# dComArk: Aflevering 2
# Heltalsmultiplikation

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

20. november 2015

**Spørgsmål A** Hvis det $x$ der videregives til funktionen er negativ returnerer funktionen tallet 0, da `while`-løkken slet ikke køres hvis $x \leq 0$, hvorved variable $p$, der indeholder den værdi der returneres ved afslutning af programmet, ikke ændres fra sin oprindelige værdi 0. Således giver en negativ værdi af $x$ samme resultat som $x = 0$. For $x = 0$ er returværdien 0 dog korrekt, modsat returværdien for $x < 0$. Returværdien $p$ beholder sin oprindeligt satte værdi (0) hvis `while`-løkken ikke køres, hvilket medfører det misvisende resultat.

Resultatet er dog korrekt for negative $y$-værdier så længe $x$ er positiv. Proceduren for at udføre heltalsmultiplikation på den måde som det er gjort i `imul.c` er ens for positive og negative værdier af $y$, da programmet udfører addition af $y$ til $p$, $x$ antal gange. Hvis programmet køres med $x = 2$, $y = -4$ fås følgende output:

```
imul(2,-4) = -8
```

**Spørgsmål B** Hvis $x > 0$ har stakken den maksimale størrelse 11. Dette optræder på to tidspunkter:[1]

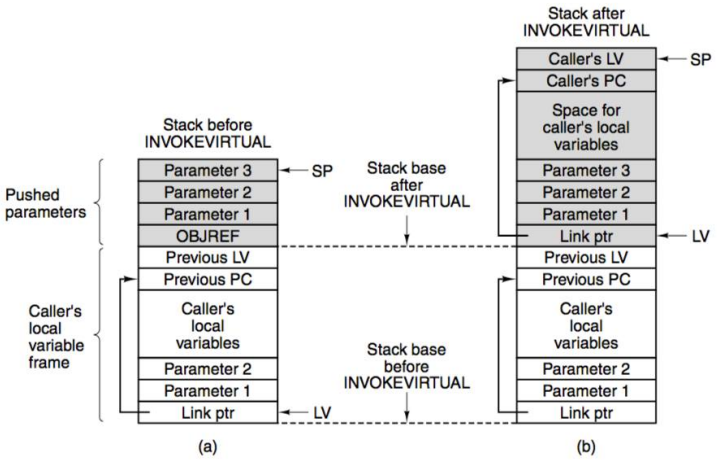
```
bipush 1      [10 01]    stack = 1, 2, 15, 51, 0, 3, 2, 22, 0, 1, 16

iload 2       [15 02]    stack = 3, 0, 15, 51, 0, 3, 1, 22, 0, 1, 16
```

Stakken er altså størst lige inden $x$-værdien reduceres med én under kørslen af den improviserede `while`-løkke samt lige efter indlæsningen af $y$-variablen, når

[1] Et fuldt *trace* af udførslen med parametre $x = 2$ og $y = 3$ er vedhæftet dokumentet.

summen af $p$ og $y$ skal findes inde i `while`-løkken. Stakkens maksimale størrelse er 11 hvis $x > 0$, hvilket medfører at `while`-løkken køres. Hvis $x \leq 0$ er stakkens maksimale størrelse 10, da `while`-løkken ikke evalueres i disse tilfælde. Denne afhængighed skyldes forudsætningen for `while`-løkken. Hvorvidt `while`-løkken køres, afhænger ikke af værdien $y$, hvilket gør at stakkens maksimale størrelse heller ikke afhænger af $y$.




Figur 1: (a) Hukommelsen før udførsel af `invokevirtual`. (b) Efter udførslen *Figuren er fra [Tannenbaum og Austin, 2012]*

```
stack=15,51,0,3,2,22,0,1,16
```

Figur 2: Udskrift af stakken umiddelbart efter udførslen af `invokevirtual imul`

**Spørgsmål C** Vedhæftet i slutningen af dette dokument er et manuelt opsat print af hukommelsen umiddelbart efter udførslen af `invokevirtual imul` i programmet. Sammenholdt med Figur 1 kan værdierne i hukommelsen tydes. Det ses på figuren at det næstøverste element på stakken vil være den tidligere værdi af `PC`. I metodeområdet af hukommelsen ses det at `invokevirtual`-ordren og dens dertilhørende operander (`opcode 0xB6, operands 00 00`) der kaldes i programmet ligger i adresserne 48-50.[2] Da `PC`-registret udpeger den byte i metodeområdet som skal afvikles ved den kommende cyklus, vil den tidligere værdi af `PC` derfor være 51. Instruktionen ved denne adresse i metodeområdet er den instruktion der skal eksekveres når programmet vender tilbage fra den indledte virtuelle metode. For dette program vil denne ordre være `ireturn`

[2]Metodeområdet adresseres i bytes, hvilket betyder at adresserne 48-50 her refererer til byte 48-50 i metodeområdet.

(`opcode 0xAC`), hvilket ses i `imul.j` ved at denne ordre står på linjen efter `invokevirtual imul`. Den formodede tidligere værdi af `PC` passer med hukommelsesprintet, hvor det næstøverste element på stakken (adresse word 22) har værdien 51. Ved adressen byte 51 i metodeområdet findes værdien `0xAC`, hvilket bekræfter at byte 51 indeholder opkoden til `ireturn`.

Adressen på dette element er samtidig forklaringen til betydningen af tallet 22. Link pointeren, hvis adresse indeholdes i `LV`-registret, peger nemlig på den adresse som indeholder den tidligere værdi af `PC`. Denne ligger som tidligere nævnt i adressen word 22. Tallet 51 peger altså til byte nr. 51 der indeholder instruktionen som skal udføres efter metoden kaldt i `invokevirtual` er færdig, og tallet 22 peger til word-adressen 22, hvori denne værdi ligger.

**Spørgsmål D** Eftersom der er 13 ordrer i koden for `while`-løkken samt 13 ordrer udover `while`-løkken bør antallet af ordrer kunne beskrives ved følgende udtryk:

$$\text{antal instruktioner} = 13 \cdot (x + 1) \quad \text{for} \quad x \geq 0$$

Programmet returnerer som tidligere nævnt forkerte resultater for $x < 0$. Dog kan antallet af ordrer stadig tælles. Den virtuelle imul-metode kalder `ireturn` tidligere hvis $x < 0$ end hvis $x = 0$, hvilket kan ses i følgende kodestykke fra `imul.j`:

```
1 iload x
2 iflt end_while
3 iload x
4 ifeq end_while
```

Hvor det bemærkes at der benyttes to ordrer mindre hvis $x < 0$ end hvis $x = 0$. Antallet af ordrer for tilfælde hvor $x < 0$ vil altså svare til

$$13 \cdot (0 + 1) - 2 = 11$$

så

$$\text{antal instruktioner} = 11 \quad \text{for} \quad x < 0$$

**Spørgsmål E** Ved at køre kommandoen

```
ijvm-asm imul.j | ijvm - | wc -l
```

tælles antallet af linjer i outputtet der fremkommer ved fortolkning af bytekoden. Antallet af udførte ordrer kan findes ved at trække 4 fra dette antal, da der er fire linjer i *trace*-outputtet, der ikke repræsenterer udførte ordrer.

Variationen i antallet af udførte ordrer ved ændringer i $x$ og $y$ kan således undersøges ved at ændre disse værdier i `lmul.j` og derefter køre ovenstående kommando og derefter betragte outputtet af denne. Resultaterne af de udførte tests ses i tabellen nedenfor, der beskriver værdierne af $x$, $y$ samt returværdien for forskellige kørsler. Værdierne er valgt for at afprøve særlige tilfælde hvor det kunne tænkes at antallet af ordrer kunne afvige fra det forslag på et generelt udtryk, der blev givet tidligere.

| $x$ | $y$ | Returværdi | Antal ordrer |
|---|---|---|---|
| -2 | 3 | 0 | 11 |
| -1 | 3 | 0 | 11 |
| -1 | -3 | 0 | 11 |
| 0 | 3 | 0 | 13 |
| 1 | 3 | 3 | 26 |
| 2 | 3 | 6 | 39 |
| 2 | -3 | -6 | 39 |
| 3 | 3 | 9 | 52 |
| 4 | 3 | 12 | 65 |

Figur 3: Tests af programmet for specielle værdier af $x$ og $y$

Denne tabel verificerer de formodede udtryk for antallet af ordrer. For at opsummere skrives disse igen her:

$$\text{antal instruktioner} = 13 \cdot (x + 1) \quad \text{for} \quad x \geq 0$$

$$\text{antal instruktioner} = 11 \quad \text{for} \quad x < 0$$

# Litteratur

[Tannenbaum og Austin, 2012] Tannenbaum, A. S. og Austin, T. (2012). *Structured Computer Organization*. Pearson Education, (International) 6. udgave.

# A Bilag

Som tidligere nævnt kan et trace af kørslen af `imul.bc` med $x = 2$ og $y = 3$ findes på næste side. Efter dette trace findes det tidligere nævnte manuelt opstillede print af hukommelsen umiddelbart efter kørslen af `invokevirtual imul`.

```
IJVM Trace of - Mon Nov 16 15:37:54 2015

                                stack = 0, 1, 16
bipush 44           [10 2c]     stack = 44, 0, 1, 16
bipush 2            [10 02]     stack = 2, 44, 0, 1, 16
bipush 3            [10 03]     stack = 3, 2, 44, 0, 1, 16
invokevirtual 0     [b6 00 00]  stack = 15, 51, 0, 3, 2, 22, 0, 1, 16
bipush 0            [10 00]     stack = 0, 15, 51, 0, 3, 2, 22, 0, 1, 16
istore 3            [36 03]     stack = 15, 51, 0, 3, 2, 22, 0, 1, 16
iload 1             [15 01]     stack = 2, 15, 51, 0, 3, 2, 22, 0, 1, 16
iflt 25             [9b 00 19]  stack = 15, 51, 0, 3, 2, 22, 0, 1, 16
iload 1             [15 01]     stack = 2, 15, 51, 0, 3, 2, 22, 0, 1, 16
ifeq 20             [99 00 14]  stack = 15, 51, 0, 3, 2, 22, 0, 1, 16
iload 1             [15 01]     stack = 2, 15, 51, 0, 3, 2, 22, 0, 1, 16
bipush 1            [10 01]     stack = 1, 2, 15, 51, 0, 3, 2, 22, 0, 1, 16
isub                [64]        stack = 1, 15, 51, 0, 3, 2, 22, 0, 1, 16
istore 1            [36 01]     stack = 15, 51, 0, 3, 1, 22, 0, 1, 16
iload 3             [15 03]     stack = 0, 15, 51, 0, 3, 1, 22, 0, 1, 16
iload 2             [15 02]     stack = 3, 0, 15, 51, 0, 3, 1, 22, 0, 1, 16
iadd                [60]        stack = 3, 15, 51, 0, 3, 1, 22, 0, 1, 16
istore 3            [36 03]     stack = 15, 51, 3, 3, 1, 22, 0, 1, 16
goto -24            [a7 ff e8]  stack = 15, 51, 3, 3, 1, 22, 0, 1, 16
iload 1             [15 01]     stack = 1, 15, 51, 3, 3, 1, 22, 0, 1, 16
iflt 25             [9b 00 19]  stack = 15, 51, 3, 3, 1, 22, 0, 1, 16
iload 1             [15 01]     stack = 1, 15, 51, 3, 3, 1, 22, 0, 1, 16
ifeq 20             [99 00 14]  stack = 15, 51, 3, 3, 1, 22, 0, 1, 16
iload 1             [15 01]     stack = 1, 15, 51, 3, 3, 1, 22, 0, 1, 16
bipush 1            [10 01]     stack = 1, 1, 15, 51, 3, 3, 1, 22, 0, 1, 16
isub                [64]        stack = 0, 15, 51, 3, 3, 1, 22, 0, 1, 16
istore 1            [36 01]     stack = 15, 51, 3, 3, 0, 22, 0, 1, 16
iload 3             [15 03]     stack = 3, 15, 51, 3, 3, 0, 22, 0, 1, 16
iload 2             [15 02]     stack = 3, 3, 15, 51, 3, 3, 0, 22, 0, 1, 16
iadd                [60]        stack = 6, 15, 51, 3, 3, 0, 22, 0, 1, 16
istore 3            [36 03]     stack = 15, 51, 6, 3, 0, 22, 0, 1, 16
goto -24            [a7 ff e8]  stack = 15, 51, 6, 3, 0, 22, 0, 1, 16
iload 1             [15 01]     stack = 0, 15, 51, 6, 3, 0, 22, 0, 1, 16
iflt 25             [9b 00 19]  stack = 15, 51, 6, 3, 0, 22, 0, 1, 16
iload 1             [15 01]     stack = 0, 15, 51, 6, 3, 0, 22, 0, 1, 16
ifeq 20             [99 00 14]  stack = 15, 51, 6, 3, 0, 22, 0, 1, 16
iload 3             [15 03]     stack = 6, 15, 51, 6, 3, 0, 22, 0, 1, 16
ireturn             [ac]        stack = 6, 0, 1, 16
ireturn             [ac]        stack = 6
return value: 6
```

```
Byte|   Word|   Value
92  |   23  |   00 00 00 15
88  |   22  |   00 00 00 51 // Previous PC
84  |   21  |   00 00 00 00
80  |   20  |   00 00 00 03
76  |   19  |   00 00 00 02
72  |   18  |   00 00 00 22
68  |   17  |   00 00 00 00
64  |   16  |   00 00 00 01
60  |   15  |   00 00 00 16 // BEGIN Stack
56  |   14  |   00 00 00 26 // Reference to main label
52  |   13  |   00 00 00 00 // BEGIN constant pool - imul label
48  |   12  |   b6 00 00 ac // END Method area
44  |   11  |   10 02 10 03
40  |   10  |   00 00 10 2c
36  |   9   |   03 ac 00 01
32  |   8   |   a7 ff e8 15
28  |   7   |   02 60 36 03
24  |   6   |   01 15 03 15
20  |   5   |   10 01 64 36
16  |   4   |   00 14 15 01
12  |   3   |   19 15 01 99
8   |   2   |   15 01 9b 00
4   |   1   |   10 00 36 03
0   |   0   |   00 03 00 01 // BEGIN Method area
```