mgr inż. Tomasz Jaworski
Instytut Informatyki Stosowanej PŁ

Zadanie

Zaprojektuj i zaimplementuj maszynę wirtualną/emulator dla podstawie podanej listy rozkazów oraz jej specyfikacji.

Wstęp

(z Wikipedii http://pl.wikipedia.org/wiki/Emulator)

Emulator – program komputerowy, który duplikuje funkcje jednego systemu informatycznego w innym, dzięki czemu ów drugi system zwraca te same rezultaty, co pierwszy. Mówimy wtedy, że pierwszy system jest emulowany przez drugi.

Jedną z form emulatorów są programy, które umożliwiają uruchamianie aplikacji na komputerze lub systemie operacyjnym innym niż ten, na który zostały napisane np. uruchomienie emulatora Amigi w systemie operacyjnym Windows, czy emulatora konsoli do gier Nintendo w systemie Linux. Emulatory są tak programowane, aby jak najdokładniej potrafiły **'udawać'** emulowaną maszynę z lepszym lub gorszym efektem (są np. trudności z poprawnym oddaniem palety barw, czy dźwięków oraz z obsługą urządzeń zewnętrznych takich jak np. pady, jednak wiele emulatorów posiada praktycznie 100% zgodność z oryginalną maszyną).

Programy te mogą też emulować poszczególne funkcje oprogramowania i sprzętu standardowo niedostępne w ramach danego systemu, np. karty dźwiękowe.

Zadaniem projektowanego emulatora/maszyny wirtualnej jest poprawne wykonanie programów w postaci binarnej, załączonych do niniejszej instrukcji, jako tablice liczb całkowitych *int P[]*.

Architektura

Dla programu użytkownika dostępne są trzy rejestry numeryczne, *R1..R3* mogące przechowywać zarówno liczby zmiennoprzecinkowe jak i całkowite. Dostępna jest również pamięć danych *D[],* przechowująca dane wejściowe dla programu użytkownika.

Wykonywanie programu *P[]* odbywa się poprzez przesuwanie po tablicy P zmiennej indeksującej, zwanej *licznikiem programu* (ang. *program counter*), np. *int n*. Wykonywanie rozpoczyna się od pierwszego rozkazu (*n=0*). Aktualnie wykonywany rozkaz to *P[n]*. W zależności od kodu rozkazu (patrz **Lista rozkazów**), wykonywana jest pewna operacja, np. P[n] == 7, co oznacza rozkaz *PRINT Rk*. Parametry rozkazów umieszczone są zaraz po jego kodzie, np. dla rozkazu *PRINT*, parametr *k=P[n+1]*. Po wykonaniu rozkazu należy przejść do kolejnego rozkazu, czyli zwiększyć *n* o długość już wykonanego rozkazu. Długości rozkazów zamieszczono w tabeli **Lista rozkazów**. W przypadku rozkazu skoków, licznik programu ustawiany jest na nową wartość, bez konieczności jego zwiększania o długość rozkazu skoku.

Lista rozkazów:

Rozkaz	Kod rozkazu + parametry	Długość rozkazu	Opis
Rk <= V	**1**,k,V	3	Przypisz wartość *V* do rejestru *Rk*
Rk += V	**4**,k,V	3	Zwiększ wartość rejestru *Rk* o wartość *V*
Rk +=D [Rp]	**3**,k,p	3	Dodaj do rejestru *Rk* wartość komórki *Rp* z pamięci danych D[]. *Rp* oznacza wartość rejestru *Rp*.
JE Rk, Rl, a	**2**,k,l,a	4	Skok warunkowy: Jeśli *Rk* jest równe *Rl* to skocz pod adres *a*; w przeciwnym razie kontynuuj wykonywanie programu użytkownika – idź do następnego rozkazu
JMP a	**5**,a	2	Skocz bezwarunkowy do adresu *a*.
Rk /= Rl	**6**,k,l	3	Podziel wartość rejestru *Rk* przez wartość rejestru *Rl* a wynik zapisz do rejestru *Rk*.
PRINT Rk	**7**,k	2	Wyświetl wartość rejestru *Rk* na ekranie.
END	**8**	1	Przerwij wykonywanie programu użytkownika.

Przykładowy program

Poniżej przedstawiono przykładowy program, napisany z wykorzystaniem przedstawionej listy rozkazów:

```
          R2 <= 0
          R1 <= 10

PETLA:    R2 += 1
          PRINT R2
          JE    R2, R1, KONIEC
          JMP   PETLA
KONIEC:   END
```

Zadaniem programu jest wyświetlenie wartości od 1 do 10 na ekranie.
W pierwszej kolejności, rejestry *R1* i *R2* inicjowane są wartościami odpowiednio 10 i 0. W zamierzeniu autora tego programu, *R2* przechowuje wartość startową (-1) a *R1* wartość końcową.
Rozkaz **R2+=1** odpowiada za zwiększenie wartości rejestru *R2* o 1. Jest on również opatrzony etykietą **PETLA**. Jest to słowne oznaczenie miejsca w programie. Etykiecie odpowiada adres rozkazu (tutaj wartość **6**, wyjaśnione w listingu dalej)**,** jednak forma tekstowa jest bardziej czytelna od numerycznej wartości adresu.
Rozkaz **PRINT R2** odpowiada za wyświetlenie wartości rejestru *R2* na ekranie.
Skok **JE** (ang. *Jump if Equal*) w **JE R2,R1,KONIEC** sprawdza, czy wartość rejestru *R2* jest równa wartości *R1*. Jeśli tak, to wykonywany jest skok do adresu etykiety **KONIEC** (zdefiniowanej dalej w programie). Jeśli wartość *R1* nie równa się wartości *R2*, to emulator przechodzi do wykonywania kolejnego rozkazu, którym jest skok bezwarunkowy **JMP** (ang. *jump*). W tym wypadku skok wykonywany jest pod adres etykiety **PETLA**.
Rozkaz **END** kończy wykonywanie programu w emulatorze.

Postać binarna powyższego programu jest następująca:
int P[] = {**1**,2,0, **1**,1,10, **4**,2,1, **7**,2, **2**,1,2,17, **5**,6, **8**};

Łącząc wartości pól tablicy *P[]* z listingiem programu można zauważyć, że poszczególne wartości *P* bazują na liście rozkazów:

Adres	*Etykieta*	*Rozkaz*	*Postać binarna*	*Uwagi*
0		R2 <= 0	**1**,2,0	k=2;V=0
3		R1 <= 10	**1**,1,10	k=1;V=10
6	PETLA:	R2 += 1	**4**,2,1	k=2;V=1
9		PRINT R2	**7**,2	k=2
1		JE R2, R1, KONIEC	**2**,1,2,17	k=1;l=2;a=17
15		JMP PETLA	**5**,6	a=6
17	KONIEC:	END	**8**	

Kod rozkazu (zgodny z listą rozkazów) został zaznaczony pogrubioną czcionką. Tłem oznaczono adresy skoków i adresy odpowiadających im etykiet.

Programy

Zadaniem programu użytkownika jest wyznaczenie średniej z kilku wartości, znajdujących się w pamięci danych *D[]*. Program użytkownika zakłada, że w pamięci danych *D[]* znajdują się 4 wartości numeryczne. Poniżej podano trzy różne wersje tego samego programu.

Wynik ich działania jest identyczny.

Program 1

Kod binarny:
int P[]={1,1,0, 1,2,0, 1,3,4, 2,2,3,21, 3,1,2, 4,2,1, 5,9, 6,1,3, 7,1, 7,2, 8};

Listing:

```
          R1 <= 0
          R2 <= 0
          R3 <= 4
PETLA:    JE R2, R3, WYNIKI
          R1 += D[R2]
          R2 += 1
          JMP PETLA
WYNIKI:   R1 /= R3
          PRINT R1
          PRINT R2
          END
```

Program 2

Kod binarny:

```
int P[] = {5,29, 1,2,0, 5,34, 2,2,3,19, 3,1,2, 4,2,1, 5,7, 6,1,3, 7,1, 7,2, 8, 8, 8,
           1,1,0, 5,2, 1,3,4, 5,7};
```

Listing:

```
        JMP     A
B:      R2 <=   0
        JMP     C
PETLA:  JE      R2, R3, WYNIK
        R1 += D[R2]
        R2 += 1
        JMP     PETLA
WYNIK:  R1 /= R3
        PRINT   R1
        PRINT   R2
        END
        END
        END

A:      R1 <=   0
        JMP     B
C:      R3 <=   4
        JMP     PETLA
```

Program 3

Kod binarny:

```
int P[] = {5,29, 1,2,0, 5,34, 2,2,3,19, 3,1,2, 4,2,1, 5,7, 6,1,3, 7,1, 5,39, 8, 8, 8,
           1,1,0, 5,2, 1,3,4, 5,7, 7,2, 5,26};
```

Listing:

```
        JMP     A
B:      R2 <=   0
        JMP     C
PETLA:  JE      R2, R3, WYNIK
        R1 += D[R2]
        R2 += 1
        JMP     PETLA
WYNIK:  R1 /= R3

        PRINT   R1
        JMP     WYN2
KONIEC: END
        END
        END

A:      R1 <=   0
        JMP     B
C:      R3 <=   4
```

```
        JMP     PETLA
WYN2:   PRINT   R2
        JMP     KONIEC
```

Przykładowa implementacja rozkazu

Poniżej przedstawiono przykładową implementację rozkazu **Rk += D[Rp]**.

```
if (P[n] == 3) // kod rozkazu Rk += D[Rp] - z listy rozkazów
{
      // pobierz numer k rejestru
      int k = P[n + 1];

      // pobierz wartość p rejestru
      int p = P[n + 2];

      // pobierz wartosc rejestru Rp
      int rp = R[p];

      // pobierz wartosc komórki D[] - o jaką
      // rejestr Rk ma być zwiększony
      float V = D[rp];

      // zwiększ rejestr Rk o wartosc V, czyli o D[rp]
      R[k] = R[k] + V;

      // przejdź do kolejnego rozkazu
      n = n + 3; // długość rozkazu znajduje się
                 // w liście rozkazów
}
```