Ugeopgave 3

Computerarkitektur

Kristian Gausel[1], Rasmus Skovdal[2], og Steffan C. S. Jørgensen[3]

[1]201509079, 201509079@post.au.dk
[2]201509421, rasmus.skovdal@post.au.dk
[3]201505832, 201505832@post.au.dk

1. maj 2016

1 Indledning

Denne opgave omhandler IJVM programmet *GEC.j*, der implementerer relationen $a \geqslant b$ ved brug af subtraktion. Kildekoden for *GEC.j* ses i kodeudsnit 1.

Kodeudsnit 1: Programmet *GEC.j*

```
1  .method main       // int main
2  .args  3           // ( int a, int b )
3  .define a = 1
4  .define b = 2
5
6  if:                // {
7      iload a        // if (a >= b)
8      iload b
9      isub
10     // stack = a - b, ... ; a - b < 0 => a < b
11
12     iflt else
13 then:
14     bipush 1       //   return 1;
15     goto endif
16 else:
17     bipush 0       //   return 0;
18 endif:
19     ireturn        // }
```

2 Spørgsmål A

Da *IJVM* repræsenterer heltal som 32-bit lange med andenkomplement, så kan alle tal i følgende interval repræsenteres.

$$[-2^{31}\ ,\ 2^{31}-1]$$

Programmet *GEC.j* vil have problemer med overløb hvis én af to nedenstående uligheder gør sig gældende:

$$a-b<-2^{31} \qquad \text{eller} \qquad a-b>2^{31}-1$$

Altså, hvis resultatet af $a-b$ på linje 9 i kodeudsnit 1 er under -2^{31} eller over $2^{31}-1$, så vil der ske et overløb. Eksempler på overløb ses i appendiks A.

3 Spørgsmål B

Under forudsætning af, at argumenterne a og b, der bliver givet som input til programmet, befinder sig inden for intervallet beskrevet under spørgsmål A er der kun risiko for overløb, hvis a og b har forskellige fortegn. Sammenhængen beskrives nedenfor i tabel 1.

$\pm$**a**	$\pm$**b**	**Mulighed for overløb**
$-a$	b	$a-b<-2^{31}$
a	$-b$	$a-b>2^{31}-1$
a	b	Ingen mulighed for overløb
$-a$	$-b$	Ingen mulighed for overløb

Tabel 1: Potentielle situationer for *overløb* ved brug af programmet *GEQ.j*

Man kan for disse *farlige* fortegnskombinationer, afgøre relationen $a \geqslant b$ ved at betragte fortegnene. Hvis a er positiv og b er negativ gælder det altid, at $a > b$, og omvendt hvis a er negativ og b er positiv, så er $a < b$.

4 Spørgsmål C

Hvis vi følger argumentationen fra spørgsmål B, kan programmet sikres mod faren for overløb ved at undersøge om en af de to *farlige* situationer er aktuelle. Koden for et program, der implementerer sådanne check, findes i kodeudsnit 2.

5 Spørgsmål D

Det overløbssikre program, hvis kildekode findes i kodeudsnit 2 er blevet afprøvet med de argumenter, der forårsagede fejl i den oprindelige kode for *GEC.j*. Resultatet af disse test finde i appendiks B.

Kodeudsnit 2: Programmet *GEC2.j*, der sikrer ingen overløb

```
1  .method main       // int main
2  .args  3           // ( int a, int b )
3  .define a = 1
4  .define b = 2
5
6  if: iload a
7      iflt alt       //If a < 0: goto alt
8      goto amt       //else: goto amt
9  alt:
10     iload b
11     iflt standard  //If a < 0 AND b < 0:
12                    //goto standard (subtraction)
13     bipush 0       //else (a < 0 AND b >= 0):
14     ireturn        //return 0 (false)
15 amt:
16     iload b
17     iflt amt_cont  //If a >= 0 AND b < 0:
18                    //goto amt_cont (returns 1)
19     goto standard  //else (a >= 0 AND b >= 0):
20                    //goto standard (subtraction)
21 amt_cont:
22     bipush 1       //return 1
23     ireturn
24 standard:
25     iload a
26     iload b
27     isub           //a - b
28     // stack = a - b, ... ; a - b < 0 => a < b
29     iflt else
30 then:
31     bipush 1       //   return 1;
32     goto endif
33 else:
34     bipush 0       //   return 0;
35 endif:
36     ireturn        // }
```

6 Spørgsmål E

Vi ved at de to input a og b befinder sig inden for det føromtalte interval. Dette medfører, at det højeste mulige tal vi kan få, når vi subtraherer er $a + |b|$. Pga. andenkomplementsopbygningen vil man aldrig ved addition/subtraktion kunne få tallene helt over på 0 igen ved et overløb, og det tætteste man ville kunne komme på er -1, fordi den øvre intervalgrænse netop er $2^{31} - 1$. Ud fra ovenstående, kan relationen $a = b$ implementeres som i kodeudsnit 3.

Kodeudsnit 3: Programmet *EQC_Naive.j*, der ikke tager hensyn til overløb

```
1  .method main      // int main
2  .args  3          // ( int a, int b )
3  .define a = 1
4  .define b = 2
5                    // {
6  if:     iload a   // if (a >= b)
7          iload b
8          isub
9          // stack = a - b, ... ; a - b < 0 => a < b
10         ifeq else
11 then:
12         bipush 0  //   return 0;
13         goto endif
14 else:
15         bipush 1  //   return 1;
16 endif:
17         ireturn   // }
```

Hvis det antages, at overløb ved subtraktion *kan* resultere i 0, kan en overløbssikker implementation af relationen $a = b$ foretages som i afsnit 4. Denne kode, der ses i kodeudsnit 4, er betragteligt mere omfattende og vil derfor også i de fleste tilfælde være langsommere, men derimod er problemer med overløb udelukket.

Kodeudsnit 4: Programmet *EQC.j*, der sikrer ingen overløb

```
1  .method main       // int main
2  .args  3           // ( int a, int b )
3  .define a = 1
4  .define b = 2
5
6  if: iload a
7      iflt alt       //If a < 0: goto alt
8      goto amt       //else: goto amt
9  alt:
10     iload b
11     iflt standard  //If a < 0 AND b < 0
12                    //goto standard (subtraction)
13     bipush 0       //else (meaning a < 0 AND b >= 0)
14     ireturn        //return 0 (false)
15 amt:
16     iload b
17     iflt amt_cont  //If a >= 0 AND b < 0:
18                    //goto amt_cont (returns 0)
19     goto standard  //else (meaning a >= 0 AND b >= 0)
20                    //goto standard (subtraction)
21 amt_cont:
22     bipush 0       //return 0
23     ireturn
24 standard:
25     iload a
26     iload b
27     isub           //a - b
28     // stack = a - b, ... ; a - b < 0 => a < b
29     ifeq else
30 then:
31     bipush 0       //return 0;
32     goto endif
33 else:
34     bipush 1       //return 1;
35 endif:
36     ireturn        // }
```

A Opgave A

Eksempler med passende argumenter for overløb.

Kodeudsnit 5: Output ved kørsel af *GEC.j* med et overløb i positiv retning

```
0 #ijvm GEC.bc a = 2147483647 b = -1
1                       stack = 0, 1, -1, 2147483647, 9
2 iload 1     [15 01]   stack = 2147483647, 0, 1, -1, 2147483647, 9
3 iload 2     [15 02]   stack = -1, 2147483647, 0, 1, -1, 2147483647, 9
4 isub        [64]      stack = -2147483648, 0, 1, -1, 2147483647, 9
5 iflt 8      [9b 00 08] stack = 0, 1, -1, 2147483647, 9
6 bipush 0    [10 00]   stack = 0, 0, 1, -1, 2147483647, 9
7 ireturn     [ac]      stack = 0
8 return value: 0
```

Kodeudsnit 6: Output ved kørsel af *GEC.j* med et overløb i negativ retning

```
0 #ijvm GEC.bc a = -2147483648 b = 1
1                       stack = 0, 1, 1, -2147483648, 9
2 iload 1     [15 01]   stack = -2147483648, 0, 1, 1, -2147483648, 9
3 iload 2     [15 02]   stack = 1, -2147483648, 0, 1, 1, -2147483648, 9
4 isub        [64]      stack = 2147483647, 0, 1, 1, -2147483648, 9
5 iflt 8      [9b 00 08] stack = 0, 1, 1, -2147483648, 9
6 bipush 1    [10 01]   stack = 1, 0, 1, 1, -2147483648, 9
7 goto 5      [a7 00 05] stack = 1, 0, 1, 1, -2147483648, 9
8 ireturn     [ac]      stack = 1
9 return value: 1
```

B Opgave D

Kodeudsnit 7: Output ved kørsel af *GEC.j* med et overløb i positiv retning

```
0 #ijvm GEC2.bc a = 2147483647 b = -1
1                       stack = 0, 1, -1, 2147483647, 17
2 iload 1     [15 01]   stack = 2147483647, 0, 1, -1, 2147483647, 17
3 iflt 6      [9b 00 06] stack = 0, 1, -1, 2147483647, 17
4 goto 13     [a7 00 0d] stack = 0, 1, -1, 2147483647, 17
5 iload 2     [15 02]   stack = -1, 0, 1, -1, 2147483647, 17
6 iflt 6      [9b 00 06] stack = 0, 1, -1, 2147483647, 17
7 bipush 1    [10 01]   stack = 1, 0, 1, -1, 2147483647, 17
8 goto 18     [a7 00 12] stack = 1, 0, 1, -1, 2147483647, 17
9 ireturn     [ac]      stack = 1
10 return value: 1
```

Kodeudsnit 8: Output ved kørsel af *GEC.j* med et overløb i negativ retning

```
0 #ijvm GEC2.bc a = -2147483648 b = 1
1                       stack = 0, 1, 1, -2147483648, 17
2 iload 1     [15 01]   stack = -2147483648, 0, 1, 1, -2147483648, 17
3 iflt 6      [9b 00 06] stack = 0, 1, 1, -2147483648, 17
4 iload 2     [15 02]   stack = 1, 0, 1, 1, -2147483648, 17
5 iflt 21     [9b 00 15] stack = 0, 1, 1, -2147483648, 17
6 bipush 0    [10 00]   stack = 0, 0, 1, 1, -2147483648, 17
7 goto 31     [a7 00 1f] stack = 0, 0, 1, 1, -2147483648, 17
8 ireturn     [ac]      stack = 0
9 return value: 0
```