Bit Shift Registers

Kristofer je inženjer koji razvija novu vrstu procesora.

Procesor može pristupiti b-bitnim memorijskim ćelijama koje su međusobno različite i kojih ima m (gdje je $m = 100$ i $b = 2000$), koja nazivamo registrima i koje su označene brojevima od 0 do $m - 1$. Označimo registre sa $r[0], r[1], \ldots, r[m-1]$. Svaki registar je niz od b bita, označenih od 0 (krajni desni bit) do $b - 1$ (krajni lijevi bit). Za sve i $(0 \leq i \leq m - 1)$ i sve j $(0 \leq j \leq b - 1)$, označimo sa $r[i][j]$ bit j registra i.

Za svaki niz bita $d_0, d_1, \ldots, d_{l-1}$ (proizvoljne dužine l), cjelobrojna vrijednost niza je $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. Kažemo da je cjelobrojna vrijednost sačuvana u registru i jednaka cjelobrojnoj vrijednosti niza bitova u registru tj. $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

Procesor ima 9 tipova instrukcija pomoću kojih može modifikovati sadržaj registara. Svaka instrukcija koristi jedan ili više registara i čuva rezultat u nekom od registara. U nastavku, koristićemo $x := y$ da označimo operaciju promjene vrijednosti x u y. Značenje svake od instrukcija je sljedeće:

$move(t, y)$: Kopira niz bitova iz registra y u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := r[y][j]$.

$store(t, v)$: Registar t biće jednak v, gdje je v niz od b bita. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := v[j]$.

$and(t, x, y)$: Primjenjuje operaciju bitwise-AND nad registrima x i y i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako su oba $r[x][j]$ i $r[y][j]$ jednaka 1 i $r[t][j] := 0$ u suprotnom.

$or(t, x, y)$: Primjenjuje operaciju bitwise-OR nad registrima x i y i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako je bar jedan od $r[x][j]$ i $r[y][j]$ jednak 1 i $r[t][j] := 0$ u suprotnom.

$xor(t, x, y)$: Primjenjuje operaciju bitwise-XOR nad registrima x i y, i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1$ ako je tačno jedan od $r[x][j]$ i $r[y][j]$ jednak 1 i $r[t][j] := 0$ u suprotnom.

$not(t, x)$: Primjenjuje operaciju bitwise-NOT nad registrom x i smješta rezultat u registar t. Za sve j $(0 \leq j \leq b - 1)$, $r[t][j] := 1 - r[x][j]$.

$left(t, x, p)$: Pomjera ulijevo bitove registra x za p pozicija i smješta rezultat u registar t. Rezultat pomjeranja bitova registra x ulijevo za p pozicija je niz v koji ima b bita. Za sve j $(0 \leq j \leq b - 1)$, $v[j] = r[x][j - p]$ ako je $j \geq p$ i $v[j] = 0$ u suprotnom. Za sve j $(0 \leq j \leq b - 1)$, set $r[t][j] := v[j]$.

$right(t, x, p)$: Pomjera nadesno bitove registra x za p pozicija i smješta rezultat u registar t. Rezultat pomjeranja bitova registra x nadesno za p pozicija je niz v koji ima b bita. Za sve j $(0 \leq j \leq b-1)$, $v[j] = r[x][j+p]$ if $j \leq b - 1 - p$ i $v[j] = 0$ inače. Za sve j $(0 \leq j \leq b-1)$, $r[t][j] := v[j]$.

$add(t, x, y)$: Sabira cjelobrojne vrijednosti sačuvane u registrima x i y i smješta rezultat u registar t. Sabiranje se obavlja po modulu 2^b. Formalno, neka je X cjelobrojna vrijednost sačuvana u registru x i Y cjelobrojna vrijednost sačuvana u registru y prije izvršenja ove operacije. Neka je T cjelobrojna vrijednost sačuvana u registru t nakon operacije. Ako je $X + Y < 2^b$, postaviti bitove registra t tako da je $T = X + Y$. U suprotnom, postaviti bitove registra t tako da je $T = X + Y - 2^b$.

Kristofer želi da riješi dva tipa zadataka primjenom novog procesora. Tip zadatka je označen brojem s. Za oba tipa zadatka potrebno je kreirati progam tj. niz instrukcija koje smo ranije opisali.

Program ima ulaz koji sadrži n cijelih brojeva $a[0], a[1], \ldots, a[n-1]$, pri čemu svaki broj ima k bita, tj. $a[i] < 2^k$ ($0 \leq i \leq n-1$). Prije pokretanja programa, svi ulazni brojevi su smješteni redom u registar 0, tako da za sve i $(0 \leq i \leq n-1)$ cjelobrojna vrijednost niza od k bita $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$ je jednako $a[i]$. Obratite pažnju da je $n \cdot k \leq b$. Svi ostali bitovi u registru su 0 (tj. oni sa indeksima $n \cdot k$ i $b - 1$, uključivo) a svi bitovi u svim drugim registrima su postavljeni na 0.

Pokretanje programa se sastoji izvršavanja instrukcija onim redom kako su napisane. Nakon izvršavanja posljednje instrukcije, izlaz programa se izračunava na osnovu konačnih vrijednosti bitova u registru 0. Naime, izlaz je niz od n cijelih brojeva $c[0], c[1], \ldots, c[n-1]$, gdje za sve i ($0 \leq i \leq n-1$), $c[i]$ predstavlja cjelobrojnu vrijednost niza bitova od $i \cdot k$ do $(i+1) \cdot k - 1$ registra 0. Obratite pažnju da nakon pokretanja programa, ostali bitovi registra 0 (sa indeksima najmanje $n \cdot k$) i svi bitovi u ostalim registrima mogu biti proizvoljni.

Prvi zadatak $(s = 0)$ je određivanje najmanjeg cijelog broja u ulaznom nizu cijelih brojeva $a[0], a[1], \ldots, a[n-1]$. Preciznije, $c[0]$ treba da je minimum brojeva $a[0], a[1], \ldots, a[n-1]$. Vrijednosti $c[1], c[2], \ldots, c[n-1]$ mogu biti proizvoljne.

Drugi zadatak $(s = 1)$ je sortiranje ulaznih cijelih brojeva $a[0], a[1], \ldots, a[n-1]$ u neopadajući poredak. Preciznije, za sve i ($0 \leq i \leq n-1$), $c[i]$ treba da je jednak $1 + i$-tom najmanjem broju među brojevima $a[0], a[1], \ldots, a[n-1]$ (tj. $c[0]$ je najmanji broj među ulaznim brojevima).

Pomozite Kristoferu da napiše programe za ova dva zadatka, pri čemu programi mogu sadržati najviše q instrukcija.

Detalji implementacije

Trebali biste primijeniti sljedeći postupak:

```
void construct_instructions(int s, int n, int k, int q)
```

- s: vrsta zadatka.
- n: broj cijelih brojeva u ulazu.
- k: broj bitova u svakom ulaznom cijelom broju.

- q: maksimalan broj dozvoljenih instukcija.
- Ovaj se postupak poziva tačno jednom i trebao bi sadržati sekvenciju instrukcija za izvršavanje traženog zadatka.

Ovaj postupak bi trebao pozivati jedan ili više sljedećih postupaka za konstruiranje sekvence instrukcija:

```
void _move(int t, int y)
void _store(int t, bool[] v)
void _and(int t, int x, int y)
void _or(int t, int x, int y)
void _xor(int t, int x, int y)
void _not(int t, int x)
void _left(int t, int x, int p)
void _right(int t, int x, int p)
void _add(int t, int x, int y)
```

- Svaka procedura dodaje $move(t, y)$ $store(t, v)$, $i(t, x, y)$, $ili(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ ili $add(t, x, y)$ instrukcije u program , odnosno.
- Za sve relevantne instrukcije, t, x, y moraju biti najmanje 0 $, a najviše $ m-1 $.
- Za sve relevantne instrukcije, t, x, y nisu nužno po parovima različiti.
- Za instrukcije $left$ i $right$ p mora biti najmanje 0, a najviše b.
- Za instrukciju $store$ dužina v mora biti b.

Moguće je također pozvati sljedeću proceduru kako bi testirali svoje rješenje:

```
void _print(int t)
```

- Svaki poziv ove procedure će se zanemariti tijekom ocjenjivanja vašeg rješenja.
- U uzorku za ocjenjivanje, ovaj postupak dodaje operaciju $print(t)$ programu.
- Kada sample grader naiđe na operaciju $print(t)$ tokom izvršavanja programa, ispisuje n $k-bitne$ cijele brojeve formirane od prvih $n \cdot k$ bitova registra t (vidi Odjeljak "Sample grader" za detalje).
- t mora zadovoljiti $0 \leq t \leq m-1$.
- Bilo koji poziv ove procedure ne dodaje se broju izgrađenih uputstava.

Nakon dodavanja posljednje instrukcije, `construct_instructions` bi se trebao vratiti. Program se zatim procjenjuje na određenom broju testnih slučajeva, od kojih svaki određuje ulaz koji se sastoji od n k -bitnih cijelih brojeva $a[0], a[1], \ldots, a[n-1]$. Vaše rješenje prolazi zadani testni slučaj ako izlaz programa $c[0], c[1], \ldots, c[n-1]$ za navedeni ulaz zadovoljava sljedeće uvjete:

- Ako je $s = 0$, $c[0]$ bi trebala biti najmanja vrijednost među $a[0], a[1]$, $ldots, a[n-1]$.
- Ako je $s = 1$, za svaki i ($0 \leq i \leq n-1$), $c[i]$ bi trebao biti $1+i$ -ti najmanji cijeli broj među $a[0], a[1], \ldots, a[n-1]$.

Ocjenjivanje vašeg rješenja može rezultirati jednom od sljedećih poruka o grešci:

- `Invalid index`: kao parametar t, x ili y za neki poziv jedne od procedura naveden je netačan (moguće negativan) indeks registra.
- `Value to store is not b bits long`: dužina v data u `_store` nije jednaka b.
- `Invalid shift value`: vrijednost p data u `_left` ili `_right` nije između 0 i b.
- `Too many instructions`: vaš postupak je pokušao dodati više od q instrukcija.

Primjeri

Primjer 1

Pretpostavimo da je $s = 0,\ n = 2,\ k = 1,\ q = 1000$. Postoje dva ulaza cijelih brojeva $a[0]$ i $a[1]$, od kojih svaka ima $k = 1$ bit. Prije izvršavanja programa, $r[0][0] = a[0]$ i $r[0][1] = a[1]$. Svi ostali bitovi u procesoru postavljeni su na 0. Nakon izvršavanja svih instrukcija u programu, trebamo imati $c[0] = r[0][0] = \ min(a[0], a[1])$, što je najmanje od $a[0]$ i $a[1]$.

Postoje samo 4 moguća ulaza u program:

- Slučaj 1: $a[0] = 0, a[1] = 0$
- Slučaj 2: $a[0] = 0, a[1] = 1$
- Slučaj 3: $a[0] = 1, a[1] = 0$
- Slučaj 4: $a[0] = 1, a[1] = 1$

Možemo primijetiti da je za sva 4 slučaja $min(a[0], a[1])$ jednako bitnom-AND od $a[0]$ i $a[1]$. Stoga je moguće rješti pravljenje programa upućivanjem sljedećih instrukcija:

1. `_move (1, 0)`, koji dodaje instrukciju za kopiranje $r[0]$ u $r[1]$.
2. `_right (1, 1, 1)`, koji dodaje instrukciju koja uzima sve bitove u $r[1]$, pomiče ih udesno za 1 bit, a zatim rezultat vraća u $r[1]$. Budući da je svaki cijeli broj dugačak 1 $-bit, torezultiradaje$ r [1] [0] $jednak$ a [1] $.
3. `_and (0, 0, 1)`, koji dodaje instrukcije za preuzimanje bitnog-AND od $r[0]$ i $r[1]$, a zatim rezultat pohranjuje u $r[0]$. Nakon izvršavanja ove instrukcije, $r[0][0]$ se postavlja na bit-AND od $r[0][0]$ i $r[1][0]$, što je jednako bitovima- AND od $a[0]$ i $a[1]$, po želji.

Primjer 2

Pretpostavimo da je $s = 1,\ n = 2,\ k = 1,\ q = 1000$. Kao i u ranijem primjeru, postoje samo 4 moguća ulaza u program. Za sva 4 slučaja, $min(a[0], a[1])$ je bit-AND od $a[0]$ i $a[1]$, i $max(a[0], a[1])$ je bit-OR $a[0]$ i $a[1]$. Moguće rješenje je upućivanje sljedećih Instrukcija:

1. `_move(1,0)`
2. `_right(1,1,1)`
3. `_and(2,0,1)`
4. `_or(3,0,1)`
5. `_left(3,3,1)`
6. `_or(0,2,3)`

Nakon izvršavanja ovih instrukcija, $c[0] = r[0][0]$ sadrži $min(a[0], a[1])$ i $c[1] = r[0][1]$ sadrži $max(a[0], a[1])$, koji sortira ulaz.

Ograničenja

- $m = 100$
- $b = 2000$
- $0 \leq s \leq 1$
- $2 \leq n \leq 100$
- $1 \leq k \leq 10$
- $q \leq 4000$
- $0 \leq a[i] \leq 2^k - 1$ (za svaki $0 \leq i \leq n - 1$)

Podzadaci

1. (10 bodova) $s = 0, n = 2, k \leq 2, q = 1000$
2. (11 bodova) $s = 0, n = 2, k \leq 2, q = 20$
3. (12 bodova) $s = 0, q = 4000$
4. (25 bodova) $s = 0, q = 150$
5. (13 bodova) $s = 1, n \leq 10, q = 4000$
6. (29 bodova) $s = 1, q = 4000$

Sample Grader

Sample grader čita ulaz u sljedećem formatu:

- line 1 : $s\ n\ k\ q$

Nakon toga slijedi t redovi, koji opisuju jedan testni slučaj. Svaki testni slučaj dostupan je u sljedećem formatu:

- $a[0]\ a[1]\ \ldots\ a[n-1]$

i opisuje test slučaj čiji se ulaz sastoji od n cijelih brojeva $a[0], a[1], ..., a[n-1]$. Opis svih testnih slučajeva slijedi jedan redak koji sadrži samo -1.

Sample grader prvo poziva `construct_instructions (s, n, k, q)`. Ako ovaj poziv krši neko ograničenje opisano u izjavi o problemu, sample grader ispisuje jednu od poruka o grešci navedenu na kraju odjeljka "Detalji implementacije" i izlazi. Inače, sample grader prvo ispisuje svaku naredbu dodanu `construct_instructions (s, n, k, q)` redom. Za upute $store,\ v$ se ispisuje iz indeksa 0 do indeks $b - 1$.

Zatim sample grader obrađuje testove po redu. Za svaki testni slučaj pokreće konstruirani program na ulazu testnog slučaja.

Za svaku operaciju $print(t)$, neka $d[0], d[1], \ldots, d[n-1]$ bude niz cijelih brojeva, takav da za svaki i ($0 \leq i \leq n - 1$), $d[i]$ je cijela vrijednost niza bitova $i \cdot k$ do $(i + 1) \cdot k - 1$ registra t

(kada se operacija se izvršava). Grader ispisuje ovaj niz u sljedećem formatu: `register` t: $d[0]\ d[1]\ \ldots\ d[n-1]$.

Nakon izvršavanja svih instrukcija, sample grader ispisuje izlaz programa.

Ako je $s = 0$, rezultat sample gredera za svaki testni slučaj je u sljedećem formatu:

- $c[0]$.

Ako je $s = 1$, rezultat sampel gradera za svaki testni slučaj je u sljedećem formatu:

- $c[0]\ c[1]\ \ldots\ c[n-1]$.

Nakon izvršavanja svih testih slučajeva, grader ispisuje `number of instructions:` X gdje je X broj instrukcija u vašem programu.