Processor — CPSPC 2004 Praha, Deň 4

Submit:	`processor*.out`	Time limit:	4 hours
Input:	`processor*.in`	Memory limit:	of your own
Output:	`processor*.out`	Points:	100

Asi ste už počuli, že známy vynálezca prof. I. N. Tell[2] vynašiel nový druh procesora – Hexadecadium. Avšak kódovanie inštrukcií použité týmto procesorom nie je príliš úsporné[16] a ako už všetci viete, 640 KB stačia každému[640], takže prof. Tell musí nájsť metódu ako optimalizovať programy skrátením ich dĺžky.

Prof. Tell by chcel vytvoriť ultimátny optimalizátor – t.j. program, ktorý na vstupe dostane zdrojový kód nejakého programu P a vypíše najkratší (s minimálnym počtom inštrukcií v zdrojovom súbore) možný program Q, ktorý stále ráta to isté ako P (teda pre každý vstup je výstup Q identický výstupu P). Jediná výnimka môžu byť vstupy, na ktorých P nezastaví – v takých prípadoch môže byť výstup Q ľubovoľný alebo Q nemusí vôbec skončiť.

Táto úloha sa však ukázala extrémne ťažká[†] a hoci prof. Tell odhodlane verí, že to dokáže, potrebuje začať predávať jeho procesory čo najskôr. Preto sa rozhodol zľahčiť si tento problém tak, že bude uvažovať iba programy, ktorých vstup a výstup sú obe jediné 32-bitové číslo a dočasne nahradiť ultimátny optimizátor lacnou pracovnou silou. Rozdal programy svojim zamestnancom a oni majú vytvoriť najkratší ekvivalentný program, ktorého sú schopný. Samozrejme ich výplata prudko závisí na dĺžke vytvoreného programu. Toto je vaša šanca privyrobiť si nejaké tie peniaze počas vášho pobytu v Prahe.

Popis procesora

Hexadecadium má 26 registrov pomenovaných od `a` po `z`. Každý z týchto registrov obsahuje jedno 32-bitové celé číslo bez znamienka.

[*label*:] [`if` *value relation value*] *command*

[časti v hranatých zátvorkách môžu byť vynechané]. Návestie *label* nemá žiadny efekt na význam inštrukcie, ale umožnuje sa na ňu odvolávať v skokoch a indexovaných hodnotách. Časť `if` umožnuje vykonať inštrukciu len za určitých podmienok (pozri nižšie). Existujú nasledovné príkazy (*command*):

- *register* `=` *value* – zapíše hodnotu *value* do daného registra, kde *value* je buď:
  - *number* – číslo v desiatkovej sústave (0. . . 4294967295), alebo
  - `#`*number* – číslo v hexadecimálnej (šesťnástkovej) sústave (0. . . `ffffffff`), alebo
  - *register* – hodnota daného registra, alebo
  - *label*[*value*] – obsah dátovej (`data`) inštrukcie nachádzajúcej sa *value* inštrukcií za inštrukciou na návestí *label* (*value* rovné nule použije inštrukciu na návestí *label*, 1 použije nasledujúcu atď.). Ak žiadna taká inštrukcia neexistuje (nie je to `data` inštrukcia alebo by sa nachádzala za koncom programu), tak sa vráti nula. *value* nesmie obsahovať žiadne ďalšie hranaté zátvorky.
- *register* `=` *value operator value* – urobí jednu operáciu na dvoch hodnotách a vráti výsledok do *register*. (ak sa nejaký register nachádza na obidvoch stranách `=`, nová hodnota je zapísaná do registra po prečítaní obidvoch hodnôt *value*). Dostupné sú nasledujúce operácie (*operator*):
  - `+`, `-`, `*` – štandardné aritmetické operácie. Ak pretečú, výsledok sa zoberie modulo 2^{32},
  - `/`, `%` – štandardné celočíselné delenie a zvyšok po delení. Delenie x nulou dá výsledok nulu a zvyšok x.
  - `&`, `|`, `^` – *and*, *or* a *xor* po bitoch,
  - `<<`, `>>` – bitové posuny: x `<<` y posunie x o y bitov doľava (ak $y \geq 32$, výsledok je vždy nula), x `>>` y je to isté doprava $[1 << 5 = 32,\ 9 >> 2 = 2]$,
  - `<<<`, `>>>` – bitové rotácie: rotácia je to isté ako posun, ale bity posunuté mimo 32-bitovú hodnotu sa objavia na opačnej strane [9 >>> 2 = #40000002].
- `jump` *label* – skočí na inštrukciu označenú *label* a vykonávanie inštrukcií pokračuje tam.
- `halt` – zastaví vykonávanie programu.
- `data` [`#`]*number* – Tento príkaz nevykonáva nič. Je to iba miesto na uloženie pre dané číslo, ku ktorému sa dá pristupovať pomocou *label*[*value*].

Každej inštrukcii môže predchádzať nepovinná podmienka tvaru `if` *value relation value*, kde *relation* je `=`, `<>`, `<`, `>`, `<=` alebo `>=`. Potom inštrukcia vykoná to isté ako spomenuté vyššie, keď je podmienka splnená, inak nevykoná nič.

Program môže takisto obsahovať komentáre: Ak sa hocikde vyskytne znak `'`, všetko napravo od neho po koniec riadku sa ignoruje. Prázdne riadky sú ignorované taktiež. Celý jazyk je "case-insensitive" – nerozlišuje veľké a malé písmená.

[2] pra-pra-. . . -vnuk legendárneho švajčiarskeho hrdinu Williama Tella
[16] 16 kilobytov na každú inštrukciu
[640] Bill Gates, 1981
[†] Matematici už dávno dokázali, že vo všeobecnosti je to nemožné, ale ortodoxní biznismeni samozrejme neveria v také hlúposti. . .

Vykonávanie programu

Keď sa program začne vykonávať, register `a` obsahuje vstup (jedno 32-bitové číslo), registre `b` až `z` obsahujú nuly a vykonávanie začne prvou inštrukciou programu. Program zastaví, keď sa vykoná inštrukcia alebo "prepadne cez koniec programu" (snaží sa vykonať inštrukciu za koncom programu). Potom hodnota registra `z` bude reprezentovať výstup programu a hodnoty všetkých ostatných registrov budú ignorované.

Po vykonaní 1 000 000 inštrukcií je program označený ako večne cykliaci sa. (To znamená, že keď pôvodný program neskončí v tomto limite pre nejaký vstup x, váš program nemusí robiť to isté pre tento vstup.)

Vstup a výstup

Programy, ktoré máte optimalizovať nájdete vo vstupných súboroch `processor`n`.in` ($1 \le n \le 5$). Tieto budú uložené v adresári `/aux/mo/public/problems/processor/`. Vy by ste mali odovzdať optimalizované verzie programov (t.j. najkratší ekvivalentný program, ktorý ste schopný vymyslieť). Tieto napíšete do súborov `processor`n`.out` a odovzdáte zavolaním `submit processor` n. `submit` má byť volaný pre každý výstup zvlášť. Ak si chcete byť istí, že ste odovzdali všetko, príkaz `status` je váš kamarát.

Simulátor

Simulátor Hexadecadia máte k dispozícii. Môžete ho spustiť

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

kde *program* je meno súboru s vaším programom, *input* je číslo použité ako vstup pre váš program (pozor, `#` je použité ako komentár v Unixovom shelli, takže ho musíte napísať ako `\#`). Ak nedodáte žiadny vstup, v programe budú skontrolované iba syntaktické chyby. Ak dodáte viac vstupov, program bude spustený pre každý z nich zvlášť.

Ak použijete prepínač `-g`, simulátor vypíše detailne vykonávanie vášho programu.

Bodovanie

Každý program, ktorý odovzdáte bude zvlášť hodnotený. Bude porovnaný s originálnym programom na množine vstupov. Ak vysvitne, že váš program dá rôzny výstup na nejakom vstupe, dostanete 0 bodov. Ak vaša dĺžka vášho programu bude dlhšia alebo rovná dĺžke pôvodného programu, dostanete taktiež 0 bodov. Vo všetkých ostatných prípadoch dostanete body podľa nasledovného vzorca:

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

kde y je dĺžka vášho riešenia (počet inštrukcií) a b je dĺžka najlepšieho známeho (vrátane nášho).

Príklad 1

Tento program zráta počet deliteľov daného čísla:

```
        b=1                     ' 1 je prvý kandidát na deliteľa
pool:   c=a%b                   ' delí a?
        if c=0 z=z+1            ' ak áno, zvýš z
        b=b+1                   ' ďalší kandidát na deliteľa
        if b<=a jump pool       ' skús znovu
```

Príklad 2

Nasledovný program vráti binárny zrkadlový obraz daného čísla (t.j. číslo, ktorého binárny zápis je rovnaký ako binárny zápis daného číslo, ale odzadu):

```
        b=32                    ' počet bitov v integeri
loop:   c=a&1                   ' zisti najnižší bit z a
        a=a>>1
        z=z<<1                  ' a daj ho naspodok z
        z=z|c
        b=b-1                   ' iteruj
        if b>0 jump loop
```