Progetto finale
Implementazione di una virtual machine

Consegna del progetto e modalità di esame

Il progetto potrà essere realizzato individualmente oppure in gruppi di due persone e consegnato entro il 9 giugno 2019. La discussione dei progetti avverrà nella settimana successiva come da calendario pubblicato su Moodle. **Si raccomanda di seguire gli interventi nei forum per eventuali chiarimenti/modifiche.**

Composizione dei gruppi

V. forum per i dettagli

Modalità di consegna

Il progetto dovrà essere consegnato tramite Moodle includendo in un unico file compresso (formato ZIP) il codice sorgente (*.c,*.h,makefile) ed i file di input per la virtual machine contenenti i casi di test (*.cvm).

Il programma deve poter essere compilato senza errori o warning con le opzioni `-g3 -fsanitize=address -fsanitize=undefined -std=gnu89 -pedantic-errors -Wall -Wextra` ed essere eseguito senza errori. La memoria allocata dinamicamente deve essere rilasciata prima del termine dell'esecuzione.

Macchina virtuale

Lo scopo del progetto è implementare una macchina virtuale con registri e stack che esegua dei programmi specificati in codice macchina (v. sezioni successive) in un'array di interi nel quale ciascuna istruzione occupa un numero variabile di posizioni. La macchina virtuale può produrre un output a console utilizzando delle apposite istruzioni per il debugging.

Stack

La macchina virtuale utilizza un'area di memoria (array di interi) di 64KB come stack (accesso LIFO). Il registro speciale SP (unsigned int) indica la posizione (indice) della prossima posizione dello stack da scrivere (0 stack vuoto). L'istruzione **PUSH** inserisce un intero nello stack (ed incrementa SP), l'istruzione **POP** preleva un elemento dallo stack (dopo aver decrementato SP). In caso di stack overflow (PUSH su stack pieno) o stack underflow (POP da stack vuoto) il programma deve terminare immediatamente segnalando un errore.

Instruction pointer, jump, chiamate a subroutine

La macchina virtuale, dopo il bootstrap durante il quale vengono inizializzate le strutture dati ed i registri, ripetutamente legge le istruzioni del programma in codice macchina (fetch), ne prepara gli operandi e le esegue (execute). Il registro speciale **instruction pointer (IP)** contiene l'indice (unsigned int) della prossima istruzione da eseguire. Dopo il fetch di un'istruzione viene incrementato in modo tale da indicare l'istruzione successiva (dipende dal numero di argomenti dell'istruzione, v. sezione instruction set). L'istruzione **jump (JMP)** sostituisce il valore corrente

di IP con il valore indicato mentre le istruzioni **jump condizionate** eseguono il salto solo se le rispettive condizione sono vere (v. instruction set). La **chiamata a subroutine (CALL)** inserisce nello stack il valore corrente di IP (già aggiornato alla posizione successiva a CALL) e lo sostituisce con il valore indicato. **RET** permette di rientrare da una chiamata a subroutine sostituendo ad IP il valore prelevato dallo stack.

Registri

La macchina virtuale può utilizzare 32 registri interi R0-R31. Ciascun registro può essere utilizzato come operando o contenere il risultato di operazioni. Vedere la sezione instruction set per i dettagli.

Esempio

In questo esempio viene mostrato come implementare una subroutine che calcoli il fattoriale di un numero e la sua invocazione per calcolare il fattoriale del numero 5 e visualizzarlo. Viene mostrata l'implementazione in C, dalla quale passiamo prima all'implementazione in C senza cicli (sostituiti dall'esecuzione condizionata di salti) e poi all'implementazione in linguaggio assembly. Per finire il linguaggio assembly viene trasformato in codice macchina, che in questo progetto consiste in una sequenza di interi che rappresenta le istruzioni a lunghezza variabile ed i loro parametri come da Tabella 1.

```c
/* Implementazione in C */
#include <stdio.h>

int fattoriale (int n){
  int i,risultato = 1;
  for (i=2; i<=n; i++)
     risultato *= i;
  return risultato;
}

int main() {
  int r = fattoriale(5);
  printf("%d", r);
  return 0;
}
```

```c
/* Implementazione in C utilizzando l'istruzione GOTO */
#include <stdio.h>

int fattoriale (int n){
  int i=2,risultato = 1;
inizio:
  if (i>n)
     goto fine;
  risultato *= i;
  i++;
  goto inizio;
fine:
  return risultato;
 }

int main() {
  int r = fattoriale(5);
  printf("%d\n", r);
  return 0;
}
```

```
;//Implementazione assembly
MOV R0 5
CALL fattoriale      ; fattoriale(5)
DISPLAY R30          ; printf(...., r)
HALT

fattoriale:
MOV R1 2    ; i = 2
```

```
MOV R20 1   ; costante 1
MOV R30 1   ; risultato = 1
inizio:
SUB R0 R1
JNEG fine   ; if (n-i<0) goto fine
MUL R30 R1
POP R30     ; risultato = risultato * i
ADD R1 R20 ;
POP R1      ; i = i + 1
JMP inizio ;

fine:
RET         ; return risultato  (in R30)
```

Codice macchina nel formato richiesto per il file di input(v. sezione seguente): solo il numero iniziale deve essere letto dalla macchina virtuale. A partire dal punto e virgola il contenuto della linea deve essere ignorato e considerato un commento. Per migliorare la leggibilità, nel listato seguente i commenti sono stati utilizzati per indicare le posizioni dell'array e le istruzioni assembly corrispondenti.

```
;//Codice macchina
35   ; numero linee (posizione massima)
12   ; [0] MOV R0 5
0    ; [1]
5    ; [2]
20   ; [3] CALL fattoriale
8    ; [4]
1    ; [5] DISPLAY R30
30   ; [6]
0    ; [7] HALT
;fattoriale:
12   ; [8]  MOV R1 2
1    ; [9]
2    ; [10]
12   ; [11]  MOV R20 1
20   ; [12]
1    ; [13]
12   ; [14]  MOV R30 1
30   ; [15]
1    ; [16]
;inizio:
31   ; [17]  SUB R0 R1
0    ; [18]
1    ; [19]
25   ; [20]  JNEG fine
34   ; [21]
32   ; [22]  MUL R30 R1
30   ; [23]
1    ; [24]
11   ; [25]  POP R30
30   ; [26]
30   ; [27]  ADD R1 R20
1    ; [28]
20   ; [29]
11   ; [30]  POP R1
1    ; [31]
22   ; [32]  JMP inizio
17   ; [33]
;fine:
21   ; [34]  RET
```

Utilizzo della virtual machine

La virtual machine deve accettare da linea di comando due parametri: operazione ('esegui' o 'stampa') e nome del file di testo contenente il programma. Ciascuna linea del file di testo contiene un numero intero corrispondente ad un elemento del codice macchina del programma (una posizione dell'array di interi) ed eventuali commenti

preceduto dal carattere ';'. La prima linea contiene il numero di linee non vuote del file (esclusa la prima), uguale alla lunghezza dell'array di interi che conterrà il programma da eseguire.

Nel caso l'operazione richiesta sia 'stampa' la virtual machine deve:

- verificare che il file esista (altrimenti termina con un errore)
- aprire il file e stampare le istruzioni del programma corrispondente secondo il formato:
  `[posizione] istruzione P1 P2`
  dove posizione è la posizione nell'array, istruzione il codice mnemonico, P1 e P2 sono gli eventuali parametri. Nel caso uno del parametri sia un registro stampare la rappresentazione mnemonica corrispondente (es: R3 anziché 3)

Per il codice macchina dell'esempio, l'operazione 'stampa' dovrà produrre l'output seguente:

```
[  0] MOV R0 5
[  3] CALL 8
[  5] DISPLAY R30
[  7] HALT
[  8] MOV R1 2
[ 11] MOV R20 1
[ 14] MOV R30 1
[ 17] SUB R0 R1
[ 20] JNEG 34
[ 22] MUL R30 R1
[ 25] POP R30
[ 27] ADD R1 R20
[ 30] POP R1
[ 32] JMP 17
[ 34] RET
```

Nel caso l'operazione richiesta sia 'esegui' VM deve:

- verificare che il file esista (altrimenti termina con un errore)
- aprire il file, allocare blocco di memoria di dimensione adeguata (array di interi, dimensione dalla prima linea del file) e leggere le istruzioni dal file scrivendo gli interi corrispondenti in memoria
- inizializzare i registri e lo stack
- inizializzare IP a zero
- ripetere fetch-execute fino a che l'istruzione è diversa da HALT

Programmi dimostrativi

Realizzare alcuni programmi dimostrativi opportunamente documentati, che verifichino il funzionamento dell'intero instruction set. **Non è richiesto che i programmi dimostrativi siano originali, più gruppi possono utilizzare liberamente gli stessi esempi.** Alcune possibili idee:

- determinare se un numero è primo (subroutine e codice che la chiama per alcuni numeri stampando i risultati)
- somma dei primi N numeri interi (subroutine e codice che la chiama per alcuni numeri stampando i risultati)
- somma dei primi N quadrati di numeri interi (subroutine e codice che la chiama per alcuni numeri stampando i risultati)

Plagio

Si ricorda che in caso di plagio, in aggiunta ad eventuali altri provvedimenti, saranno considerati nulli tutti i progetti coinvolti.

mnemonico	lunghezza (numero di interi)	codice macchina	P1 (param 1)	P2 (param 2)	descrizione
HALT	1	0	-	-	Termina il programma
DISPLAY	2	1	0-31 (reg R0-R31)	-	stampa su console il valore del registro indicato
PRINT_STACK	2	2	numero	-	stampa su console il numero indicato di posizioni dello stack. Stampare le posizioni in ordine inverso a partire dalla posizione SP-1 fino a SP-N inclusa. Stampare l'indice della posizione ed il valore in essa contenuto
PUSH	2	10	0-31 (reg R0-R31)	-	Inserisce il contenuto del registro indicato nello stack (in posizione SP) ed incrementa SP
POP	2	11	0-31 (reg R0-R31)	-	Decrementa SP e copia il valore in posizione SP (dopo il decremento) nel registro indicato
MOV	2	12	0-31 (reg R0-R31)	numero	Copia il valore P2 nel registro indicato
CALL	2	20	posizione	-	Chiamata a subroutine. PUSH IP (posizione successiva a CALL) e JMP alla posizione indicata
RET	1	21	-	-	Ritorno da chiamata a subroutine. POP in IP.
JMP	2	22	posizione	-	Sostituisce il valore di IP con il valore indicato
JZ	2	23	posizione	-	Sostituisce il valore di IP con il valore indicato se l'ultimo elemento inserito nello stack è diverso da zero e lo rimuove, decrementando SP.
JPOS	2	24	posizione	-	Sostituisce il valore di IP con il valore indicato se l'ultimo elemento inserito nello stack è maggiore di zero e lo rimuove, decrementando SP.
JNEG	2	25	posizione	-	Sostituisce il valore di IP con il valore indicato se l'ultimo elemento inserito nello stack è minore di zero e lo rimuove, decrementando SP.
ADD	3	30	0-31 (reg R0-R31)	0-31 (reg R0-R31)	Addizione intera P1 + P2. Il risultato viene inserito nello stack. Terminazione in caso di overflow.
SUB	3	31	0-31 (reg R0-R31)	0-31 (reg R0-R31)	Sottrazione intera P1 - P2. Il risultato viene inserito nello stack. Terminazione in caso di overflow.
MUL	3	32	0-31 (reg R0-R31)	0-31 (reg R0-R31)	Moltiplicazione intera P1 * P2. Il risultato viene inserito nello stack. Terminazione in caso di overflow.
DIV	3	33	0-31 (reg R0-R31)	0-31 (reg R0-R31)	Divisione intera P1 / P2. Il risultato viene inserito nello stack. Terminazione con errore in caso di divisione per zero.

Tabella 1: Instruction set e codice macchina