dComArk: Aflevering 3
Håndtering af overløb

Christoffer Müller Madsen, 201506991
Jacob Hartmann, 20094613
Casper Vestergaard Kristensen, 201509411
Datalogi

29. november 2015

Spørgsmål A

IJVM-maskinen benytter 32-bit 2-komplement til at repræsentere heltal. To-komplement-systemet har for 32-bit heltal med fortegn følgende rækkevidde:

$$I_{32} = \left[-2^{31},\, 2^{31} - 1\right]$$

Grunden til at den positive del tilsyneladende er ét tal mindre end den negative del er at den positive del af sættet starter ved 0 og går opad mod $2^{31} - 1$, mens den negative del starter ved -1 og går til -2^{31}. Pladsen til tallet 0 er altså taget fra den positive del af sættet.

Grænserne for de værdier der kan repræsenteres ved brug af et 32-bit 2-komplement heltal kan medføre at der kan skabes uventede resultater ved uforsigtig brug af regneoperationer med tal af denne type. I programmet `geq.j` kan der som følge af implementeringen ske problemer med overløb i nogle tilfælde grundet argumenternes fortegn. Givet at de argumenter der gives til programmet holder sig indenfor grænserne af I_{32}, vil overløbsproblemer udelukkende opstå hvis de to argumenter ikke har samme fortegn. Programmet kan i disse tilfælde f.eks. risikere at trække et positivt tal fra tallet -2^{31}, hvorved der vil ske et overløb i den negative retning, således at resultatet af subtraktionen bliver betragtet som et positivt tal, hvilket ikke er det forventede resultat. Ligeledes kan der opstå problemer hvis der trækkes et negativt tal fra positive tal, da der i så fald kan ske overløb i den positive retning, eftersom man blandt andet kan komme ud for at negative tal trækkes fra—hvilket effektivt set svarer til at addere til—et

positivt tal der ligger lige på den øvre grænse af I_{32}. Konkrete eksempler på kørsler som disse kan ses på Figur 1.

a	b	Fortegn a	Fortegn b	Forventet resultat	Resultat `geq.j`	Korrekthed
$2^{31}-1$	$2^{31}-1$	$+$	$+$	1	1	✓
-2^{31}	-2^{31}	$-$	$-$	1	1	✓
-2^{31}	2	$-$	$+$	0	1	✗
$2^{31}-1$	-2	$+$	$-$	1	0	✗
-2	$2^{31}-1$	$-$	$+$	0	1	✗
2	-2^{31}	$+$	$-$	1	0	✗

Figur 1: Tabel over tests udført med det oprindelige program `geq.j`. De to første sæt af testværdier er valgt for at vise at det ikke er at problem at benytte tal der ligger i grænsen af hvad der kan repræsenteres indenfor I_{32}, så længe disse deler fortegn. De andre fire kombinationer er valgt for at vise at programmet giver et forkert resultat hvis fortegnene for a og b ikke er ens.

Spørgsmål B

De mulige sammensætninger af fortegn for de to programargumenter a og b spænder over fire forskellige kombinationer, der er vist i Figur 2. I tabellen er der en kolonne navngivet "*Overløbssikkerhed*", der indikerer hvorvidt de enkelte kombinationer af fortegn er sikre for overløb. De fortegnskombinationer der er markeret med et ✗ i denne kolonne, har potentiale for at frembringe overløb grundet brugen af `isub`-instruktionen. Derfor er det for disse kombinationer nødvendigt at benytte en sammenligningsmetode, der ikke har mulighed for at skabe overløb.

Fortegn a	Fortegn b	Overløbsikkerhed
$+$	$+$	✓
$+$	$-$	✗
$-$	$+$	✗
$-$	$-$	✓

Figur 2: Tabel over hvilke fortegnskombinationer for programargumenterne a og b, der kan skabe overløbsproblemer ved kørsel af programmet `geq.j` grundet programmets brug af `isub`-instruktionen.

Det ses på Figur 2, at de fortegnskombinationer hvor a og b ikke har samme

fortegn har risiko for at skabe overløb, hvilket også blev nævnt i forrige afsnit. Med en smule matematisk ræsonnement er det dog muligt at finde en metode til at afgøre relationen $a \geq b$ uden brug af subtraktion for netop disse to '*farlige*' fortegnssammensætninger.

Når a er positiv og b er negativ må det være således at a er større end b. Det omvendte må gælde for de tilfælde hvor b er negativ og a positiv. Dette kan opsummeres med følgende logiske udtryk:

$$\begin{cases} a > b & \text{for} \quad a \geq 0,\, b < 0 \\ b > a & \text{for} \quad a < 0,\, b \geq 0 \end{cases}$$

Dette udtryk kan omskrives således at det i endnu højere grad gøres klart hvorledes a og b er relateret til hinanden:

$$\begin{cases} a > b & \text{for} \quad b < 0 \leq a \\ b > a & \text{for} \quad a < 0 \leq b \end{cases}$$

Endvidere kan a ikke være lig b ved forskellige fortegn mellem de to værdier. Således er der taget højde for alle mulige tilfælde efter at disse relationer mellem a og b er blevet klargjort.

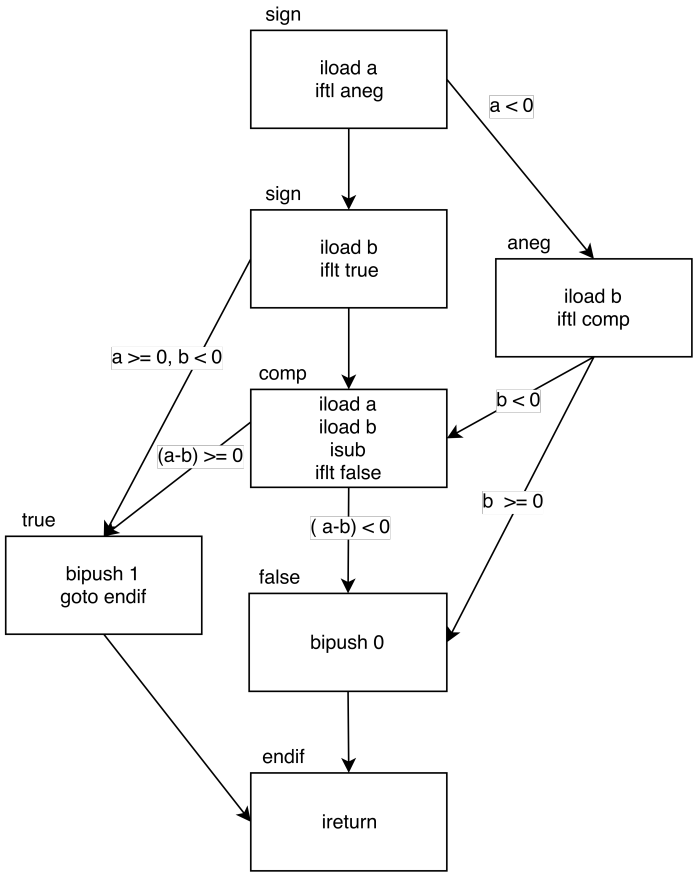
Spørgsmål C

Ved at benytte disse sammenhænge er det muligt at skrive en implementering af relationen $a \geq b$ som er sikret mod overløb. Den nye, overløbssikre implementering ses i Listing 1.[1]

[1] Programmet har med sine nye og forbedrede evner til håndtering af overløb samtidig erhvervet sig et nyt filnavn. Det kendes nu som `geqqo.j`.

Listing 1: geqqo.j

```
1  .method main
2  .args   3
3  .define a = 1
4  .define b = 2
5
6  sign:   iload a          // Check for negative signs
7          iflt aneg        // If a is negative, go check if b is too
8          iload b
9          iflt true        // If only b is negative, a > b
10                          // a > b is a subset of a >= b
11                          // return 1
12
13 comp:   iload a          // Compare the two values to check a >= b
14         iload b
15         isub
16         iflt false       // If (a-b) < 0, return 0, else return 1
17 true:   bipush 1         // Push 1 to top of stack and return
18         ireturn
19
20 aneg:   iload b          // Check if b is negative
21         iflt comp        // If a and b are negative go compare them
22                          // else a < b; a < b is not part of a >= b;
23                          // return 0
24
25 false:  bipush 0         // Push 0 to top of stack and return
26         ireturn
```



Figur 3: Blokdiagram over den nye og forbedrede kode. Hver af de horisontale forgreninger viser de steder i koden med *branch*-instruktioner. Den vertikale linje viser hvorledes programmet forløber hvis der ikke foretages nogen spring overhovedet.

I koden ved labelet `sign` tjekkes det først hvorvidt værdien af a, og dernæst værdien af b er negative. Hvis begge værdier er positive vil programmet fortsætte med udførsel af koden i label `comp`.[2] Her sammenlignes a og b ved brug af subtraktion, da der ikke er nogen fare for overløb hvis begge værdier har samme fortegn. Programmet er således funktionelt identisk med det oprindelige program `geq.j` hvis både a og b er positive.

Positive værdier af a Hvis a er positiv, fortsætter koden, ved at undersøge hvorvidt b er negativ. Hvis b viser sig at være negativ, må b være mindre end a jf. de logiske udtryk, der blev vist i forbindelse med spørgsmål B. Denne konklusion kan drages da a *skal* være positiv for at programmet kan nå til at eksekvere

[2]Der er ikke en eksplicit `goto` instruktion her eftersom koden blot fortsætter uden spring hvis den kravet for konditionelle branch-instruktioner som eksempelvis `iflt` ikke opfyldes.

fortegnstjekket for b. Derfor hopper programmet direkte til koden med label `true` hvor værdien 1 lægges på stakken og returneres.

Negative værdier af a Ved de tilfælde hvor a er negativ springer koden til `aneg`. Her undersøges det hvorvidt b er negativ. Hvis b viser sig at være at være positiv går koden videre til koden med label `false` hvor værdien 0 lægges på stakken og returneres. I tilfælde hvor b er negativ leder `aneg`-delen af programmet videre til label `comp` hvor a og b sammenlignes ved brug af subtraktion.

Gennem brugen af de branch-instruktioner der er indført for at undersøge hvorvidt a og b er negative er programmet altså blevet sikret mod overløb på grund af fortegnsforskelle.

Spørgsmål D

Overløbssikkerheden i det nye program kan verificeres ved at indsætte de samme værdier som fremprovokerede forkerte resultater fra det oprindelige program. Testværdierne er således valgt på samme baggrund som testværdierne i spørgsmål A. Resultaterne af de nye tests ses i Figur 4.

a	b	Fortegn a	Fortegn b	Forventet resultat	Resultat `geqqo.j`	Korrekthed
$2^{31}-1$	$2^{31}-1$	$+$	$+$	1	1	✓
-2^{31}	-2^{31}	$-$	$-$	1	1	✓
-2^{31}	2	$-$	$+$	0	0	✓
$2^{31}-1$	-2	$+$	$-$	1	1	✓
-2	$2^{31}-1$	$-$	$+$	0	0	✓
2	-2^{31}	$+$	$-$	1	1	✓

Figur 4: Tabel over tests udført med den udvidede udgave af programet `geqqo.j`. Testværdierne er magen til dem der blev brugt i de første tests med programmet `geq.j`. Det ses her at den udvidede udgave af programmet giver det korrekte resultat for alle testcases.

Som det fremgår af tabellen er den omskrevne udgave af programmet sikret mod overløb forårsaget af fortegnskombinationer hvor a og b havde modsatte fortegn.

Spørgsmål E

Relationen $a = b$ er forholdsvist simpel at implementere i IJVM. Én måde hvorpå dette kan gøres ses i Listing 2. Her er skrevet et program der tager to argumenter

og finder ud af hvorvidt disse er lig hinanden. Igen forudsættes det at begge programargumenter ligger indenfor grænserne af I_{32}.

Listing 2: eq.j

```
1  .method main
2  .args   3
3  .define a = 1
4  .define b = 2
5
6          iload a
7          iload b
8          isub            // Find the difference between a and b
9          ifeq true       // If a and b are equal, return 1
10 false:  bipush 0        // else: return 0
11         ireturn
12 true:   bipush 1
13         ireturn
```

I programmet hentes først de to værdier a og b. $a - b$ findes dernæst ved brug af `isub` og resultatet af dette evalueres hvorvidt resultatet af `isub`-instruktionen er lig nul ved brug af `ifeq`-instruktionen. Hvis resultatet af `isub`-instruktionen er lig nul, må de to tal være lig hinanden, eftersom resultatet af `isub` er differensen af de to tal, hvis a og b har samme fortegn. I de tilfælde hvor de to talværdier har forskellige fortegn, vil de uanset hvad ikke være lig hinanden, grundet fortegnsvariationen.

På grund af dette faktum er det ikke nødvendigt at tage højde for overløb, da eventuelle overløb ved brugen af de to usikre fortegnskombinationer resulterer i at programmet returnerer 0, dvs. at de to tal ikke er lig hinanden, fordi omstændigheden for den konditionelle branch-instruktion `ifeq`[3] ikke opfyldes. Dette er den forventede opførsel for to tal der ikke er lig hinanden, uanset om de to tal begge er positive, negative eller har forskellige fortegn. Således er overløb ikke et problem for relationen $a = b$, hvilket retfærdiggør den lettere simple, naive—og af disse grunde også effektive—implementering der ses i Listing 2.

[3]Det øverste element på stakken skal være lig nul for at instruktionen fører til et branch.