# Bit Shift Registers

De technicus Christopher werkt aan een nieuw soort computer processor.

De processor heeft toegang tot $m$ verschillende $b$-bit geheugen cellen (waarbij $m = 100$ en $b = 2000$), die **registers** genoemd worden en genummerd zijn vanaf $0$ tot en met $m - 1$. We noemen de registers als $r[0], r[1], \ldots, r[m-1]$.

Elk register is een array van $b$ bits, genummerd vanaf $0$ (het meest rechtse bit) tot en met $b - 1$ (het meest linkse bit). Voor iedere $i$ ($0 \leq i \leq m - 1$) en iedere $j$ ($0 \leq j \leq b - 1$) geeft $r[i][j]$ de $j$-de bit van register $i$ aan.

Voor elke rij bits $d_0, d_1, \ldots, d_{l-1}$ (van willekeurige lengte $l$) is de **integer waarde** van een rij gelijk aan $2^0 \cdot d_0 + 2^1 \cdot d_1 + \ldots + 2^{l-1} \cdot d_{l-1}$. We zeggen dat de **integer waarde opgeslagen in register** $i$ de integer waarde van de rij van zijn bits is, oftewel $2^0 \cdot r[i][0] + 2^1 \cdot r[i][1] + \ldots + 2^{b-1} \cdot r[i][b-1]$.

De processor heeft $9$ soorten **instructies** die je kunt gebruiken om bits in de registers te bewerken. Elke instructie werkt op één of meerdere registers en slaat het resultaat op in één van de registers. In het stuk hieronder gebruiken we $x := y$ om de operatie aan te geven die de waarde van $x$ aanpast zodat deze gelijk wordt aan $y$. De bewerkingen die alle soorten instructies uitvoeren staan hieronder beschreven.

- $move(t, y)$: Kopiëer de array van bits in register $y$ naar register $t$. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := r[y][j]$ .
- $store(t, v)$: Geef register $t$ een waarde gelijk aan $v$, waarbij $v$ een array van $b$ bits is. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := v[j]$.
- $and(t, x, y)$: Neem de bitwise-AND van registers $x$ en $y$ en sla het resultaat op in register $t$. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := 1$ toe indien **zowel** $r[x][j]$ **als** $r[y][j]$ gelijk zijn aan $1$, en maak anders $r[t][j] := 0$.
- $or(t, x, y)$: Neem de bitwise-OR van registers $x$ en $y$ en sla het resultaat op in register $t$. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := 1$ als **ten minste één** van $r[x][j]$ en $r[y][j]$ gelijk is aan $1$ is, en maak anders $r[t][j] := 0$.
- $xor(t, x, y)$: Neem de bitwise-XOR van registers $x$ en $y$ en sla het resultaat op in register $t$. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := 1$ als er **precies één** van $r[x][j]$ en $r[y][j]$ gelijk is aan $1$, en maak anders $r[t][j] := 0$.
- $not(t, x)$: Neem de bitwise-NOT van register $x$ en sla het resultaat op in register $t$. Voor elke $j$ ($0 \leq j \leq b - 1$), maak $r[t][j] := 1 - r[x][j]$.

- $left(t, x, p)$: Schuif alle bits die in register $x$ zitten $p$ stappen naar links en sla het resultaat op in register $t$. Het resultaat van het $p$ stappen naar links schuiven van bits in register $x$ is een array $v$ bestaand uit $b$ bits. Voor elke $j$ ( $0 \leq j \leq b-1$) is $v[j] = r[x][j-p]$ als $j \geq p$ en anders is $v[j] = 0$. Voor elke $j$ ( $0 \leq j \leq b-1$), maak $r[t][j] := v[j]$.

- $right(t, x, p)$: Schuif alle bits die in register $x$ zitten $p$ stappen naar rechts en sla het resultaat op in register $t$. Het resultaat van het $p$ stappen naar rechts schuiven van bits in register $x$ is een array $v$ bestaand uit $b$ bits. Voor elke $j$ ( $0 \leq j \leq b-1$) is $v[j] = r[x][j+p]$ als $j \leq b-1-p$ en anders is $v[j] = 0$. Voor elke $j$ ( $0 \leq j \leq b-1$), maak $r[t][j] := v[j]$.

- $add(t, x, y)$: Tel de integer waardes opgeslagen in registers $x$ en $y$ bij elkaar op en stop het resultaat in register $t$. De optelling wordt modulo $2^b$ uitgevoerd. Formeel, noem $X$ de integer waarde, opgeslagen in register $x$, en $Y$ de integer waarde, opgeslagen in register $y$, voor de bewerking uitgevoerd is. Noem $T$ de integer waarde, opgeslagen in register $t$ na de bewerking. Als $X + Y < 2^b$ maak de bits van $t$ zodanig dat $T = X + Y$ geldt. Anders, maak de bits van $t$ zodanig dat $T = X + Y - 2^b$ geldt.

Christopher wil dat je twee soorten taken voor hem oplost waarbij je de nieuwe processor gebruikt. Het soort taak wordt beschreven door een geheel getal $s$. Voor beide soorten taken moet je een **programma** genereren, wat een rij van bovenstaande instructies is.

De **invoer** van het programma bestaat uit $n$ integers $a[0], a[1], \ldots, a[n-1]$, die elk $k$ bits hebben, oftewel $a[i] < 2^k$ ( $0 \leq i \leq n-1$). Voordat het programma wordt uitgevoerd, zijn alle invoergetallen achtereenvolgend opgeslagen in register $0$ zodanig dat voor iedere $i$ ( $0 \leq i \leq n-1$) de integer waarde van de rij van $k$ bits $r[0][i \cdot k], r[0][i \cdot k + 1], \ldots, r[0][(i+1) \cdot k - 1]$, gelijk aan $a[i]$. Merk op dat $n \cdot k \leq b$. Alle andere bits in register $0$ (oftewel de indices vanaf $n \cdot k$ tot en met $b-1$) en alle bits in alle andere registers zijn initieel $0$.

Het uitvoeren van een programma bestaat uit het uitvoeren van de instructies in de gegeven volgorde. Na de uitvoering van de laatste instructie, wordt de **uitvoer** van het programma berekend op basis van de uiteindelijke waarde van de bits in register $0$. Om precies te zien is de uitvoer een rij van $n$ integers $c[0], c[1], \ldots, c[n-1]$, waarbij voor elke $i$ ( $0 \leq i \leq n-1$), $c[i]$ de integer waarde is van de rij die bestaat uit de bits $i \cdot k$ tot en met $(i+1) \cdot k - 1$ van register $0$. Merk op dat, nadat het programma is uitgevoerd, de andere bits van register $0$ (met indices minstens $n \cdot k$) en alle bits van andere registers willekeurig mogen zijn.

- De eerste taak $(s = 0)$ is het vinden van het kleinste getal uit de rij invoergetallen $a[0], a[1], \ldots, a[n-1]$. Om precies te zijn, moet $c[0]$ gelijk zijn aan het minimum van $a[0], a[1], \ldots, a[n-1]$. De waardes van $c[1], c[2], \ldots, c[n-1]$ mogen willekeurig zijn.

- De tweede taak $(s = 1)$ is het sorteren van de invoergetallen $a[0], a[1], \ldots, a[n-1]$ in oplopende volgorde. Om precies te zijn moet voor elke $i$ ( $0 \leq i \leq n-1$) $c[i]$ gelijk zijn aan het $1+i$-e kleinste getal uit $a[0], a[1], \ldots, a[n-1]$ ( $c[0]$ is bijvoorbeeld het kleinste getal van de invoergetallen).

Voorzie Christopher van programma's, die ieder afzonderlijk bestaan uit hoogstens $q$ instructies, die deze taken kunnen oplossen.

## Implementatiedetails

Implementeer de volgende functie:

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

- $s$: soort taak.
- $n$: aantal integers in de invoer.
- $k$: aantal bits van elk invoergetal.
- $q$: maximale aantal instructies wat toegestaan is.
- Deze functie wordt precies één keer aangeroepen en moet een rij van instructies opstellen die de gevraagde taak uitvoert.

Deze functie moet één of meer van de volgende functies aanroepen om de rij instructies te construeren:

```
void append_move(int t, int y)
void append_store(int t, bool[] v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
```

- Elke functie voegt een $move(t, y)$ $store(t, v)$, $and(t, x, y)$, $or(t, x, y)$, $xor(t, x, y)$, $not(t, x)$, $left(t, x, p)$, $right(t, x, p)$ or $add(t, x, y)$ instructie toe aan het eind van het programma, respectievelijk.
- Voor alle relevante instructies, moeten $t$, $x$, $y$ minstens $0$ en hoogstens $m - 1$ zijn.
- Voor alle relevante instructies, zijn $t$, $x$, $y$ niet per se onderling verschillend.
- Voor de instructies $left$ en $right$, moet $p$ minstens $0$ en hoogstens $b$ zijn.
- Voor de instructie $store$, moet de lengte van $v$ precies $b$ zijn.

Je mag ook de volgende functie aanroepen om je te helpen bij het testen van je oplossing:

```
void append_print(int t)
```

- Elke aanroep naar deze functie zal genegeerd worden tijdens de jurering van je oplossing.
- In de voorbeeldgrader voegt de functie een $print(t)$ bewerking aan het eind van het programma toe.
- Als de voorbeeldgrader een $print(t)$ operatie tegenkomt tijdens de executie van het programma, print het $n$ $k$-bit integers gevormd door de eerste $n \cdot k$ bits van register $t$ (zie de sectie "Sample Grader" voor details).
- $t$ moet voldoen aan $0 \leq t \leq m - 1$.

- Aanroep van deze functie tellen niet mee voor het aantal geconstrueerde instructies.

Na het toevoegen van de laatste instructie, moet de functie `construct_instructions` returnen. Het programma wordt dan geëvalueerd op een aantal testcases, elk gespecificeerd door een invoer die bestaat uit $n$ $k$-bit integers $a[0], a[1], \ldots, a[n-1]$. Je oplossing wordt goed gerekend voor een testcase als de uitvoer van het programma $c[0], c[1], \ldots, c[n-1]$ voor de gegeven invoer voldoet aan de volgende voorwaarden:

- Als $s = 0$, moet $c[0]$ de kleinste waarde van $a[0], a[1], \ldots, a[n-1]$ zijn.
- Als $s = 1$, moet voor alle $i$ ($0 \leq i \leq n-1$) $c[i]$ gelijk zijn aan het $1+i$-e kleinste getal uit $a[0], a[1], \ldots, a[n-1]$.

De beoordeling van je oplossing kan één van de volgende foutmeldingen geven:

- `Invalid index`: een onjuiste (mogelijk negatieve) register index werd meegegeven als parameter $t$, $x$ of $y$ bij een aanroep van één van de functies.
- `Value to store is not b bits long`: de lengte van $v$ die doorgegeven is aan `append_store` is not gelijk aan $b$.
- `Invalid shift value`: de waarde van $p$ die gegeven is aan `append_left` of `append_right` is niet tussen $0$ en $b$ inclusief.
- `Too many instructions`: je functie probeerde meer dan $q$ instructies te gebruiken.

# Voorbeelden

## Voorbeeld 1

Stel $s = 0$, $n = 2$, $k = 1$, $q = 1000$. Er zijn twee invoergetallen $a[0]$ and $a[1]$, en elk bestaat uit $k = 1$ bit. Voordat het programma wordt uitgevoerd, geldt $r[0][0] = a[0]$ en $r[0][1] = a[1]$. Alle andere bits in de processor zijn $0$. Nadat alle instructies van het programma zijn uitgevoerd, moeten we hebben dat $c[0] = r[0][0] = \min(a[0], a[1])$, wat het minimum is van $a[0]$ en $a[1]$.

Er zijn slechts 4 mogelijke invoeren mogelijk voor het programma:

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

Het valt te zien dat voor alle 4 de gevallen, $\min(a[0], a[1])$ gelijk is aan de bitwise-AND van $a[0]$ en $a[1]$. Daarom is een mogelijke oplossing om een programma te construeren door de volgende aanroepen:

1. `append_move(1, 0)`, die een instructie toevoegt om $r[0]$ naar $r[1]$ te kopiëren.
2. `append_right(1, 1, 1)`, die een instructie toevoegt die alle bits in $r[1]$ neemt, deze $1$ bit naar links schuift, en dan het resultaat terug in $r[1]$ opslaat. Omdat alle integers uit $1$ bit bestaan, zorgt dit ervoor dat $r[1][0]$ gelijk is aan $a[1]$.

3. `append_and(0, 0, 1)`, die een instructie toevoegt die de bitwise-AND neemt van $r[0]$ en $r[1]$ en het resultaat opslaat in $r[0]$. Na het uitvoeren van deze instructie, is $r[0][0]$ nu de bitwise-AND van $r[0][0]$ en $r[1][0]$, wat gelijk is aan de bitwise-AND van $a[0]$ en $a[1]$, zoals gevraagd.

### Voorbeeld 2

Neem nu $s = 1,\ n = 2,\ k = 1,\ q = 1000$ aan. Zoals bij het eerdere voorbeeld, zijn er maar 4 invoeren mogelijk voor het programma. Voor alle 4 de gevallen, is $\min(a[0], a[1])$ de bitwise-AND van $a[0]$ en $a[1]$, en $\max(a[0], a[1])$ is de bitwise-OR van $a[0]$ en $a[1]$. Een mogelijke oplossing is het doen van de volgende aanroepen:

1. `append_move(1,0)`
2. `append_right(1,1,1)`
3. `append_and(2,0,1)`
4. `append_or(3,0,1)`
5. `append_left(3,3,1)`
6. `append_or(0,2,3)`

Nadat deze instructies zijn uitgevoerd, bevat $c[0] = r[0][0]$ de waarde $\min(a[0], a[1])$, en bevat $c[1] = r[0][1]$ de waarde $\max(a[0], a[1])$, wat de invoer sorteert.

## Randvoorwaarden

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

## Subtasks

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

## Voorbeeldgrader

De voorbeeldgrader leest de invoer in het volgende formaat:

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

Daarna volgen er een aantal regels, die ieder een enkel testgeval beschrijven. Elk testgeval wordt gegeven in het volgende formaat:

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

en beschrijft een testgeval waarvan de invoer bestaat uit $n$ integers $a[0], a[1], ..., a[n-1]$. De omschrijving van alle testgevallen wordt gevolgd door een losse regel die alleen $-1$ bevat.

De voorbeeldgrader roept eerst `construct_instructions(s, n, k, q)` aan. Als deze aanroep een van de voorwaarden in de taakomschrijving schendt, print de voorbeeldgrader een van de error-berichten die aan het eind van "Implementatiedetails" wordt genoemd en sluit af. Anders print de voorbeeldgrader elke instructie die werd toegevoegd door `construct_instructions(s, n, k, q)` op volgorde. Voor $store$ instructies wordt $v$ geprint van index $0$ tot index $b-1$.

Daarna beoordeelt de voorbeeldgrader alle testgevallen op volgorde. Voor elk testgeval voert die het gebouwde programma uit op de invoer van het testgeval.

Voor elke $print(t)$ operatie geldt: laat $d[0], d[1], \ldots, d[n-1]$ een reeks van integers zijn zodat voor iedere $i$ ( $0 \leq i \leq n-1$) $d[i]$ de integer-waarde is van de reeks bits $i \cdot k$ tot $(i+1) \cdot k - 1$ van register $t$ (wanneer de bewerking wordt uitgevoerd). De grader print deze reeks in het volgende formaat: `register` $t$: $d[0]$ $d[1]$ $\ldots$ $d[n-1]$.

Wanneer alle instructies zijn uigevoerd print de voorbeeldgrader de uitvoer van het programma.

Als $s = 0$, dan is de uitvoer van de voorbeeldgrader voor elk testgeval in het volgende formaat:

- $c[0]$.

Als $s = 1$, dan is de uitvoer van de voorbeeldgrader voor elk testgeval in het het volgende formaat:

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

Na het uitvoeren van alle testgevallen schrijft de grader `number of instructions:` $X$, waarbij $X$ het aantal instructies in je programma is.