

Assembler

Danas programeri mogu da biraju programski jezik u kojem će programirati i imaju na raspolaganju stotine programskih jezika. Međutim, nekada računari nisu bili ovako napredni i moglo se programirati samo u jeziku Assembleru, programskom jeziku jako niskog nivoa. Naime, Assembler je veoma sličan mašinskom jeziku (a mašinski jezik je prosto kombinacija nula i jedinica koje računar "prepoznaje"), ali ipak ne morate programe pisati u nulama i jedinicama. Za potrebe ovog zadatka pretpostavimo da svakoj mašinskoj instrukciji odgovara jedna instrukcija u Assembleru. Svaka Assemblerska instrukcija se sastoji iz dva dijela, ime instrukcije i parametri instrukcije. Npr. jedna instrukcija može izgledati ovako:

SABERI broi

Za potrebe ovog zadatka pretpostavit ćemo da kada programirate u Assembleru imate na raspolaganju memoriju i još jedan poseban pomoćni registar. Memorija je organizovana kao niz memorijskih lokacija, na svakoj memorijskoj lokaciji se nalazi jedan broj dužine 32 bita (tj. binarni zapis tog broja ima najviše 32 cifre). Izvršavanje programa počinje od prve memorijske lokacije.

Imamo na raspolaganju četiri instrukcije:

LOAD n STORE n ADD n BNEG n

Instrukcija "LOAD n" ima značenje:

U pomoćni registar kopiraj broj koji se nalazi na memorijskoj lokaciji n.

Instrukcija "STORE n" ima značenie:

Broj iz pomoćnog registra kopiraj u memorijsku lokaciju n.

Instrukcija "ADD n" ima značenje:

Broj iz pomoćnog registra saberi sa brojem koji se nalazi na memorijskoj lokaciji n i rezultat prepiši u pomoćni registar.

Instrukcija "BNEG n" ima značenje:

Ukoliko je broj u pomoćnom registru negativan, nastavi izvršavanje programa od memorijske lokacije n.

Svaka Assemblerska naredba se prevodi u jednu mašinsku instrukciju, tj. jedan 32 bitni broj. Za te potrebe svakoj instrukciji ćemo dodijeliti jedinstven kod: (LOAD,000), (STORE,001), (ADD,010), (BNEG,011). Kada prevodimo naredbu u mašinsku instrukciju, prvo zapišemo kod intrukcije pa njen parametar u binarnom zapisu. Tako, npr. instrukcija "ADD 12" će binarno biti kodirana kao:

*010*00000000000000000000000001100

Ovdie 010 označava kod instrukcije, dok 00000000000000000000001100 je parametar, tj. broj 12 napisan u binarnom sistemu. Primjetimo da se broj 12 u binarnom sistemu biše kao 1100, ali smo ispred dodali dovoljno nula, tako da dobijemo 32 bitni broj (broj sa 32 binarne cifre). Također, prva cifra broja neka nam označava njegov znak, broj je pozitivan ukoliko je prva cifra 0, negativan ukoliko je prva cifra 1. Tako bi se instrukcija ADD -12 zapisala ovako:

*010*1000000000000000000000001100

Podvučena jedinica nam govori da je riječ o negativnom broju.

Također, ukoliko se na nekoj lokaciji ne nalazi nikakva instrukcija, nego samo neki običan broj, prve tri cifre tog broja prosto ignorišemo (one su namijenjene samo za kod instrukcije), tako da ako je sadržaj memorijske lokacije 0100000000000000000000000001100 i potrebno je pročitati broj s te lokacije, smatramo da ćemo pročitati broj 12.



Vaš zadatak je da izvšite mašinski program. Precizinije, bit će vam poznato koliko imate memorije na početku (tj. koliko vam je memorijskih lokacija dostupno) i znat ćete stanje memorije, tj. koji se broj nalazi na svakoj lokaciji. Vi trebate da izršite program, za koji ćemo pretpostaviti da počinje od prve memorijske lokacije. Program staje onda kada naiđe na instrukciju 1111111111111111111111111111111111 Vaš zadatak je da ispjšete stanje memorije nakon izvšenog programa.

Uzmimo, na primier, da imamo memoriju koja se sastoji od 10 memorijskih lokacija (podsjetimo, svaka lokacija je jedan 32-bitni broj). Neka je stanje memorije sljedeće:

0000000000000000000000000000110 01000000000000000000000000000111 000000000000000000000000000000001001 0110000000000000000000000001010 111000000000000000000000000000101 010000000000000000000000000110 11111111111111111111111111111111111

Prva instrukcija je LOAD 6. (prva 3 bita su 000. što je kod za instrukciju LOAD, dok je parametar binarno zapisan broj 6). Nakon te insturkcije u pomoćnom registru imamo broj 5, jer se on nalazi na memorijskoj lokaciji 6. Druga instrukcija je ADD 7, pa ćemo broj iz pomoćnog registra (a to je 5), sabrati sa brojem 6, koji se nalazi na memorijskoj lokaciji 7. Rezultat je 11 i sada se u pomoćnom registru nalazi broj 11. Sljedeća naredba je STORE 8, pa ćemo broj 11 smjestiti na memorijsku lokaciju 8. Nakon toga slijedi naredba LOAD 9. Na memorijskoj lokaciji 9 se nalazi broj -2 (negativan je broj jer mu je prva cifra 1), pa se on kopira u pomoćni registar. Dolazimo do naredbe BNEG 10. Pošto je -2, broj u pomoćnom registru, negativan, izvrašavanje nastavljamo od memorijske lokacije 10. Na memorijskoj lokaciji 10 su sve jedinice, pa program staje. U toku programa smo izmijenili memorijsku lokaciju 8, na nju smo upisali broj 11, pa stanje memorije sada izgleda:

00000000000000000000000000000110 01000000000000000000000000000111 01100000000000000000000000001010 010000000000000000000000000110 00000000000000000000000000001011 11111111111111111111111111111111111

Dakle, promijenjen je samo broj na memorijskoj lokaciji 8.

Ulazni podaci

Ulazni podaci se čitaju iz tekstualne datoteke "asm.in".

Na prvoj liniji se nalazi prirodan broj N, koji neće biti veći od 1000. Nakon toga slijedi N redova koji opisuju stanje memorije. Svaki red se sastoji iz 32 znaka, 0 ili 1.

Izlazni podaci

Izlazne podatke je potrebno ispisati u datoteku "asm.out".

U izlaznu datoteku treba ispisati stanje memorije nakon izvršenog programa. Garantuje se da program neće upasti u beskonačnu petlju, tj. da će doći do naredbe za kraj programa.

Primjeri

Primjer 1

asm.in

asm.out