Procesor

CPSPC 2004 Praha, Den 4.

Submit:	`processor*.out`	Časový limit:	4 hodiny
Vstup:	`processor*.in`	Paměťový limit:	váš vlastní
Výstup:	`processor*.out`	Bodů:	100

Jednou z vedoucích společností v oboru technologických inovací [...][1] je firma proslaveného vynálezce prof. I. N. Tella[2]. Právě vyvinula prototyp nového procesoru nazvaného Hexadecadium. Kódování instrukcí používané tímto procesorem je však zatím poněkud marnotratné[16] a jelikož jak každý ví, 640 KB musí stačit každému[640], potřebují ještě nalézt nějaký způsob, jak programy optimalizovat a zkrátit je.

Profesor Tell zamýšlí stvořit dokonalý optimalizátor – to je program, který si přečte zdrojový text nějakého programu P a vygeneruje nejkratší (co do počtu instrukcí ve zdrojovém textu) možný program Q počítající totéž co P (tzn. který pro každý možný vstup vydá tentýž výstup jako P). Jedinou výjimkou mohou být vstupy, pro které se P nezastaví – v takových případech může Q vydat libovolný výstup nebo se také nezastavit.

Tato úloha se nicméně ukázala být zákeřně těžkou[†] a ačkoliv prof. Tell stále pevně věří, že se mu ji jednou povede vyřešit, chtěl by také své procesory začít co nejdříve prodávat. A tak se rozhodl, že problém zatím poněkud zjednoduší tím, že bude brát v úvahu pouze programy, jejichž vstupem i výstupem je jen jediné 32-bitové číslo, a že místo optimalizačního programu bude k optimalizování najímat levné pracovní síly z Východu. Rozesílá tedy programy svým zaměstnancům a oni produkují nejkratší ekvivalentní programy, jaké umí. Jejich odměna samozřejmě závisí na délce výsledných programů. Správně jste zvětřili příležitost trochu si při svém pobytu v Praze přivydělat.

Popis procesoru

Hexadecadium má 26 registrů nazvaných `a` až `z`. Každý z těchto registrů obsahuje jedno 32-bitové celé číslo bez znaménka.

Procesor je řízen programem, což je posloupnost až 1 000 instrukcí. Ty vypadají takto:

[*návěští*:] [`if` *hodnota relace hodnota*] *příkaz*

[části v hranatých závorkách jsou nepovinné]. Přitom *návěští* nijak neovlivňuje význam instrukce, pouze umožňuje se na instrukci odkazovat při skoku nebo indexaci. Část `if` dává možnost provádění instrukce podmínit (viz níže). *Příkaz*y existují následující:

- *registr* `=` *hodnota* – uvedená *hodnota* je zapsána do zadaného *registr*u, přičemž *hodnota* je buďto:
  - *číslo* – desítkové číslo (0...4294967295), nebo
  - `#`*číslo* – číslo v šestnáctkové soustavě (0...`ffffffff`), či
  - *registr* – hodnota daného registru, popřípadě
  - *návěští*[*hodnota*] – obsah instrukce `data` umístěné *hodnota* instrukcí za *návěští*m (nulová *hodnota* odpovídá instrukci, u které je *návěští*, hodnota 1 té po ni následující atd.). Pokud taková instrukce neexistuje (buďto na příslušném místě je jiná instrukce než `data` nebo se toto místo nachází za koncem programu), je výsledkem nula. Navíc *hodnota* uvnitř hranatých závorek nesmí obsahovat žádné další hranaté závorky.
- *registr* `=` *hodnota operátor hodnota* – provede jednu operaci s danými dvěma hodnotami a zapíše její výsledek do *registr*u (pokud se jeden registr vyskytne na obou stranách rovnítka, nová hodnota je zapsána až po vyhodnocení celé pravé strany). *Operátor*y jsou tu následující:
  - `+`, `-`, `*` – obvyklé aritmetické operátory; pokud přetečou, výsledek se vezme modulo 2^{32},
  - `/`, `%` – standardní celočíselné dělení a zbytek po něm; dělíme-li x nulou, vyjde podíl 0 a zbytek x,
  - `&`, `|`, `^` – bitový *and*, *or* and *xor*,
  - `<<`, `>>` – bitové posuny: x `<<` y dá x posunuté o y bitů doleva (je-li $y \geq 32$, vyjde vždy 0), x `>>` y je totéž doprava [$1 << 5 = 32$, $9 >> 2 = 2$],
  - `<<<`, `>>>` – bitové rotace: rotace je totéž, co posun, jen se bity vysunuté z čísla na jedné straně objevují na straně opačné [$9 >>> 2 =$ `#40000002`].
- `jump` *návěští* – skočí na instrukci označenou *návěští*m a pokračuje v provádění programu od ní.
- `halt` – zastaví provádění programu.
- `data` [`#`]*číslo* – tento příkaz nedělá vůbec nic, pouze slouží jako držáček na *číslo*, aby bylo přístupné pomocí *návěští*[*hodnota*].

Libovolné instrukci může navíc předcházet (nepovinně) podmínka typu `if` *hodnota relace hodnota*, kde *relace* je `=`, `<>`, `<`, `>`, `<=` nebo `>=`. Tehdy se instrukce vykoná, jak bylo popsáno výše, pouze pokud podmínka platí, jinak neprovede nic.

Mimo instrukcí mohou být v programu ještě komentáře: ty začínají znakem `'` a vše od něj až do konce řádku je ignorováno. Stejně tak se ignorují i prázdné řádky. V celém jazyku nikde nezáleží na velikosti písmen – velká písmenka, malá písmenka, palcové titulky, vše jedno jest.

---

[1] dalších 1463 marketingových blábolů jsme s ohledem na duševní zdraví čtenářů vypustili

[2] pra-pra-...-vnuk legendárního Švýcarského hrdiny Viléma Tella

[16] 16 kilobytů na instrukci

[640] Bill Gates, 1981

[†] Oni, pravda, nějací matematici dokázali, že je v obecném případě neřešitelná, ale pravověrný businessman přece takovým věcem nebude věřit!

Provádění programů

Program se spouští tak, že se do registru `a` uloží vstup (jediné 32-bitové číslo), registry `b` až `z` se vynulují a začnou se provádět instrukce od začátku programu. Program se zastaví, provede-li instrukci `halt`, nebo pokud "přepadne přes konec programu" (tj. pokusí se provést instrukci za koncem programu). Hodnota v registru `z` se pak interpretuje jako výstup programu a obsah všech ostatních registrů putuje na smetiště, kde je zcela ignorován.

Pokud program provede více než 1 000 000 instrukcí, považuje se za beznadějně zacyklený. (To tedy znamená, že pokud se původní program v tomto čase pro nějaký vstup x nezastaví, vaše optimalizované řešení se může pro tento vstup chovat jakkoliv.)

Vstup a výstup

Programy k optimalizování najdete ve vstupních souborech `processor`*n*`.in` ($1 \leq n \leq 5$), které se na vašich počítačích vyskytují v adresáři `/aux/mo/public/problems/processor/`. Vaším úkolem je submitnout optimalizované verze těchto programů (tj. nejkratší ekvivalentní programy, jaké dovedete vytvořit). Ty zapište do souborů `processor`*n*`.out` a zavolejte `submit processor` *n*. Nezapomeňte ale, že je potřeba `submit` zavolat pro každý výstup zvlášť. Pokud si chcete být jisti, že jste submitli všechno, příkaz `status` jistě bude vaším přítelem.

Simulátor

Při práci máte k dispozici simulátor Hexadecadia. Spustit ho můžete příkazem

`hdsim` [-g] *program* [*vstup* ...]

kde *program* je jméno souboru s programem a *vstup* je číslo, která se má použít jako vstup (pozor, Unixový shell používá `#` jako znak začátku komentáře, takže ho musíte zapisovat jako `\#`). Pokud nezadáte žádný *vstup*, program se pouze zkontroluje na syntaktické chyby; pokud *vstup*ů zadáte více, spustí se program postupně pro všechny. Na konci simulátor vypíše výstup programu.

Pokud uvedete přepínač `-g`, simulátor bude vypisovat, jaké instrukce přesně provádí a jaké jsou jejich výsledky.

Bodování

Každý program, který submitnete, bude vyhodnocen zvlášť. Porovnáme ho s původním programem na nějaké množině vstupů. Pokud se bude na libovolném vstupu lišit, bude bez milosti ohodnocen 0 body. Tento chmurný osud ho nemine, ani když bude delší nebo stejně dlouhý, než původní program. Ve všech ostatních případech dostane body podle této formule:

$$\max\left(20 - 5 \cdot (y - b),\ 1\right),$$

kde y je délka vašeho řešení (v instrukcích) a b je délka nejlepšího známého řešení (včetně našeho).

Příklad 1

Tento program počítá, kolik dělitelů má zadané číslo:

```
        b=1                     ' 1 je první kandidát na dělitele
pool:   c=a%b                   ' dělí a?
        if c=0 z=z+1            ' ano-li, zvyšme z
        b=b+1                   ' a šup tam s dalším kandidátem
        if b<=a jump pool       ' znovu ... nuda
```

Příklad 2

Následující program vrací binární zrcadlový obraz zadaného čísla (tj. číslo, jehož binární zápis je stejný jako u zadaného čísla, jenže pozpátku):

```
        b=32                    ' kolik bitů má číslo
loop:   c=a&1                   ' vytáhne nejnižší bit čísla a
        a=a>>1
        z=z<<1                  ' a přidá ho na konec z
        z=z|c
        b=b-1                   ' a znovu...
        if b>0 jump loop
```