Tölvutækni og forritun Heimadæmi 4

Ragnar Björn Ingvarsson, rbi3

14. september 2024

1

`int x = 5, y;`

Hér er `x = 5` vegna þess að við segjum að svo sé en `y` er enn óskilgreint.

`x *= y = 2;`

Hér verður `y = 2` á undan og þá margfaldast `x` við 2 og verður 10.

`x = y == 2;`

Hér verður `x` sanngildið á `y == 2` sem er þá satt, og vegna þess að sanngildi í c eru 0 eða 1 verður `x = 1`

`x == (y = 3);`

Hér erum við aðeins að skilgreina segð svo `x` er óbreytt, en `y` fær gildið 3.

`x = (unsigned)-3 > 0 ?  y << 1 :  y >> 1;`

Hér, þar sem `(unsigned)x`, $x \neq 0$ er alltaf jákvæð, þá er segðin sönn og við gefum `x` gildi `y` nema bitunum er hliðrað til vinstri um einn bita, svo í stað 11_2 verður `x = 6`, eða 110_2.

2

a) Án formerkis getum við notað alla bitana í töluna og fáum $UMax_6 = 2^6 - 1 = 63$

b) Við fáum hér að við notum einn bita í formerkið og getum þá fengið $TMax_6 = 2^5 - 1 = 31$ og $TMin_6 = -2^5 = -32$.

c) Við sjáum að $0x17 = 00010111_2 = 010111_2$ í 6-bita tvíandhverfu-formi, og $0x21 = 100001_2$. Þá framkvæmum við samlagningu og fáum:

$$\begin{array}{r} \scriptstyle 111 \\ 010111 \\ +100001 \\ \hline 111000 \end{array}$$

Svo við fáum niðurstöðuna $111000_2 = (-32 + 16 + 8)_{10} = -8_{10}$

d) -9 á 6-bita tvíandhverfu-formi er 110111_2, en ef við stýfum hana niður í fjóra bita fæst að tveir gildismestu bitarnir detta út svo við fáum $0111_2 = 7$. Gildi tölunnar breytst töluvert.

3

a) Einfölduð útgáfa myndi nota þá `char` í stað `int` svo við myndum fá fallið

```
int hmm( unsigned char n ) {
        return (~n) & (~n << 1);
}
```

i) `n = 01010101` Við byrjum þá á að fá `~n = 10101010` sem við svo hliðrum um einn til vinstri og fáum `~n << 1 = 01010100`. Síðan og-um við þessi saman og skilum 00000000.

ii) `n = 10010010` Hér gerum við það sama og fáum:

`~n = 01101101`

`~n << 1 = 11011010`

og-um það svo saman og skilum `01001000`.

b) Fallið vinnur svo að við neitum öllum bitunum, svo við fáum jákvæða útkomu fyrir núllin, og svo og-um við saman töluna og tölunni hliðrað um einn til vinstri. Þá fáum við einn allstaðar þar sem tvö núll hafa legið saman í upprunalega gildinu.

c) Við einfaldlega tékkum einnig hvort það sé núll vinstra megin við hvert núll líka, svo:

```
int hmm( unsigned int n ) {
        return (~n) & (~n << 1) & (~n >> 1);
}
```

Þetta tékkar fyrst hvort það séu tvö núll hlið við hlið og svo tékkar það hvort fyrirfinnist annað núll hinum megin.

d) Hér þurfum við einungis að sleppa því að neita tölunni því við viljum þá fá jákvæða útkomu á einum, sem er þannig nú þegar.

```
int hmm( unsigned int n ) {
        return n & (n << 1);
}
```

4

a)
```
unsigned long sammengi(unsigned long a, unsigned long b) {
        return a | b;
}
```

b)
```
unsigned long munur(unsigned long a, unsigned long b) {
        return a & ~b;
}
```

c) Segðin `1ul << i` táknar semsagt 2^i, þar sem við tökum 64 bita, og setjum minnsta sem einn en alla hina sem 0.

Síðan hliðrum við þessum bita um 0-63 til vinstri, sem gefur okkur alltaf bara einn ás, en á mismunandi stað, svo hann hefur vægi sem fer eftir hvað *i* er.

d)
```
int stakI(unsigned long a, int i) {
        return (a & (1ul << i)) > 0;
}
```

5

a) Við sjáum að með heiltöludeilingu ættum við að fá $^{-69}/_{8} = -8$ en með bitum fæst $10111011_2 \gg 3 = 11110111_2 = -9$. Þetta er vegna þess að bit shift deiling í c með neikvæðum tölum rúnnar til $-\infty$.

Til að laga þetta getum við bætt við bjögun, og fengið semsagt

$(10111011_2 + (1 \ll 3) - 1) \gg 3 = (10111011_2 + 00001000_2 - 1) \gg 3$
$= (11000100_2) \gg 3 = 11111000_2 = -8_{10}$

b) Þetta virkar ekki alltaf, til dæmis ef við tökum jákvæðu töluna $8 = 00001000_2$ og deilum með 3, þá fáum við:

$(00001000_2 + (1 \ll 3) - 1) \gg 3$
$= (00001000_2 + 00001000 - 1) \gg 3$
$= (00010000_2 - 1) \gg 3$
$= 00001111_2 \gg 3$
$= 00000001_2 = 1_{10}$

Við sjáum að þetta er alls ekki rétta niðurstaðan, þar sem rétt hefði verið að fá $\lfloor 2.66667 \rfloor = 2$.