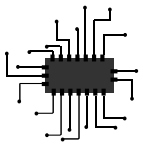
Úloha č. 2
Cizokrajný přikazovací úřad



*Tato úloha je vyhodnocována automaticky. Je potřeba, aby výstup programu **přesně** korespondoval se specifikací výstupu níže. Jak odevzdávat tento typ úloh se můžeš dočíst na webových stránkách FIKSu pod záložkou „Jak řešit FIKS“.*

V prvním kole jsi při shánění různých povolenek a vyplňování všelijakých formulářů ne vždy plně porozuměl otázkám a políčkům, které bylo třeba vyplnit. Celou dobu jsi tajně doufal, že to prošlo bez problémů a že jsi vše vyplnil správně. Minulý týden jsi ale obdržel dopis, ze kterého tě polilo horko. Zřejmě jsi při papírování nechtěně přislíbil pomoc v rozvoji progresivních vlasových přípravků v rámci projektu, který má pod palcem jakási zlomyslná farmakologická společnost. Podle ní se tvoje ZOO nyní účastní krutého programu pro testování na zvířatech. To jsi samozřejmě nikdy nechtěl, ale nedalo se nic dělat – bez toho nešly získat jiné formuláře. Odvolací doba už také uplynula a nejde se z této povinnosti vyvléci (ukončovací podmínky jsou ještě brutálnější).

Spolu s dopisem jsi obdržel i podivný přístroj. Jednou měsíčně přijde do tvojí ZOO soupis příkazů z cizokrajného přikazovacího úřadu. Jejich výsledky je možné velmi rychle získat pomocí tohoto přístroje a vybraných zvířat. Naštěstí se ti po několika dnech podařilo získat z přístroje dokumentaci jednotlivých příkazů. Svitla naděje, že bys mohl z nechutných experimentů odstranit zvířecí složku, a tak byrokracii uspokojit bez trápení zvířat.

Tvým úkolem v této úloze je simulovat procesor s danou instrukční sadou. Čím více druhů instrukcí zvládneš vyhodnotit, tím více zvířat zachráníš. Tak hurá do práce, cizokrajný úřad bude již brzy očekávat výsledky.

Instrukční sada CPÚ

Protože původní záměr instrukční sady počítal s testováním mnoha zvířat najednou, humánní implementace simuluje několik procesů běžících ve sdíleném prostředí.

Každý proces má 8-bitovou hodnotu `PC`, což je index do hlavní paměti. `PC` určuje, kterou instrukci má proces vykonat jako další, a slouží tak jako jeho pozice v paměti. Skok na adresu znamená změnu hodnoty `PC` na tuto novou adresu. `PC` se po vykonání každé instrukce zvětší o jedna. Toto platí i pro všechny skokové instrukce – například nepodmíněný skok na adresu 5 ve skutečnosti skočí na adresu 6 kvůli inkrementaci `PC`. Změny hodnoty `PC` se provádí modulo 256.

Dále má každý proces svůj vlastní zásobník (LIFO datová struktura) mimo hlavní paměť, k němuž nemůžou žádné jiné procesy přistupovat. Zásobník obsahuje až 16 32-bitových čísel. Při pokusu přidat další dojde k přetečení zásobníku a ukončení procesu.

Hlavní paměť je mezi procesy sdílená a nerozlišuje mezi prostorem pro data a prostorem pro instrukce. Fyzicky má 256 buněk o velikosti 32 bitů. Buňky jsou číslovány od 0 do 255, ale hlavní paměť je cyklická, takže adresa 256 je rovna adrese 0, 257 je rovna adrese 1, atp. Na adrese 0 najdeme vždy hodnotu 0, jakýkoliv zápis dat na tuto adresu jakoby neproběhl. Díky cyklickému adresnímu prostoru platí stejné podmínky pro adresy 256, 512, 768, ... Jedinou výjimkou v cyklickém adresním prostoru je adresa 666, jejíž fyzická implementace je zřejmě poškozená – pokus o čtení či zápis na této adrese končí s chybou, nikoliv ale na adresách $k \cdot 256 + 666, k \in \mathbb{Z} \setminus \{0\}$ (např. adresa 410 tímto poškozením netrpí).

Kódování instrukcí

Paměťové buňky mají každá 32 bitů, do kterých se zakóduje osmibitový operační kód instrukce (v přehledu níže) a 16-bitový argument instrukce. Operační kód zabírá nejnižších 8 bitů, prostředních 16 zabírá argument (ignorovaný u instrukcí, které argument nepotřebují) a nejvyšších 8 bitů je při interpretaci instrukcí ignorováno vždy.

```
0x00 NOP          0x0b DIV
0x01 PC           0x0c POW
0x02 PUSH         0x0d BRZ
0x03 POP          0x0e BR3
0x04 SWAP         0x0f BR7
0x05 DUP          0x10 BRGE
0x06 PUSHSSZ      0x11 JMP
0x07 LOAD         0x12 ARMED_BOMB
0x08 STORE        0x13 BOMB
0x09 ADD          0x14 TLPORT
0x0a SUB          0x15 JNTAR
```

Níže je vidět binární reprezentace různých instrukcí. Instrukce `nop` nic nedělá a svůj argument ignoruje, ale díky operačnímu kódu 0 je na jejím zápisu dobře vidět kam se ukládá argument instrukce.

```
00000000000000000000000000000000  NOP   0
00000000000000000000000100000000  NOP   1
00000000000000001111111100000000  NOP   255
00000000111111111111111100000000  NOP   65535
```

Nejnižší bit operačního kódu se shoduje s nejnižším bitem celé reprezentace.

```
00000000000000001111111100000001  PC    255
```

Pro ukázku ještě zápis instrukce `push` s různými argumenty.

```
00000000000000000000000000000010  PUSH  0
00000000000000000000000100000010  PUSH  1
00000000000000001111111100000010  PUSH  255
00000000111111111111111100000010  PUSH  65535
```

Průběh simulace procesů

Simulace běhu procesů probíhá v krocích zvaných cykly. V každém cyklu se každý z dosud žijících procesů probudí a pokusí se interpretovat instrukci, na kterou ukazuje jeho `PC`. Interpretace může selhat (dělení nulou, pokus o interpretaci paměťové buňky, která žádné instrukci neodpovídá, pokus o sejmutí hodnoty z prázdného zásobníku, ...), v takovém případě proces umírá a jeho simulace tím končí. Mrtvý proces se k životu vrátit nemůže.

V každém cyklu se tak každý živý proces probudí právě jednou aby se pokusil o nějakou akci. Do dalšího cyklu postupuje nanejvýš stejný počet procesů (některé mohly zemřít). V rámci jednoho cyklu jsou ovšem akce procesů taktéž uspořádány: na řadu přichází v pořadí, v jakém jsou zapsány na vstupu (viz sekce se specifikací vstupu). Tak je zajištěno, že průběh simulace je určen jednoznačně v případech, kdy závisí na pořadí akcí v rámci jednoho cyklu, například když více procesů přistupuje k jediné paměťové buňce.

Nezařazené instrukce

- `nop` – žádná operace
- `pc` – přidá na vrchol zásobníku aktuální PC

Operace se zásobníkem

Všechny instrukce pracující se zásobníkem (např. i skokové a aritmetické nebo instrukce `pc` výše) dodržují maximální a minimální velikost zásobníku. Pokus o odebrání čísla z prázdného zásobníku a pokus o přidání do plného zásobníku ukončí proces, který se o takovou akci pokusil.

- `push <immediate>` – přidá konstantu na vrchol zásobníku
- `pop` – odebere vrchní hodnotu ze zásobníku
- `swap` – prohodí dvě nejvrchnější hodnoty na zásobníku
- `dup` – duplikuje vrchní hodnotu na zásobníku
- `pushssize` – přidá na vrchol zásobníku aktuální velikost (počet prvků) zásobníku

Instrukce pro přístup k hlavní paměti

- `load`
  - odebere jednu hodnotu ze zásobníku a použije ji jako index do paměti
  - hodnotu načtenou z paměti přidá na vrchol zásobníku
- `store`
  - odebere ze zásobníku dvě hodnoty (nejdříve adresu na vrcholu zásobníku a pak hodnotu k zápisu)
  - adresu použije jako index do paměti, kam zapíše hodnotu k zápisu

Aritmetické operace

Instrukční sada pracuje pouze s nezápornými celými čísly. Během aritmetické operace může dojít k přetečení či podtečení, taková situace není chybou a výsledkem je zbytek po dělení v příslušném modulu (2^{32}).

- `add`
  - odebere dvě hodnoty z vrcholu zásobníku
  - jejich součet poté přidá na vrchol zásobníku
- `sub`
  - odebere dvě hodnoty z vrcholu zásobníku
  - jejich rozdíl (první − druhá) poté přidá na vrchol zásobníku
- `div`
  - odebere dvě hodnoty z vrcholu zásobníku
  - jejich podíl (první/druhá) poté přidá na vrchol zásobníku
  - dělení nulou způsobuje ukončení procesu
- `pow`
  - odebere dvě hodnoty z vrcholu zásobníku
  - na vrchol zásobníku přidá první hodnotu umocněnou na druhou hodnotu
  - pro účely této operace definujeme $0^0 = 1$

Skokové instrukce

- `brz <immediate>`
  - odebere jednu hodnotu z vrcholu zásobníku
  - pokud je hodnota rovna 0, skočí na adresu `PC` + `immediate`
- `br3 <immediate>`

    - odebere jednu hodnotu z vrcholu zásobníku
    - pokud je hodnota rovna 3, skočí na adresu `PC` + `immediate`

- `br7 <immediate>`

    - odebere jednu hodnotu z vrcholu zásobníku
    - pokud je hodnota rovna 7, skočí na adresu `PC` + `immediate`

- `brge <immediate>`

    - odebere dvě hodnoty z vrcholu zásobníku
    - pokud je první hodnota vetší nebo rovna druhé, skočí na adresu `PC` + `immediate`

- `jmp <immediate>` – nepodmíněný skok na adresu `immediate`

Speciální instrukce

- `bomb` – funguje jako nášlapná mina, kterou je nejprve potřeba odjistit. Interpretace této instrukce na adrese a zapíše na adresu a hodnotu `0x12` (tj. `armed_bomb` s nulovým argumentem).
- `armed_bomb` – ukončí proces
- `teleport`

    - počkej dokud další proces (nebo procesy) nedorazí na další instrukci tohoto typu (kdekoliv v paměti)
    - `PC` se po této instrukci neinkrementuje
    - když na konci cyklu (tedy až potom, co každý proces načte a zpracuje instrukci) alespoň dva procesy čekají na této instrukci, hodnoty jejich `PC` se prohodí. Poté se `PC` těchto procesů inkrementují, tím se posunou z adres, na které instrukce `teleport` potkali, a vliv těchto instrukcí tak končí
    - v opačném případě (proces s `PC` ukazující na `teleport` je jen jeden) postupuje proces s `PC` na stále stejné adrese do dalšího cyklu
    - je-li `teleport` instrukce, na které proces čeká, během čekání přepsána, proces dále čeká na ostatní kandidáty k teleportaci (změnu jakoby neviděl)
    - procesy, které už skončily s chybou, se teleportace neúčastní (z pohledu teleportační sémantiky jakoby neexistují).
    - pokud jsou teleportující se procesy alespoň tři, operace prohození je zobecněna na cyklické posunutí. Pořadí procesů určuje jejich zápis na vstupu a posouvají se doleva. Jinými slovy, jsou-li zadány tři procesy na vstupu a je-li v nějakém cyklu první zadaný proces na adrese 1, druhý na adrese 8 a třetí na adrese 5, na každé z těchto adres je instrukce `teleport` a všechny tři procesy jsou naživu, postupují do dalšího cyklu s následujícími hodnotami `PC`: první proces 9, druhý 6 a třetí 2.

- `jantar`

    - ve vzdálenosti $2^i, i \in \{1, 2, 3\}$ v obou směrech od `PC` uloží do hlavní paměti číslo odpovídající instrukci `bomb` s nulovým argumentem

Vstup

Na prvním řádku je číslo $0 < Q \leq 10^3$ udávající počet zadání, která následují. Na prvním řádku každého zadání je číslo $0 < P \leq 10^3$ udávající počet procesů v tomto zadání. Následuje P řádků, na i-tém řádku jsou čísla $0 \leq \mathit{offset}_i < 2^8$ a $0 \leq n_i < 2^8$ a za nimi n_i čísel $0 \leq x_j < 2^{32}$.

Každé zadání je vyhodnocováno zvlášť a odpovídá jedné simulaci. Nejprve je paměť inicializována samými nulami. Následně je do ní načten kód každého programu (sekvence n_i čísel) počínaje adresou offset_i. Programy jsou zapisovány v pořadí, v jakém byly specifikovány na

vstupu. Po zápisu programů do paměti je vytvořeno P procesů, každý se svým `PC` = $offset_i$. Pokud platí $offset_a = offset_b$ pro nějaké $a \neq b$, vytvoří se dva procesy (či více procesů, je-li takových více) se shodným `PC`. Po vytvoření procesů je zadání simulováno po 5000 cyklů.

Výstup

Výstupem je Q řádků, jeden za každé zadání. Na každém řádku jsou čísla $0 \leq a < 2^{32}$ a $0 \leq b < 2^{32}$, kde a značí obsah paměti na adrese 42 na konci simulace (tedy 43. paměťové buňky) a b značí součet hodnot čítačů `PC` všech (i mrtvých) procesů na konci simulace. Pro mrtvé procesy se jako `PC` počítá hodnota čítače bezprostředně před ukončením procesu (tedy adresa instrukce u které došlo během interpretace k chybě).

Bodové ohodnocení se odvíjí od složitosti instrukcí ve vstupních programech.

- 1 bod za vyřešení všech zadání, ve kterých programy pracují jen se zásobníkem a pamětí, tedy instrukce `0x00` až `0x08` (včetně)
- 3 body za vyřešení všech zadání, ve kterých programy používají zásobník, paměť a aritmetiku, instrukce `0x00` až `0x0c` (včetně)
- 6 bodů za vyřešení všech zadání, ve kterých programy používají navíc i skokové instrukce, tzn. `0x00` až `0x11` (včetně)
- 8 bodů za zvládnutí celé instrukční sady (bomby, teleportace)
- 10 bodů za zvládnutí celé instrukční sady a mnoha programů zároveň

Ukázkové vstupy

Vstup

```
1
1
0 7 14225673 258 514 6958345 6351877 10754 8
```

Výstup

```
3 136
```

Program výše odpovídá pseudokódu

```
ADD
PUSH 1
PUSH 2
ADD
DUP
PUSH 42
STORE
```

První instrukce se ale snaží přepsat adresu 0, což není povoleno. Při spuštění programu je na adrese 0 stále 0, odpovídající instrukci `NOP`.

Vstup

```
1
1
1 1 42
```

Výstup

```
0 1
```

V této ukázce skončí proces hned v prvním cyklu, protože se pokusí dekódovat neexistující instrukci.

Vstup

```
2
1
1 47 0 1282 2 4 258 4 5 1549 5 258 4 10 2 63501 3 4 5
6413 4 5 258 8 4 514 8 514 7 258 4 10 61453 258 7 9 514
7 258 4 10 514 8 2 60941 3 5 10754 8
3
1 4 258 10754 8 9
5 4 770 10754 8 9
10 6 514 10754 7 270 9 10
```

Výstup

```
120 1
3 27
```