\n"
str_spazio: .asciiiz " "
str_valore_x: .asciiiz "\nValore inserito: "
str_inserire_x: .asciiiz "\n\nInserisci il valore di x da cercare: "
str_valore_x_minore: .asciiiz "\n\nE' stato inserito un valore minore di -511"
str_valore_x_maggiore: .asciiiz "\n\nE' stato inserito un valore maggiore di 511"
str_ms_array: .asciiiz "\nConversione modulo e segno: "
str_ms_zero_array: .asciiiz "\nConversione modulo e segno alt: "
str_c2_array: .asciiiz "\nConversione complemento a due: "
str_c1_array: .asciiiz "\nConversione complemento a uno: "
str_occorrenza_ms: .asciiiz "\nCaso MS - Elenco indici di inizio occorrenza: "
str_occorrenza_c2: .asciiiz "\nCaso C2 - Elenco indici di inizio occorrenza: "
str_occorrenza_c1: .asciiiz "\nCaso C1 - Elenco indici di inizio occorrenza: "
str_nessuna_occorrenza: .asciiiz "nessuna occorrenza"

#$s0 salvo il valore inserito in input
#s7 numero caratteri sequenza

.text

.globl main
#FUNZIONE PER ANDARE A CAPO
new_line:
    li $v0, 4
    la $a0, str_new_line
    syscall
    jr $ra

#FUNZIONE PER STAMPARE CON INDIRIZZO IN $a0
stampa_stringa:
    li $v0, 4          # Print String Syscall
    syscall
    jr $ra

#FUNZIONE PER STAMPARE CON valore IN $a0
stampa_intero:
    li $v0, 1
    syscall
    jr $ra

#FUNZIONE PER STAMPARE ARRAY CON INDIRIZZO IN $t0
stampa_array:
    move $t0, $a0
    li $t5, 40
    li $t1, 0          #la grandezza del vettore è 40 questo mi serve pr i controlli
```

```

        stampa_int_array:      #ciclo dopo aver salvato la variabile ra di ritorno
            lw $a0, 0($t0)      #salvo in $a0 l'intero contenuto nell'array ms_array
            li $v0, 1 #stampo l'intero contenuto in a0
            syscall
            li $v0, 4
            la $a0, str_spazio   #stampo uno spazio, per separare gli elementi
            syscall
            addi $t1, $t1, 4
            addi $t0, $t0, 4 #incremento il mio offset di 4 per passare all'intero successivo
            blt $t1, $t5, stampa_int_array #if $t1 (offset) < $t5 (40) salto a stampa_Array per stampare quelli
successivi
            jr $ra #se ha finito di leggere tutto l'array posso tornare.

```

```

#NEL CASO CHE IL VALORE INSERITO SIA MINORE DI -511 CHIEDO DI REINSERIRE IL VALORE
valore_minore:

```

```

    la $a0, str_valore_x_minore
    jal stampa_stringa
    j richiedi_input

```

```

#NEL CASO CHE IL VALORE INSERITO SIA MAGGIORE DI 511 CHIEDO DI REINSERIRE IL VALORE
valore_maggiore:

```

```

    la $a0, str_valore_x_maggiore
    jal stampa_stringa
    j richiedi_input

```

```

#FUNZIONE PER CALCOLARE IL COMPLEMENTO A DUE IN CASO L'INPUT FOSSE NEGATIVO

```

```

converti_c2_negativo:

```

```

    move $t0, $a1
    addi $t0, $t0, 1024 #aggiungo 1024 al valore inserito da input. poi esegue la conversione in binario

```

```

puro.

```

```

    move $a1, $t0
    j converti_binario #salto a convertire il numero $a1 in binario puro

```

```

# FUNZIONE PER CONVERTIRE IN BINARIO PURO, UN NUMERO IN $a1, SALVANDOLO IN ARRAY PARTENDO
DALL'INDIRIZZO $a0 + 40

```

```

converti_binario:

```

```

    li $t2, 2 # $t2 = 2 per eseguire la divisione per due.
    move $t3, $a1 #sposto il valore da dividere in $t3
    move $t0, $a0 #metto l'indirizzo dell'array in un registro temporaneo $t0
    addi $t0, $t0, 40 #incremento l'offset di 40 per andare in fondo all'array
    converti_binario_:
    div $t3, $t2 #divido il mio numero inserito (o quello che è stato diviso, quoziente) per due
    mfhi $t1 #metto il resto della divisione in $t1
    mflo $t3 #metto il quoziente della divisione in $t3
    addi $t0, $t0, -4 #decremento l'offset per andare alla word dell'array precedente
    sw $t1, 0($t0) #salvo il resto della divisione appena effettuata nel vettore
    bgtz $t3, converti_binario_ #se il nostro quoziente è maggiore di zero torno continuo con la

```

```

conversione

```

```

    jr $ra

```

```

converti_c1_negativo:

```

```

    move $t0, $a1 # $t0 = valore inserito da input (t0 verrà modificato)
    addi $t0, $t0, 1023 #aggiungo 1024 al valore inserito da input. poi esegue la conversione in binario

```

```

puro.

```

```

    move $a1, $t0 #salto a convertire a1 in binario puro
    j converti_binario

```

```

main:

```

```

#salvo tutti i valori dei registri che userò nel main

```

```

addi $sp, $sp, -12

```

```

sw $ra, 0($sp) #salvo nello stack il valore di $ra, (jal main + 4)

```

```

sw $s0, 4($sp) #salvo nello stack il valore di $s0

```

```

sw $s7, 8($sp) #salvo nello stack il valore di $s7

```

```

#stampo messaggio benvenuto

```

```

la $a0, str_benvenuto

```

```

jal stampa_stringa

```

Daniel Zanchi

Codici

```

# Apri File
li      $v0, 13          # Apri File Syscall
la      $a0, file # carica nome file
li      $a1, 0           # Read-only Flag
li      $a2, 0           # (ignored)
syscall

blt      $v0, 0, errore   # Goto Error
move     $t0, $v0 # Save File Descriptor

#leggi file
leggi:
    li      $v0, 14          # leggi File Syscall
    move $a0, $t0 # carica File Descriptor
    la      $a1, buffer      # carica Buffer Address
    li      $a2, 101 # Buffer Size (101 perchè 100 sono i caratteri massimi, e uno è la fine del file)
    syscall
    move $t2,$a1 #metto in $t2 l'indirizzo buffer

    la      $a0, str_numero_caratteri #stampo stringa "numeri caratteri sul file: "
    jal stampa_stringa
#conta numero caratteri
# move $t2, $t3 #metto in $t2 l'indirizzo dove iniziare a contare, indirizzo buffer (verrà modificato)
li $t1,0 #inializzo contatore caratteri a 0
prossimo_char:
    lb $t0,($t2) # leggo un carattere della stringa
    beqz $t0,stampaNumero # se è zero ho finito
    add $t1,$t1,1 # incremento il contatore caratteri
    add $t2,$t2,1 # incremento la posizione sulla stringa
    j prossimo_char # e continuo a leggere

stampaNumero:
    #stampo il numero di caratteri nella stringa
    move $s7, $t1 #mi salvo su #s7 il numero di caratteri sul file, mi servirà successivamente nel programma
    move $a0, $t1
    jal stampa_intero

jal new_line

#stampo la sequenza del file
la      $a0, str_sequenza_sul_file
jal stampa_stringa
la $a0, buffer #a0 = indirizzo sequenza su buffer
jal stampa_stringa

# chuido il File
chiudo:
    li      $v0, 16          # Close File Syscall
    move     $a0, $t0 # Load File Descriptor
    syscall
    j      richiedi_input      # salta a richiedere il numero input

# se riscontra un errore nella lettura (es. non trova file)
errore:
    # Print String Syscall
    la      $a0, fileNotFound # Load Error String
    jal stampa_stringa
    j end

richiedi_input:
    #RICHIEDO IN INPUT IL VALORE X
    la $a0, str_inserire_x
    jal stampa_stringa
    #LEGGO IN INPUT UN NUMERO
    li $v0, 5 # codice leggi intero = 5

```

```

syscall
#CONTROLO SE IL NUMERO IN INPUT VA BENE!
addi $t2, $zero, -511      #metto in $t2 -512 per controllo
blt $v0, $t2, valore_minore #se $v0 < -511 salto alla label "valore minore"
addi $t2, $zero, 511       #metto in $t2 512 per controllo
bgt $v0, $t2, valore_maggiore #se $v0 > 511 salto alla label "valore maggiore"

```

```

move $s0, $v0    #$s0 contiene il valore messo in input.

```

```

#stampo "valore inserito"
la $a0, str_valore_x
jal stampa_stringa

```

```

move $a0, $s0
jal stampa_intero

```

```

#CONVERTO IN MODULO E SEGNO

```

```

la $a0, ms_array #carico in $t0 l'indirizzo di ms_array
move $a1, $s0    #sposto il valore inserito in t1 che verrà poi modificato

```

```

#FUNZIONE PER CONTROLLARE IL BIT DEL SEGNO IN MS

```

```

bit_segno:

```

```

bgez $a1, positivo #if $a1 (input) >=0 salta a positivo
li $t2, 1          #carico in $t2 1 da mettere nella prima posizione del vettore nel caso il numero inserito sia negativo
sw $t2, 0($a0)     #metto nella prima posizione del vettore il bit 1
abs $a1, $a1       #faccio il valore assoluto del mio numero, utile per fare operazioni successivamente, lo metto in

```

```

$a1

```

```

positivo:
jal converti_binario    #convertito in binario puro

```

```

#CONVERTO COMPLEMENTO A DUE

```

```

la $a0, c2_array #t0 = indirizzo array complemento a due
move $a1, $s0    # $a1 = valore input (verrà modificato)
bgezal $s0, converti_binario #in caso di numero positivo salto a converti binario puro
bltzal $s0, converti_c2_negativo #in caso di numero negativo vado a modificare il numero input

```

```

finito_c2:

```

```

#CONVERTO COMPLEMENTO A UNO

```

```

la $a0, c1_array # $t0 = indirizzo array complemento a uno
move $a1, $s0    # $a1 = valore inserito in input (verrà modificato)
bgezal $s0, converti_binario #in caso di numero positivo salto a converti binario puro
bltzal $s0, converti_c1_negativo #in caso di numero negativo vado a modificare il numero input

```

```

#STAMPO I NUMERI CONVERTITI

```

```

#stampo modulo e segno
la $a0, str_ms_array
jal stampa_stringa
la $a0, ms_array
jal stampa_array

```

```

#se il numero inserito in input è 0 stampo anche la conversione alternativa allo 0 in MS

```

```

bnez $s0, stampo_c2
la $a0, str_ms_zero_array
jal stampa_stringa
la $a0, ms_zero_array
jal stampa_array

```

```

stampo_c2:

```

```

#stampo complemento a due
la $a0, str_c2_array
jal stampa_stringa
la $a0, c2_array
jal stampa_array

```

```

#stampo complemento a uno

```

```

la $a0, str_c1_array

```

Daniel Zanchi

Codici

```

jal stampa_stringa
la $a0, c1_array
jal stampa_array

la $a0, str_new_line
jal stampa_stringa

#INIZIO CON IL CONTROLLO MS
#stampo la stringa "occorrenze MS"
la $a0, str_occorrenza_ms
jal stampa_stringa

li $t8, 0 #rimane 0 se non viene trovata una occorrenza
li $t5, 10 #uso per il controllo dei caratteri da controllare nella sequenza.
li $t7, 0 #conta indice occorrenza

la $t0, buffer #in $1 avrò l'indirizzo della sequenza

#se l'input == 0 controllo con la sua conversione alternativa
controllo_ms_zero:
bnez $s0, controllo_ms # se input != 0 salto a controllo_ms

#INIZIO CON IL CONTROLLO MS se input è 0
move $t6, $s7 # s7 = numero caratteri in sequenza
li $t7, 0 #conta indice occorrenza
li $t5, 10 #uso per il controllo dei caratteri da controllare nella sequenza.

next_char_no_occ_zero:
li $t9, 0 #serve per contare il numero di occorrenze. appena arriva a 10 allora abbiamo trovato il numero nella
sequenza
la $t2, ms_zero_array #in $t2 avrò l'indirizzo dell'array modulo e segno
move $t1, $t0 #t1 = indirizzo buffer, t1 verrà incrementato
add $t1, $t7, $t1 #calcolo l'offset per leggere il carattere dal buffer
addi $t7, $t7, 1 #incremento indice per spostarmi nel buffer ogni volta che non viene trovata un'occorrenza

blt $t6, $t5, controllo_ms #se il numero di caratteri da leggere è < di 10 salta a finito
addi $t6, $t6, -1 #numero caratteri = numero caratteri - 1
next_char_zero:
lb $t3, ($t1) #metto in $t3 il valore del carattere letto
#adesso converto il carattere letto dal buffer e lo converto o in 1 o in 0
converti_char_int_ms0:
li $t4, 49 #metto in $s5 il valore 49 che equivale allo 1
beq $t3, $t4, converti_in_uno_ms0 #se $t3 (carattere letto dalla sequenza) == 49 (1) allora salta a
converti_in_uno, altrimenti converte in 0
li $t3, 0 #adesso in $t3 abbiamo il valore 0
j convertito_ms0
converti_in_uno_ms0:
li $t3, 1 #adesso in $t3 abbiamo il valore 1
convertito_ms0:
lw $t4, ($t2) #t4 contiene il valore letto da ms_array
addi $t1, $t1, 1 # incremento la posizione sul buffer
bne $t3, $t4, next_char_no_occ_zero #if $t3 != $t4 allora vado avanti con la sequenza. altrimenti sono = e posso
incrementare tutti e due.
addi $t9, $t9, 1 #incremento il numero di occorrenze trovate, se arriva a 10 ha trovato il numero
beq $t5, $t9, trovato_ms_zero #se ho trovato 10 caratteri uguali allora ho trovato un'occorrenza e posso
saltare a trovato_ms_zero
addi $t2, $t2, 4 #incremento il mio offset di 4 per passare all'intero successivo del ms_array
addi $t2, $t2, 4 #incremento il mio offset di 4 per passare all'intero successivo del ms_array
j next_char_zero #altrimenti continuo a scorrere la sequenza
trovato_ms_zero:
li $t8, 1 #cambio il flag a 1, ho trovato un'occorrenza, utile per non stampare il messaggio "nessuna occorrenza"
move $a0, $t7 #stampo l'indice di inizio della occorrenza appena trovata
li $v0, 1
syscall
la $a0, str_spazio
li $v0, 4

```

```

        syscall
        j next_char_no_occ_zero      #continuo a scorrere la sequenza

#CONTROLLO MS
controllo_ms:
move $t6, $s7    # s7 = numero caratteri in sequenza
li $t7, 0 #conta indice occorrenza
li $t5, 10      #uso per il controllo dei caratteri da controllare nella sequenza.

next_char_no_occ:
    li $t9, 0 #t9 conta il numero di occorrenze. appena arriva a 10 allora abbiamo trovato il numero nella sequenza

    la $t2, ms_array #in $t2 avrò l'indirizzo dell'array modulo e segno
    move $t1, $t0     #t1 = indirizzo buffer, verrà incrementato
    add $t1, $t7, $t1 #calcolo l'offset per leggere il carattere dal buffer
    addi $t7, $t7, 1  #incremento indice per spostarmi nel buffer ogni volta che non viene trovata un'occorrenza

    blt $t6, $t5, finito_controllo_ms #se il numero di caratteri della sequenza da leggere è < di 10 salta a finito
    addi $t6, $t6, -1 #numero caratteri = numero caratteri - 1
next_char:
    lb $t3, ($t1)      #metto in $t3 il valore del carattere letto
    #adesso converto il carattere letto dal buffer e lo converto a 0 o 1 o in 0
    converti_char_int_ms:
        li $t4, 49      #metto in $s5 il valore 49 che equivale allo 1
        beq $t3, $t4, converti_in_uno_ms #se $t3 (carattere letto dalla sequenza) == 49 (1) allora salta a
converti_in_uno, altrimenti converte in 0
        li $t3, 0 #adesso in $t3 abbiamo il valore 0
        j convertito_ms
        converti_in_uno_ms:
            li $t3, 1 #adesso in $t3 abbiamo il valore 1
    convertito_ms:

    lw $t4, ($t2)      #$t4 contiene il valore letto da ms_array
    addi $t1, $t1, 1 # incremento la posizione sul buffer
    bne $t3, $t4, next_char_no_occ #if $t3 != $t4 allora vado avanti con la sequenza. altrimenti sono = e posso
incrementare tutti e due.

    addi $t9, $t9, 1 #incremento il numero di occorrenze trovate, se arriva a 10 ha trovato il numero
    beq $t5, $t9, trovato_ms # se il numero di caratteri trovati == a 10 allora ho trovato un'occorrenza e salto
    addi $t2, $t2, 4 #incremento il mio offset di 4 per passare all'intero successivo del ms_array
    j next_char

trovato_ms:
    li $t8, 1 #t8 = flag che ho trovato un'occorrenza. utile per non stampare messaggio "nessuna occorrenza"
    move $a0, $t7 #stampo l'indice dove è partita l'occorrenza
    li $v0, 1
    syscall
    la $a0, str_spazio
    li $v0, 4
    syscall
    j next_char_no_occ      #salto a continuare a leggere la sequenza

finito_controllo_ms:
    bnez $t8, controllo_c2 #controllo se il flag == 0 stampo il messaggio "nessuna occorrenza", altrimenti salto
a controllo_C2
    la $a0, str_nessuna_occorrenza
    li $v0, 4
    syscall

controllo_c2:
#INIZIO CON IL CONTROLLO C2
move $t6, $s7    # s7 = numero caratteri in sequenza
li $t8, 0 #rimane 0 se non viene trovata una occorrenza
li $t7, 0 #conta indice occorrenza

#stampo la stringa "occorrenze C2"

```

```

la $a0, str_occorrenza_c2
li $v0, 4
syscall

```

next\_char\_no\_occ\_c2:

```

li $t9, 0 #serve per contare il numero di occorrenze. appena arriva a 10 allora abbiamo trovato il numero nella
sequenza

```

```

la $t2, c2_array #in $2 avrò l'indirizzo dell'array complemento a 2
move $t1, $t0 #t1 = indirizzo buffer, verrà incrementato
add $t1, $t7, $t1 #calcolo l'offset per leggere il carattere dal buffer
addi $t7, $t7, 1 #incremento indice per spostarmi nel buffer ogni volta che non viene trovata un'occorrenza

```

```

blt $t6, $t5, finito_confronto_c2 #se il numero di caratteri da leggere è < di 10 salta a finito
addi $t6, $t6, -1 #numero caratteri = numero caratteri -1

```

next\_char\_c2:

```

lb $t3, ($t1) #metto in $t3 il valore del carattere letto
#adesso converto il carattere letto dal buffer e lo converto o in 1 o in 0

```

converti\_char\_int\_c2:

```

li $t4, 49 #metto in $s5 il valore 49 che equivale allo 1
beq $t3, $t4, converti_in_uno_c2 #se $t3 (carattere letto dalla sequenza) == 49 (1) allora salta a

```

converti\_in\_uno, altrimenti converte in 0

```

li $t3, 0 #adesso in $t3 abbiamo il valore 0

```

j convertito\_c2

converti\_in\_uno\_c2:

```

li $t3, 1 #adesso in $t3 abbiamo il valore 1

```

convertito\_c2:

```

lw $t4, ($t2) #t4 contiene il valore letto da ms_array

```

```

addi $t1, $t1, 1 # incremento la posizione sul buffer

```

bne \$t3, \$t4, next\_char\_no\_occ\_c2 #if \$t3 != \$t4 allora vado avanti con la sequenza. altrimenti sono = e posso incrementare tutti e due.

```

addi $t9, $t9, 1 #incremento il numero di caratteri trovati uguali, se arriva a 10 ha trovato un'occorrenza

```

```

beq $t5, $t9, trovato_c2 #se s6 == 10 allora ho trovato un'occorrenza

```

```

addi $t2, $t2, 4 #incremento il mio offset di 4 per passare all'intero successivo del ms_array

```

j next\_char\_c2

trovato\_c2:

```

li $t8, 1 # t8 = 1 per non stampare il messaggio "nessuna occorrenza"

```

```

move $a0, $t7 #stampo l'indice della occorrenza appena trovata

```

```

li $v0, 1

```

```

syscall

```

```

la $a0, str_spazio

```

```

li $v0, 4

```

```

syscall

```

```

j next_char_no_occ_c2 #continuo a scorrere la sequenza

```

finito\_confronto\_c2:

```

bnez $t8, controllo_c1 #se t8 == 0 allora non ho trovato nessuna occorrenza e posso stampare il

```

messaggio "nessuna occorrenza", altrimenti salto

```

la $a0, str_nessuna_occorrenza

```

```

li $v0, 4

```

```

syscall

```

controllo\_c1:

#INIZIO CON IL CONTROLLO C1

```

move $t6, $s7 #s7 = numero di caratteri in sequenza

```

```

li $t8, 0 #rimane 0 se non viene trovata una occorrenza

```

```

li $t7, 0 #conta indice occorrenza

```

#stampo la stringa "occorrenze C1"

```

la $a0, str_occorrenza_c1

```

```

li $v0, 4

```

```

syscall

```

next\_char\_no\_occ\_c1:

```

li $t9, 0 #serve per contare il numero di occorrenze. appena arriva a 10 allora abbiamo trovato il numero nella
sequenza

```

```

la $t2, c1_array #in $2 avrò l'indirizzo dell'array complemento a 1

```

```

move $t1, $t0          #t1 = indirizzo buffer, verrà incrementato
add $t1, $t7, $t1      #calcolo l'offset per leggere il carattere dal buffer
addi $t7, $t7, 1       #incremento indice per spostarmi nel buffer ogni volta che non viene trovata un'occorrenza

blt $t6, $t5, finito_confronto_c1    #se il numero di caratteri da leggere è < di 10 salta a finito
addi $t6, $t6, -1    #numero caratteri = numero caratteri -1
next_char_c1:
    lb $t3, ($t1)      #metto in $t3 il valore del carattere letto
    #adesso converto il carattere letto dal buffer e lo converto o in 1 o in 0
    converti_char_int_c1:
        li $t4, 49      #metto in $s5 il valore 49 che equivale allo 1
        beq $t3, $t4, converti_in_uno_c1    #se $t3 (carattere letto dalla sequenza) == 49 (1) allora salta a
converti_in_uno, altrimenti converte in 0
        li $t3, 0    #adesso in $t3 abbiamo il valore 0
        j convertito_c1
        converti_in_uno_c1:
            li $t3, 1    #adesso in $t3 abbiamo il valore 1
    convertito_c1:
        lw $t4, ($t2)    #$t4 contiene il valore letto da ms_array
        addi $t1, $t1, 1    # incremento la posizione sul buffer
        bne $t3, $t4, next_char_no_occ_c1    #if $t3 != $t4 allora vado avanti con la sequenza. altrimenti sono = e posso
incrementare tutti e due.
        addi $t9, $t9, 1    #incremento il numero di caratteri uguali trovati, se arriva a 10 ha trovato una occorrenza
        beq $t5, $t9, trovato_c1    #se numero caratteri trovati uguali == 10 allora salto, perchè ho trovato una
occorrenza
        addi $t2, $t2, 4    #incremento il mio offset di 4 per passare all'intero successivo del ms_array
        j next_char_c1
trovato_c1:
    li $t8, 1    #setto il flag a 1 perchè ho trovato un'occorrenza, per non stampare il messaggio "nessuna occorrenza"
    move $a0, $t7
    li $v0, 1
    syscall
    la $a0, str_spazio
    li $v0, 4
    syscall
    j next_char_no_occ_c1

finito_confronto_c1:
    bnez $t8, end    #se non ho trovato occorrenze stampo il messaggio "nessuna occorrenza", altrimenti salto a
fine
    la $a0, str_nessuna_occorrenza
    jal stampa_stringa

end:
    lw $s7, 8($sp)    #ripristino dallo stack il valore di $ra, (jal main + 4)
    lw $s0, 4($sp)    #ripristino dallo stack il valore di $s0
    lw $ra, 0($sp)    #ripristino dallo stack il valore di $s7
    addi $sp, $sp, 12

    li $v0, 10        #syscall per chiudere il programma
    syscall

```



## 2.

# Daniel Zanchi - daniel.zanchi@stud.unifi.it - consegna 30 Giugno 2015

# esercizio 2 del progetto: Procedure annidate e ricorsive

.data

```
str_benvenuto: .asciiz "Daniel Zanchi, Esercizio n.2 del progetto \n\n\n"
str_inserire: .asciiz "Inserire il valore n: "
str_traccia: .asciiz "Traccia:\n"
str_risultato: .asciiz "\n\nRisultato: G("
str_risultato2: .asciiz ")="
str_p_freccia: .asciiz ") --> "
str_g: .asciiz "G("
str_f: .asciiz "F("
str_f_return: .asciiz "F-return("
str_g_return: .asciiz "G-return("
str_p: .asciiz ")"
str_overflow: .asciiz "\n\nE' stato rilevato un errore di overflow\n\n"
```

.text

.globl main

stampa\_stringa:

```
    li      $v0, 4
    syscall
    jr $ra
```

stampa\_intero:

```
    li $v0, 1
    syscall
    jr $ra
```

main:

```
addi $sp, $sp, -16
sw $ra, 0($sp)    #salvo nello stack il valore di $ra, (jal main + 4)
sw $s0, 4($sp)    #salvo nello stack il valore di $s0
sw $s1, 8($sp)    #salvo nello stack il valore di $s1
sw $s2, 12($sp)   #salvo nello stack il valore di $s2
```

```
la $a0, str_benvenuto    #stampo stringa di benvenuto, con nome cognome e n esercizio
jal stampa_stringa
```

chiedo\_n:

```
    la $a0, str_inserire    #stampo stringa "inserire il valore n: "
    jal stampa_stringa
    #LEGGO IN INPUT UN NUMERO
    li $v0, 5                # codice leggi intero = 5
    syscall                  #salva int inserito in $v0
    bltz $v0, chiedo_n       #controllo se il valore inserito è minore di zero
    move $s2, $v0            #salvo valore inserito in $s2
```

```
la $a0, str_traccia        #stampo "traccia: ""
jal stampa_stringa
```

#stampo "G(n)-->"

```
la $a0, str_g                #stampo "G("
jal stampa_stringa
move $a0, $s2                #stampo il valore inserito in input
jal stampa_intero
la $a0, str_p_freccia        #stampo ")-->"
jal stampa_stringa
```

```
move $a1, $s2    #sposto il valore inserito in input in a1
jal procedura_g
```

## #PROCEDURA G(N)

```
procedura_g:
    li $s0, 0 # $s0 = b
    li $s1, 0 # $s1 = k
    ciclo_for: # ciclo iterativo (for)
        bgt $s1, $a1, return_b # se k > n (inserito) allora ho finito il ciclo for
        move $a0, $s1 # spostato k in $a0, questo verrà usato dentro la procedura f(k)

        jal procedura_f # salto a fare la procedura F(k)

        mul $s0, $s0, $s0 # eseguo b = b * b
        mfhi $t3 # prendo il secondo registro da 32 bit della moltiplicazione
        beqz $t3, ok # se registro hi diverso da zero c'è stato un overflow, altrimenti

    salto e vado avanti
        la $a0, str_overflow # stampo errore di overflow
        jal stampa_stringa
        j chiedo_n # salto per reinserire n
    ok:
        add $s0, $s0, $v0 # b = b + u, u=v0 valore di ritorno dalla ricorsione
        addi $s1, $s1, 1 # incremento k

        j ciclo_for # ciclo for
    return_b:
        # stampo G-return
        la $a0, str_g_return
        jal stampa_stringa
        move $a0, $s0
        jal stampa_intero
        la $a0, str_p
        jal stampa_stringa

        # stampo prima parte stringa risultato
        la $a0, str_risultato
        jal stampa_stringa
        # stampo il numero n inserito
        move $a0, $a1
        jal stampa_intero
        # stampo seconda parte stringa risultato
        la $a0, str_risultato2
        jal stampa_stringa
        move $a0, $s0
        jal stampa_intero

    j end
```

## #PROCEDURA F(N)

```
procedura_f:
    addi $sp, $sp, -8 # crea spazio per due words nello stack frame
    sw $ra, 4($sp) # salva l'indirizzo di ritorno al chiamante
    sw $a0, 0($sp) # salva il parametro d'invocazione

    # #STAMPA F()
    la $a0, str_f # stampo f(
    li $v0, 4
    syscall
    lw $a0, 0($sp) # stampo l'argomento di f(), prendendolo dallo stack
    li $v0, 1
    syscall
    la $a0, str_p_freccia # stampo )-->
    li $v0, 4
    syscall

    lw $a0, 0($sp) # ripristino in $a0 l'argomento di f, prendendolo dallo stack

    bnez $a0, else # if (n != 0) salta a else
```

```

#STAMPO F-RETURN CASO BASE ( n = 0 )
#stampo il valore F-return
la $a0, str_f_return
li $v0, 4
syscall
li $a0, 1 #stampo 1, perchè se n = 0 il valore di ritorno di f è 1.
li $v0, 1
syscall
la $a0, str_p_freccia
li $v0, 4
syscall

lw $a0, 0($sp)          #ripristino il valore di $a0, prendendolo dallo stack

li $v0, 1                # n = 0, quindi ritorna 1
addi $sp,$sp,8          # ripristino lo stack frame
jr $ra                  # ritorna al chiamante

else: # n != 0
    addi $a0,$a0,-1      #decremento il K
    jal procedura_f      # chiamata ricorsiva a ricorsione (n-1)

    lw $a0,0($sp)        # ripristina i valori salvati in precedenza nello stack frame: parametro e indirizzo di ritorno
    lw $ra,4($sp)        # ripristina il registro ra dallo stack frame

    add $v0, $v0, $v0      #          2*f(n-1)
    add $v0, $a0, $v0      #          n + 2f(n-1)
    move $t0, $v0         #salvo il valore di ritorno in s3

#STAMPO F-RETURN CASO NON BASE
#stampo il valore F-return
la $a0, str_f_return
li $v0, 4
syscall
move $a0, $t0
li $v0, 1
syscall
la $a0, str_p_freccia
li $v0, 4
syscall

lw $a0, 0($sp)
lw $ra, 4($sp)
move $v0, $t0
addi $sp,$sp,8          # ripristina lo stack frame

jr $ra                  # ritorna al chiamante

end:

lw $s2, 12($sp)         #ripristino dallo stack il valore di $s2
lw $s1, 8($sp)          #ripristino dallo stack il valore di $s1
lw $s0, 4($sp)          #ripristino dallo stack il valore di $s0
lw $ra, 0($sp)          #ripristino dallo stack il valore di $ra
addi $sp, $sp, 16        #ripristino lo stack pointer al suo punto iniziale

li $v0, 10
syscall

```

### 3.

# Daniel Zanchi - daniel.zanchi@stud.unifi.it - consegna 30 Giugno 2015

# esercizio 3 del progetto: Operazioni fra matrici

.data

str\_benvenuto: .ascii "Daniel Zanchi, Esercizio n.3 del progetto \n"

str\_menu: .ascii "\nMenu:\na) Inserimento matrici\nb) Somma di matrici\nc) Sottrazione di matrici\nd) Prodotto di matrici\ne) Uscita dal programma\n\nSceita: "

str\_a\_capo: .ascii "\n"

str\_inserire\_n: .ascii "\n\nInserire grandezza matrice quadrata: "

str\_matrice\_scelta: .ascii "\n\nE' stata scelta una matrice: "

str\_x: .ascii "x"

str\_coda\_non\_vuota: .ascii "\n\nLa coda non è vuota"

str\_nuovo\_elemento: .ascii "Inserire elemento: "

str\_spazio: .ascii "\t"

str\_inserimento\_a: .ascii "\nInserimento matrice A:\n"

str\_inserimento\_b: .ascii "\nInserimento matrice B:\n"

str\_matrice\_a: .ascii "\nMatrice A:\n"

str\_matrice\_b: .ascii "\n\nMatrice B:\n"

str\_a\_piu\_b: .ascii "\nRisultato di A + B:\n"

str\_a\_meno\_b: .ascii "\nRisultato di A - B:\n"

str\_a\_per\_b: .ascii "\nRisultato di A X B:\n"

.text

.globl main

stampa\_stringa:

```
    li    $v0, 4          # syscall 4 stampa stringa con indirizzo in $a0
    syscall
    jr    $ra
```

stampa\_intero:

```
    li    $v0, 1          # syscall 1 stampa intero contenuto in $a0
    syscall
    jr    $ra
```

#INSERISCI MATRICE (scelta a)

inserisci\_matrice:

addi \$sp, \$sp, -4

sw \$ra, 0(\$sp) #salvo il valore di ritorno \$ra nello stack

chiedi\_grandezza:

```
    la    $a0, str_inserire_n          #stampo messaggio "inserire grandezza matrice: "
    jal    stampa_stringa
```

li \$v0, 5 # leggo n

syscall #salva int inserito in \$v0

blez \$v0, chiedi\_grandezza #se v0 <= 0 salta a richiedere n, altrimenti vado avanti

move \$t5, \$v0 #sposto il valore delle righe inserite in \$t5

addi \$sp, \$sp, -4

sw \$t5, 0(\$sp) #salvo nello stack la grandezza della mia matrice

#stampo messaggio "matrice scelta n x n"

la \$a0, str\_matrice\_scelta

li \$v0, 4

syscall

move \$a0, \$t5

li \$v0, 1

syscall

la \$a0, str\_x

li \$v0, 4

syscall

move \$a0, \$t5

li \$v0, 1

syscall

la \$a0, str\_a\_capo

li \$v0, 4

syscall

la \$a0, str\_inserimento\_a #stampo messaggio "inserimento matrice a:"  
li \$v0, 4  
syscall

mul \$t1, \$t5, \$t5 #t1 contiene il numero di celle necessarie per la matrice

move \$a1, \$t1 #a1 contiene il numero di celle necessarie per la mia matrice  
move \$a0, \$a2 #a0 = testa matrice A, verrà modificato  
jal riempi\_matrice #salto a inserire elementi matrice A  
addi \$sp, \$sp, -4  
sw \$v0, 0(\$sp) #salvo nello stack la testa della matrice A

la \$a0, str\_inserimento\_b #stampo messaggio "inserimento matrice b:"  
li \$v0, 4  
syscall

lw \$t5, 4(\$sp) #carico dallo stack la grandezza della mia matrice  
mul \$t1, \$t5, \$t5 # t1 = numero di celle necessarie  
move \$a1, \$t1 #a1 numero celle necessarie per la matrice  
move \$a0, \$a3 #a0 testa matrice B  
jal riempi\_matrice #stampo a inserire elementi matrice B  
addi \$sp, \$sp, -4  
sw \$v0, 0(\$sp) #salvo nello stack la testa della matrice B  
lw \$t0, 12(\$sp) #ripristino dallo stack il valore di ra, per tornare al chiamante (jal  
insesci\_matrici + 4)  
jr \$t0

riempi\_matrice: #testa della mia matrice contenuta in a0  
move \$t3, \$a0 #sposto la mia testa in t3  
move \$t2, \$a1 #sposto numero di blocchi necessari in t2  
addi \$sp, \$sp, -4  
sw \$ra, 0(\$sp) #salvo nello stack il valore di \$ra, (jal main + 4)  
move \$t4, \$zero #inizializzo il contatore di blocchi inseriti a 0  
move \$t8, \$t3  
riempi\_matrice\_:  
addi \$t4, \$t4, 1 #incremento contatore elementi inseriti  
bgt \$t4, \$t2, matrice\_piena #se il mio contatore di blocchi inseriti è >= a il  
numero dei blocchi necessari per la matrice salto a stampare la matrice

move \$t0, \$t3 #mi muovo alla cella successiva  
beqz \$t3, crea\_cella #se t3 == 0 salto a creare una nuova cella di memoria,  
altrimenti utilizzo le celle già create

la \$a0, str\_nuovo\_elemento #stampo messaggio "inserire nuovo  
elemento"  
li \$v0, 4  
syscall

li \$v0, 5  
syscall # legge un intero  
move \$t1 \$v0 # t1=input

lw \$t3, 4(\$t0) # \$t3 = campo elemento successivo

sw \$t1, 0(\$t0) # campo intero = \$t1  
move \$t9, \$t0 # t9 = coda

beqz \$t3, crea\_cella\_incr #se t3 (che contiene l'indirizzo dell'elemento successivo)  
== 0 allora ho finito le celle già create e salto a crearne nuove  
j riempi\_matrice\_ #altrimenti vado avanti con le celle già create

crea\_cella\_incr:  
addi \$t4, \$t4, 1 #incremento contatore elementi inseriti

```

crea_cella:
    bgt $t4, $t2, inserito    #se il mio contatore di blocchi inseriti è >= a il
numero dei blocchi necessari per la matrice salto a stampare la matrice
    la $a0, str_nuovo_elemento    #stampo messaggio "inserire
nuovo elemento"

    li $v0, 4
    syscall

    li $v0, 5
    syscall    # legge un intero
    move $t1 $v0    # t1 = input

    li $v0, 9    # inizio inserzione nuovo elemento (chiamata sbrk)
    li $a0, 8    #crea un blocco di 8 byte
    syscall    # chiamata sbrk: restituisce un blocco di 8 byte, puntato da v0:
il nuovo record

    # vegono riempiti i due campi del nuovo record:
    sw $t1, 0($v0)    # campo intero = t1
    sw $zero, 4($v0)    # campo elemento-successivo = nil

    bne $t8, $zero, collega_ultimo    # se t8!=nil (coda non vuota) vai a
collega_ultimo

    move $t8, $v0    # altrimenti (prima inserzione) Testa=Coda=v0, $t8 sarà
la testa della matrice

    move $t3, $v0    # aggiorno la testa della mia matrice
    move $t9, $v0    # aggiorno la coda della mia matrice

    addi $t4, $t4, 1    #incremento contatore elementi inseriti
    j crea_cella    # torna all'inizio del loop di input, per inserire altri elementi

    collega_ultimo:    # se la coda e' non vuota, collega l'ultimo elemento
della lista,
                                # puntato da Coda
(t9) al nuovo record; dopodiche' modifica Coda per farlo puntare al nuovo record
    sw $v0, 4($t9)    # il campo elemento successivo dell'ultimo del record
prende v0

    move $t9, $v0    # Coda = v0, aggiorno la coda all'ultimo elemento
inserito

    addi $t4, $t4, 1    #incremento contatore elementi inseriti
    j crea_cella    # torna all'inizio del loop di input, per inserire altri
elementi

matrice_piena:
    sw $zero, 4($t0)    #in caso di matrice riempita metto a zero il
campo elemento successivo dell'ultimo elemento
    inserito:
        move $v0, $t8
        lw $ra, 0($sp)    #ripristino dallo stack il valore di $ra
        addi $sp, $sp, 4    #ripristino lo stack pointer al suo punto iniziale
        jr $ra    #salto all'indirizzo del registro
salvato in precedenza, per andare a stampare la matrice

#SOMMA MATRICI (scelta b)
somma:
    move $t0, $a0    #sposto la testa della matrice a in t0
    move $t1, $a1    #sposto la testa della matrice B in t1
    somma_nuova_riga:
    la $a0, str_a_capo    #stampo "\n"
    li $v0, 4
    syscall
    move $t4, $zero    #setto il contatore delle colonne a 0
    loop_somma:
        beq $t0, $zero, somma_eseguita    # se t0 == 0 si e' raggiunta la fine della matrice e salto a
menu
        beq $t4, $a2, somma_nuova_riga    #se s2(numero colonne stampate)== a2 (numero colonne matrice)
salto a nuova riga, dobbiamo andare a capo.

```

```

        addi $t4, $t4, 1          # incremento contatore colonne stampate
        lw $t2, 0($t0)           # t2 = valore del campo intero matrice A
        lw $t3, 0($t1)           # t3 = valore del campo intero matrice B
        add $a0, $t2, $t3        # A[i] + B[i] = $a0
        li $v0, 1                # stampo la somma appena ottenuta
        syscall
        li $v0, 4                # stampo uno spazio, per separare un elemento dall'altro
        la $a0, str_spazio
        syscall
        lw $t0, 4($t0)           # t0 = valore del campo elemento-successivo dell'elemento
corrente matrice A
        lw $t1, 4($t1)           # t1 = valore del campo elemento-successivo dell'elemento
corrente matrice b
        j loop_somma             # continuo a sommare le due matrici
somma_eseguita:
        jr $ra

#SOTTRAIRI MATRICI (scelta c)
sottrai:
        move $t0, $a0            #t0 testa matrice A
        move $t1, $a1            #t1 testa matrice B
sottrai_nuova_riga:
        la $a0, str_a_capo       #stampo "\n", per andare a capo
        li $v0, 4
        syscall
        move $t4, $zero          #setto il contatore delle colonne a 0
loop_sottrai:
        beq $t0, $zero, sottrazione_eseguita          # se t0 == 0 si e' raggiunta la fine della
matrice e si salto al menu
        beq $t4, $a2, sottrai_nuova_riga              #se s2 (numero colonne stampate) == a2 (numero
colonne matrice) salto a nuova riga, dobbiamo andare a capo.
        addi $t4, $t4, 1          #incremento contatore colonne stampate
        lw $t2, 0($t0)           # t2 = valore del campo intero matrice A
        lw $t3, 0($t1)           # t3 = valore del campo intero matrice B
        sub $a0, $t2, $t3        # A[i] - B[i] = $a0
        li $v0, 1                # stampo la sottrazione appena ottenuta
        syscall
        li $v0, 4                # stampo lo spazio per separare un elemento dall'altro
        la $a0, str_spazio
        syscall
        lw $t0, 4($t0)           # t0 = valore del campo elemento-successivo
dell'elemento corrente matrice A
        lw $t1, 4($t1)           # t1 = valore del campo elemento-successivo
dell'elemento corrente matrice A
        j loop_sottrai           # continuo a sottrarre le matrici
sottrazione_eseguita:
        jr $ra

#MULTIPLICA MATRICI (scelta d)
moltiplica:
        addi $sp, $sp, -12
        sw $ra, 0($sp)
        sw $s7, 4($sp)
        sw $s3, 8($sp)
        move $t0, $a1            #t0 = testa matrice A temporanea
        move $t1, $a2            #t1 = testa matrice B temporanea
        move $t9, $zero          #t9 = contatore per sapere a che elemento mi trovo

mul_nuova_riga:
        la $a0, str_a_capo       #stampo "\n" per andare a capo ogni volta che moltiplico una riga.
        li $v0, 4
        syscall
        move $s7, $zero          # contatore che metto a zero, conta il numero di colonne moltiplicate
loop_moltiplica:
        beq $t0, $zero, matrici_moltiplicate          # se t0 == 0 si e' raggiunta la fine della matrice
e si esce

```

```

    beq $s7, $a3, mul_nuova_riga    #se s7 == a3 salto a nuova riga, dobbiamo andare a
capo.
    move $s3, $zero                #registro che uso per operazioni, conterrà il risultato
    div $t3, $t9, $a3              # t3 = numero della riga in cui mi trovo
    addi $t9, $t9, 1               # incremento contatore, mi dice a che elemento della
matrice mi trovo
    addi $s7, $s7, 1               # incremento contatore, mi dice a quale colonna della
riga mi trovo
    move $t5, $a2                  # t5 = testa matrice B
    move $t6, $zero               # t6 = contatore per sapere se sono alla colonna giusta
    addi $t4, $s7, -1             # $t4 = numero colonna in cui mi trovo, viene decrementata per il controllo
sottostante
    posizionali_alla_colonna:      #posiziona $t5 in cima alla colonna in cui mi trovo.
        beq $t4, $t6, colonna_trovata    #se contatore colonne t6 == alla colonna in cui
mi trovo t4, ho trovato la colonna per cui moltiplicare
        addi $t6, $t6, 1               #incremento il contatore colonne
        lw $t5, 4($t5)                #t5 = campo elemento successivo di t5
        j posizionali_alla_colonna
    colonna_trovata:
        move $t6, $zero               # t6 = contatore per sapere in che riga mi trovo
        move $t4, $s5                 # t4 = testa matrice A
        posizionali_alla_riga:        #posiziona t4 al primo elemento della riga per cui devo
moltiplicare
        beq $t3, $t6, riga_trovata    #se riga in cui mi trovo $t3 == contatore righe,
ho trovato la riga per cui moltiplicare.
        addi $t6, $t6, 1               #incremento il contatore delle righe
        move $t7, $zero               #conto colonna, appena arriva al numero di colonne
allora sono alla riga dopo
        scorri_riga:
            addi $t7, $t7, 1           #incrementa il numero di colonna
            lw $t4, 4($t4)              #t4 = campo elemento successivo di
matrice A
            beq $t7, $a3, posizionali_alla_riga    #se contatore colonna $t7 ==
numero colonne della matrice $a3, posso andare alla riga dopo
            j scorri_riga              #se non sono arrivato in fondo alla riga,
continuo a ciclare
        riga_trovata:
            move $t8, $zero            #contatore elementi moltiplicati e stampati
            scorri:
                addi $t8, $t8, 1        #incremento il contatore elementi
                lw $t3, 0($t4)          #elemento matrice A, $t3 = A[i]
                lw $t6, 0($t5)          #elemento matrice B, $t6 = B[i]
                mul $t6, $t3, $t6       # t6 = A[i] X B[i]
                add $s3, $s3, $t6       #somma di prodotti, somma gli elementi appena
moltiplicati più quelli precedenti
                beq $a3, $t8, stampa_elemento    #se contatore elementi == grandezza
matrice allora salta a stampare il risultato.
                lw $t4, 4($t4)          #passo all'elemento successivo nella matrice A
                move $t7, $zero         #contatore colonne, per andare alla riga successiva
                riga_succ:              #mette in t5 l'indirizzo della cella sottostante. va a alla
riga dopo.
                    addi $t7, $t7, 1    #incremento contatore colonne.
                    lw $t5, 4($t5)      #t5 = campo elemento successivo di t4
                    beq $t7, $a3, scorri    #se contatore colonne == grandezza
matrice torna sopra a effettuare le operazioni di moltiplicazione e addizione
                    j riga_succ         #altrimenti continua a scorrere la matrice per
andare alla riga sotto.
            stampa_elemento:
                #stampa il risultato ottenuto da righe per colonne.
                move $a0, $s3          # $a0 == risultato da stampare
                li $v0, 1               #stampa intero
                syscall
                li $v0, 4               #stampo spazio per mettere accanto elementi
                la $a0, str_spazio

```



```

                                syscall

                                lw $t0, 4($t0)                # vado all'elemento successivo da moltiplicare.
                                j loop_moltiplica #salto e continuo a moltiplicare
matrici_moltiplicate:
                                lw $s3, 8($sp)
                                lw $s7, 4($sp)
                                lw $t0, 0($sp)
                                addi $sp, $sp, 12
                                jr $t0

stampa_matrice: # a0 = Testa. t0 verra' usato come puntatore per scorrere gli elementi della matrice, $a1 grandezza
matrice
                                move $t0, $a0
nuova_riga:    #stampo un "\n" per andare a capo
                                la $a0, str_a_capo
                                li $v0, 4
                                syscall
                                move $t2, $zero #setto il contatore delle colonne a 0
                                loop_print:
                                    beq $t0, $zero, matrice_stampata # se t0 == 0 si e' raggiunta la fine della matrice e si esce
                                    beq $t2, $a1, nuova_riga #se il contatore t2 == grandezza matrice (n colonne) allora salto a
stampare nuova riga
                                    addi $t2, $t2, 1                #incremento contatore delle colonne stampate
                                    li $v0, 1                    # 1 = syscall per stampare intero
                                    lw $a0, 0($t0)                # a0 = valore del campo intero dell'elemento corrente
                                (puntato da t0)
                                    syscall                        # stampa valore intero dell'elemento corrente
                                    li $v0, 4                    # stampo uno spazio, per separare un elemento dall'altro
                                    la $a0, str_spazio
                                    syscall
                                    lw $t0, 4($t0)                # t0 = valore del campo elemento-successivo
                                dell'elemento corrente (puntato da t0)
                                    j loop_print                    #salto e continuo a stampare la matrice
                                matrice_stampata:
                                    jr $ra                        #salto all'indirizzo contenuto nel registro ra

main:
addi $sp, $sp, -12
sw $ra, 0($sp) #salvo nello stack il valore di $ra, (jal main + 4)
sw $s0, 4($sp) #salvo nello stack il valore di $s0
sw $s5, 8($sp) #salvo nello stack il valore di $s5
sw $s6, 12($sp) #salvo nello stack il valore di $s6

move $s5, $zero #testa matrice A inizializzata a 0, verrà modificata dopo aver inserito la prima matrice
move $s6, $zero #testa matrice b inizializzata a 0, verrà modificata dopo aver inserito la prima matrice

la $a0, str_benvenuto #stampo stringa di benvenuto
jal stampa_stringa
menu:
#stampo il menu
la $a0, str_a_capo
jal stampa_stringa
la $a0, str_menu
jal stampa_stringa

#richiedo una lettera in input
li $v0, 12 #12 legge char
syscall #carattere = $v0
move $t0, $v0 #sposto il mio carattere della scelta in $t0

#confronto il carattere inserito con le varie lettere per poi saltare alla funzione scelta
li $t1, 'a'
beq $t0, $t1, scelta_a

li $t1, 'b'

```

```
beq $t0, $t1, scelta_b
```

```
li $t1, 'c'  
beq $t0, $t1, scelta_c
```

```
li $t1, 'd'  
beq $t0, $t1, scelta_d
```

```
li $t1, 'e'  
beq $t0, $t1, scelta_e
```

```
j menu  
#funzioni scelta A
```

```
scelta_a:
```

```
move $a2, $s5    #a2 = testa matrice A (se già creata in precedenza)  
move $a3, $s6    #a3 = testa matrice B (se già creata in precedenza)  
jal inserisci_matrice  
  
lw $s6, 0($sp)      #ripristino dallo stack la testa della matrice B  
lw $s5, 4($sp)      #ripristino dallo stack la testa della matrice A  
lw $s0, 8($sp)      #ripristino dallo stack la grandezza delle matrici  
addi $sp, $sp, 16
```

```
matrici_inserite:  
    #stampo le matrici  
    la $a0, str_matrice_a  
    jal stampa_stringa  
    move $a0, $s5    #sposto la testa della matrice A in a0  
    move $a1, $s0    #sposto la grandezza della matrice in a1  
    jal stampa_matrice  
    la $a0, str_matrice_b  
    jal stampa_stringa  
    move $a0, $s6    #sposto la testa della matrice B in a0  
    move $a1, $s0    #sposto la grandezza della matrice in a1  
    jal stampa_matrice
```

```
j menu    # salta al menu
```

```
scelta_b:
```

```
#$s5 testa matrice A  
#$s6 testa matrice B  
#$s0 numero colonne
```

```
la $a0, str_a_piu_b    #stampo il messaggio "Risultato di A + B : "  
jal stampa_stringa
```

```
move $a0, $s5    #sposto la testa della matrice A in $a0  
move $a1, $s6    #sposto la testa della matrice B in $a1  
move $a2, $s0    #sposto la grandezza delle matrici in $a2  
jal somma  
j menu
```

```
scelta_c:
```

```
#$s5 testa matrice A  
#$s6 testa matrice B  
#$s0 numero colonne  
la $a0, str_a_meno_b    #stampo il messaggio "risultato A - B : "  
jal stampa_stringa
```

```
move $a0, $s5    #a0 = testa matrice A, verrà modificato  
move $a1, $s6    #t1 = testa matrice B, verrà modificato  
move $a2, $s0    #sposto la grandezza delle matrici in $a2  
jal sottrai
```

j menu

scelta\_d:

```
#s5 testa matrice A
#s6 testa matrice B
#s0 n colonne
la $a0, str_a_per_b      #stampo messaggio "risultato A x B: "
jal stampa_stringa

move $a1, $s5             #a1 testa matrice A
move $a2, $s6             #a2 testa matrice B
move $a3, $s0             #a3 grandezza matrici
jal moltiplica
j menu
```

scelta\_e:

```
lw $s6, 12($sp)   #ripristino dallo stack il valore di s6
lw $s5, 8($sp)    #ripristino dallo stack il valore di s5
lw $s0, 4($sp)    #ripristino dallo stack il valore di s0
lw $ra, 0($sp)    #ripristino dallo stack il valore di $ra, (jal main + 4)
addi $sp, $sp, 12

li $v0, 10        #syscall per uscita da sistema
syscall
```