Elaborato di Assembly

Architettura degli Elaboratori
Università degli Studi di Verona, Anno Accademico 2018-2019

Bonati Niccolò, Bramè Luca, Toffaletti Andrea

September 5, 2019

Contents

1 Introduzione e prerequisiti

Questo software si occupa di essere una fedele traduzione in linguaggio Assembly del programma scritto in linguaggio C `gestioneVettore.c`. Il software è stato scritto per `gas`, architettura `x86/i686` e utilizzando la sintassi AT&T.
Questo software è stato scritto per Linux su architettura `x86 / i686` (non `x86_64/amd64`), e potrebbero essere necessari pacchetti generalmente non preinstallati per compilare il programma ed eseguirne il debug: su Fedora Workstation, la distribuzione Linux che è stata usata per realizzare questo elaborato, è necessario installare `glibc-devel-i686` e `glibc-2.29-12.fc30.i686`, che possono essere installati rispettivamente con i comandi `dnf install glibc-devel-i686` e `dnf debuginfo-install glibc-2.29-15.fc30.i686`.
`gestioneVettore.c` è un programma da riga di comando che acquisisce un vettore di 10 interi da `stdin`, per poi consentire all'utente di analizzare quest'ultimo in diversi modi.
Per questioni di ordine ed organizzazione è stata presa la decisione di ripartire il progetto tra due file:

- `main.s` - file principale, contenente
  - L'equivalente Assembly della funzione `main`
  - Il codice che si occupa di sostituire la funzione `stampaOpzioni` stampando a schermo il menu
  - Il codice che si occupa della parte I/O (vale a dire, acquisizioni da `stdin` e stampe a `stdout`).
- `vectlib.s` - file secondario che contiene tutto il codice che effettivamente analizza il vettore.

Oltre ad aumentare la leggibilità del codice e ridurne il carico cognitivo, questa organizzazione ci permette di separare efficacemente la logica del programma dalle operazioni di I/O: tutte le funzioni contenute in `vectlib.s` sono completamente indipendenti l'una dall'altra e possono essere facilmente riutilizzate all'interno di un altro programma con minori modifiche.

Al fine di semplificare la portabilità del codice sono stati specificati i registri utilizzati per passare un valore ad una funzione dal main, nonché quelli utilizzati dalle funzioni per restituire un valore al main.

Consistente per l'intero programma è inoltre l'utilizzo della `camelCase` per variabili e funzioni, al fine di rendere più naturale il confronto del software con il programma originale `gestioneVettore.c`.

2 Registri

I registri vengono utilizzati in maniera coerente per l'intero programma. Nel limite del possibile, infatti, si è tentato di assegnare a ciascuno dei sei registri

a disposizione una specifica mansione. Tuttavia, commentate sono le specifiche condizioni entro le quali ciascun registro è stato adoperato in ogni specifico contesto. I commenti documentano inoltre eventuali eccezioni alla regola, introdotte per questioni di efficienza o praticità.

- `eax` è usato per restituire valori dalle funzioni al main
- `ebx` è un registro fisso, contenente il vettore da analizzare
- `ecx` e `edx` sono usati intercambiabilmente come input o argomento delle funzioni
- `edi` è un registro fisso, contentente la lunghezza del vettore
- `esi` è utilizzato come contatore

Si noti inoltre che i registri `eax`, `ecx` ed `edx` vengono modificati ogniqualvolta venga chiamata una funzione del linguaggio C con il linker dinamico. Per questa ragione, sono stati utilizzati per memorizzare dati importanti solo tra due chiamate di `printf` e `scanf` (le due funzioni della `stdio.h` del linguaggio C che sono state utilizzate per ragioni di praticità) e sono frequentemente inizializzati per assicurarsi che il loro contenuto sia nullo.

3 Scelte progettuali

Durante la scrittura di questo software la priorità è stata tenere un'elevata fedeltà con il codice originale, scrivere un codice estremamente lineare e leggibile così da facilitarne il riutilizzo.
La prima divergenza sostanziale che è stata apportata rispetto al codice originale è stata stampare il menu come un'unica stringa anziché usare una serie di `printf`: è stata presa questa decisione perché, se anche richiamare dieci volte di fila `printf` in C sia comunque un buon compromesso tra prestazioni e leggibilità, il contrario è vero in Assembly. Per praticità, infatti, è stato utilizzato il linker dinamico per utilizzare le funzioni `printf` e `scanf` del liguaggio C, scelta azzeccata per leggibilità e mantenibilità, ma sfavorevole per le prestazioni.

È facile convincersi del perché usare il linker dinamico diverse volte di fila in Assembly sia inefficiente leggendo il codice di una chiamata arbitraria:

```
pushl %eax
pushl $txtOpt3odd
call printf
addl $8, %esp
```

Per una banale stampa, infatti, è necessario memorizzare diversi valori sullo stack, operare una call ad una funzione di C (che, come è possibile verificare con il debugger `gdb` dando il comando `step` appena prima di entrare in `call`

`printf` costa parecchie istruzioni e cambia il contenuto dei registri `eax`, `ecx` ed `edx`), per poi ripristinare lo stack. Non è un costo basso, quindi si è preferito sacrificare l'aderenza al programma originale in favore delle prestazioni.
Per quanto riguarda l'implementazione delle altre funzioni, si è cercato di riutilizzare codice solo quando pratico e ripetere funzioni anziché parametrizzarle o riutilizzare codice quando questo avrebbe comportato un ingente costo in termini di prestazioni ed avrebbe inficiato la leggibilità del programma eccessivamente, avendo come unico pro la ridotta quantità di righe utilizzate, che oggigiorno non è più la priorità maggiore, vista la disponibilità e la popolarità di dischi veloci e capienti. Ripetere una funzione anziché riutilizzarla migliora drasticamente le prestazioni del software poiché diminuisce il numero di jump, che inficiano le prestazioni della pipeline, e nella maggior parte dei casi garantisce una migliore leggibilità, al piccolo prezzo di avere molto codice ripetuto altrove.
Per esempio, l'equivalente Assembly della funzione `stampaVettore`, che offre la possibilità di stampare il vettore in ordine inverso, è costituita di due sottofunzioni "ridondanti" che stampano il vettore in ordine inverso:

```
stampaVettore:
    test %edx, %edx
    jne stvtInverso
        pushl $txtStvt1
            call printf
        addl $4, %esp

        # for
        xorl %esi, %esi
        stvtFor:
            cmpl %edi, %esi
            jge stvtEnd

            pushl (%ebx,%esi,4)
            incl %esi
            pushl %esi
            pushl $txtStvt3
            call printf
            addl $12, %esp
            decl %esi

            incl %esi
            jmp stvtFor
    stvtInverso:
        pushl $txtStvt2
        call printf
        addl $4, %esp
```

```
    # for
    movl %edi, %esi
    decl %esi
    stvtForInv:
        test %esi, %esi
        jl stvtEnd

        pushl (%ebx,%esi,4)
        incl %esi
        pushl %esi
        pushl $txtStvt3
        call printf
        addl $12, %esp
        decl %esi

        decl %esi
        jmp stvtForInv
stvtEnd:
ret
```

Questa implementazione è un buon compromesso tra forma e prestazioni: al costo di una ridondanza maggiore, si possono ottenere performance migliori grazie alla ridondanza, che consente di ridurre il numero di salti da eseguire. Esiste un controllo in testa alla funzione la cui logica necessita di un unico salto condizionale per sussistere, mentre le due routine della funzione - ovvero, quella che si occupa di stampare il vettore in ordine e quella che si occupa di fare lo stesso in ordine inverso - sono state spezzate in due blocchi di codice diversi. Questo approccio crea codice ridondante, ma consente di evitare svariati salti che dovrebbero essere operati in coda alla funzione per dieci volte consecutive. Sarebbe stato certamente possibile spezzare la funzione in due diverse funzioni, ma ciò avrebbe causato ancora più ridondanza e si sarebbe discostato eccessivamente dall'algoritmo originale per un'ottimizzazione marginale.
Si è scelto di seguire un ragionamento diametralmente opposto per quanto riguarda le funzioni Assembly `cercaValore` e `posizioneMax`. La prima si occupa di cercare un valore passatole dal main all'interno del vettore per poi restituirne la posizione al chiamante: grazie alla consistenza per quanto riguarda l'utilizzo dei registri che è stata adottata nel software, si tratta di una funzione facilmente riutilizzabile all'interno di altre funzioni, come in questo caso. La funzione `posizioneMax` si occupa di stampare il valore massimo presente nel vettore: ad ogni modo, salvando semplicemente il valore trovato nel registro `eax` oltre che a restituire la posizione tramite il registro `esi` consente di richiamare la stessa funzione, la quale viene eseguita in modo regolare, senza alcun parametro o controllo in testa, e restituisce comunque il valore che serve. L'unica istruzione che può risultare ridondante è lo spostamento del valore del vettore in `eax` che viene operato anche se si è scelto di chiamare la funzione per cercarne la posizione, operazione che influisce un minimo sulle spostazioni trat-

tandosi pur sempre di una `movl`, ma in maniera minore rispetto a un salto, ergo, riscrivere un'intera funzione per evitare di eseguire quest'istruzione sarebbe una ottimizzazione banale.

main.s

```
esegui4:
        incl %eax
        cmpl %eax, opzione
        jne esegui5
            pushl $txtOpt4sel
            call printf
            addl $4, %esp

            # Che numero vuoi cercare?
            pushl $toSearch
            pushl $txtFormat
            call scanf
            addl $8, %esp

            xorl %ecx, %ecx
            movl toSearch, %ecx

            call cercaValore


            #If per ricerca inconclusiva
            cercaPrintIf:
                cmpl $11, %esi
                jz cercaPrint404
                jnz cercaPrintSuccess
            cercaPrint404:
                pushl %ecx
                pushl $txtOpt404
                call printf
                addl $8, %esp
                jmp cercaPrintEnd

            cercaPrintSuccess:
                # Stampa success
                pushl %esi                      # posizione effettiva
                pushl toSearch                  # int che si cerca
                pushl $txtOpt4pos               # la posizione è:
                call printf
                addl $12, %esp
            cercaPrintEnd:
```

```
            jmp eseguiEnd


    esegui6:
        incl %eax
        cmpl %eax, opzione
        jne esegui7
            call maxValue

            pushl %edx
            pushl $txtOpt6

            call printf
            addl $8, %esp
        jmp eseguiEnd
```

vectlib.s

```
 # %eax: numero massimo trovato
 # %edx: argomento funzione
 # %ebx: vettore
 # %edi: lunghezza del vettore
 # %esi: contatore array
 maxValue:
     xorl %edx, %edx
     xorl %esi, %esi
     xorl %eax, %eax
     movl (%ebx,%esi,4), %eax

     maxFor:
         cmpl %edi, %esi
         jge maxEnd

         cmpl (%ebx,%esi,4), %eax
         jl maxUpdate

         incl %esi
         jmp maxFor

     #in caso in cui (%ebx,%esi,4) sia maggiore di %eax, il valore verrà spostato il val

     maxUpdate:
         movl (%ebx,%esi,4), %eax        # istruzione "ridondante"
```

```
    incl %esi
    movl %esi, %edx
    jmp maxFor

maxEnd:
    ret
```

4 Variabili

4.1 `main.s`

Variabile	Tipo	Descrizione
`txtInput`	`char[]`	Stampa la prima stringa di testo del `main()`
`txtInputFor`	`char[]`	Stampa il testo del ciclo for nel main
`txtMenu`	`char[]`	Stampa il menu (in una sola variabile per efficienza)
`txtSelect`	`char[]`	Stringa di testo che chiede all'utente di scegliere un valore
`txtOpt0`	`char[]`	Stringa di testo che segnala la terminazione del programma
`txtOpt3even`	`char[]`	Stringa della funzione `cercaPari`
`txtOpt3odd`	`char[]`	Stringa il numero di numeri dispari nel vettore
`txtOpt4sel`	`char[]`	Stringa che chiede all'utente l'intero da cercare
`txtOpt4pos`	`char[]`	Stringa usata per stampare la posizione dell'intero cercato
`txtOpt404`	`char[]`	Stringa che viene stampata nel caso in cui non venga trovato l'intero
`txtOpt5`	`char[]`	Stringa usata per stampare il massimo valore del vettore
`txtOpt6`	`char[]`	Stringa usata per stampare la posizione del massimo valore del vettore
`txtOpt7`	`char[]`	Stringa usata per stampare il minimo valore nel vettore
`txtOpt8`	`char[]`	Stringa usata per stampare la posizione del minio valore nel vettore
`txtOpt9`	`char[]`	Stringa utilizzata per stampare il valore più frequente
`txtOpt10`	`char[]`	Stringa utilizzata per stampare la media dei valori del vettore
`txtOptErr`	`char[]`	Messaggio d'errore
`txtNewLine`	`char`	per andare a capo
`txtFormat`	`char`	`%i`, sintassi C per indicare la presenza di un intero (usata in `printf` e `scanf`)
`vettore`	`long`	Vettore di 10 interi
`LUNGHEZZA_VETTORE`	`long`	Lunghezza del vettore
`opzione`	`long`	Variabile utilizzata per memorizzare l'opzione del menu scelta dall'utente
`toSearch`	`long`	Variabile usata per memorizzare il numero che l'utente vuole cercare nel vettore

Nota: Tutte le variabili di tipo `char[]` sono state dichiarate in Assembly con `.asciz` per efficienza e leggibilità: ciò infatti inserisce automaticamente un terminatore alla fine della stringa.

È stata inoltre stata fatta la scelta di stampare il menu come un'unica stringa, anche se nel codice C originale viene stampato come una successione di `printf`, per ragioni di efficienza.

4.2 `vectlib.s`

Variabile	Tipo	Descrizione
`txtStvt1`	`char[]`	Stringa che usata per stampare i valori che sono stati inseriti nella funzione `stampaVettore`
`txtStvt2`	`char[]`	Stringa usata per stampare il vettore al contrario
`txtStvt3`	`char[]`	Stringa usata per stampare "Valore: "
`lunghezza_vettore`	`word`	Parola in 16 bit utilizzata per contenere la dimensione del vettore (10), da usare come divisore per la funzione che calcola la media intera degli elementi del vettore
`maxFreqStore`	`long`	Variabile utilizzata per memorizzare la frequenza massima registrata. Viene aggiornata ad ogni ciclo

5 Makefile

`make`	`make all`
`make all`	Compila ed esegue, con info di debug
`make norun`	Compila con info di debug senza eseguire
`make nodbnorun`	Compila senza info di debug
`make nodebug`	Compila ed esegue senza info di debug
`make run`	Esegue il binario
`make clean`	Pulisce binari, file oggetto e core dump

6 Pseudocodice

6.1 main.s

```
# Acquisizione vettore
print txtInput
vect = vettore
len = lunghezza_vettore

i = 0
for i in range[1,len]
    print i++
    pos = &eax
    scan vect[i]

# Menu e selezione
print txtmenu

do
    print txtSelect
    scan opzione
    call eseguiopzione
while opzione != 0
if opzione = 0
    return 0
elif opzione != 0
    print txtmenu

eseguiOpzione:
    print \n
    # Riapplico registri fissi
    vect = vettore
    len = lunghezza_vettore

    scelta = -1
    if scelta != opzione
        goto esegui0
    # Se arriva qui c'è qualcosa di sbagliato
        print txtMenu
    break

    esegui0:
        scelta++
        if scelta != opzione
            goto esegui1
        print txtOpt0
```

```
    break

esegui1:
    scelta++
    if scelta != opzione
        goto esegui2
    al_contrario = 0                # OFF
    Stampavettore(al_contrario)
    break

esegui2:
    scelta++
    if scelta != opzione
        goto esegui3
    al_contrario = 1                # ON
    stampaVettore(al_contrario)
    break

esegui3:
    scelta++
    if scelta != opzione
        goto esegui4
        risultato = numeroPari()
    print txtOpt3Even
    risultato -= edi
    print ("txtOpt3odd, %d"(risultato))
    break

esegui4:
    scelta++
    if scelta != opzione
        goto esegui5
    print txtOpt4sel
    int numero_da_cercare
    scan numero_da_cercare
    int risultato
    risultato = cercaValore(numero_da_cercare)
    if risultato = 11
        print cercaPrint404
        break
    else
    print ("txtOpt4pos", i)
        break

esegui5:
    scelta++
```

```
    if scelta != opzione
        goto esegui6
    risultato = maxValue()
    print ("txtOpt5",return)
    break

esegui6:
    scelta++
    if scelta != opzione
        goto esegui7
    risultato = maxValue()
    print("txtOpt6", risultato)
    break

esegui7:
    scelta++
    if scelta != opzione
        goto esegui8
    risultato = minValue()
    print("txtOpt7", risultato)
    break

esegui8:
    scelta++
    if scelta != opzione
        goto esegui9
    minValue()                # vedi funzione
    print edx
    break

esegui9:
    scelta++
    if scelta != opzione
        esegui10
        maxFreq()
        break

esegui10:
    scelta++
    if scelta != opzione
        eseguiError()
    risultato = calcolamediaintera()
    print("txtOpt10, %d", risultato)
    break

eseguiError:
```

```
        print(txtOptErr)
        eseguierror2()

    return 0
```

nota: diversi registri vengono azzerati a inizio funzione. Per semplicità, questa operazione verrà riportata come se fosse un azzeramento di una variabile. Per completezza, ad ogni modo, si noti che questa operazione viene compiuta compiendo uno XOR sul registro. Stando alla tabella delle verità del bitwise XOR, operare uno XOR tra due numeri identici (effettivamente lo stesso numero) restituirà sempre 0. Il risultato dell'operazione viene scritto sul primo registro menzionato, azzerandolo.

```
xorl %eax, %ebx
# xor reg1, reg2 --> reg1
```

6.2 stampaVettore

```
# esi = 1
if (al_contrario == 0){

    if (i == 0)
        return
    i++
    print (vettore[i])
    i--

    i--
    cycle back
}

if ((lunghezza_vettore - 1) >= 0)
    return
i++
print (vettore[i])

i--
cycle back
```

6.3 numeroPari

```
a = 0
i = 0

inizioCicloPari:
```

```
if ((i - lunghezza_vettore) == 0)
    return
if (vettore[i] == 0){
    // positivo

    1 AND vettore[i]
    if vettore[i] > 0 {
        // dispari
        i++
        goto inizioCicloPari
    }
}
vettore[i] = -vettore[i]
```

6.4 cercaValore

```
i = 0
risultato = 0

cercaFor:
if ((i - lunghezza_vettore) >= 0)
    i++
    return
}

if ((valore_da_cercare - vettore[i]) == 0)
    i++
    return

i++
goto cercaFor
```

6.5 maxValue

```
i = 0
risultato = 0
posizione = 0

maxFor:
    if ((i - lunghezza_vettore) >= 0)
        return
    if ((risultato - vettore[i]) <0)
        eax = vettore[i]
        i++
        posizione = i
        goto maxFor
```

6.6 minValue

```
i = i
risultato = vettore[i]

minFor:
    if ((i - lunghezza_vettore) >= 0)
        return

    if ((risultato - lunghezza_vettore) > 0)
        risultato = vettore[i]
        i++
        posizione = i
        goto minFor

    i++
    goto minFor
```

6.7 maxFreq

```
i1 = 0
i2 = 0
freq = 0
maxfreq_temp = 0
maxfreq_persist = -1

for i1 = 0 ... lunghezza_vettore
    for i2 = 0 ... lunghezza_vettore
        if vettore(i1) =! vettore(i2)
            i2++
            cycle back (internal for)
        else
            freq++
            i2++
            cycle back (internal for)

if freq > maxfreq_persist
    risultato = vettore(i1)
    maxfreq_persist = freq
else
    i1++
    freq = 0
    cycle back (external for)
```

6.8 calcolaMediaIntera

```
i = 0
risultato = 0
somma = 0
holder_vettore = =

calcolaMediaInteraFor:
    if ((i - lunghezza_vettore) >= 0)
        goto calcolaMediaInteraEnd

    # per piena correttezza, sarebbe:
        holder_vettore = vettore[i]
        somma +- holder_vettore

    somma += vettore[i]

    i++
    goto calcolaMediaInteraFor

calcolaMediaInteraEnd:
    risultato = somma
    (operand in risultato) * 2
    risultato / lunghezza_vettore -> risultato

    return
```

nota: In questo caso, il registro `eax`, chiamato `risultato` per tutto lo pseudocodice per consistenza, oltre ad essere il registro utilizzato per sostituire il risultato dell'operazione (ossia la media tra tutti gli elementi del vettore) deve contenere il dividendo della divisione per via del funzionamento dell'istruzione `div`.