dComArk: Aflevering 4
Udvidelse af IJVM

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

8. december 2015

Mic1-mikroassembleren benytter sig af kode i sproget *Micro-Assembly Language* (MAL) til at beskrive mikroprogrammer. I filen `ijvm.mal` findes et mikroprogram, der fortolker instruktioner fra instruktionssættet IJVM. Det understøttede instruktionssæt kan udvides ved at tilføje yderligere ordrer bestående af mikrokode i sproget MAL og dernæst samle filen igen med Mic1 mikroassembleren, hvorefter programmer eksempelvis kan køres ved brug af Mic1-simulatoren. For at Mic1-simulatoren genkender de nye ordrer, skal disse ordres mnemotekniske navne og opkoder tilføjes til filen `ijvm.spec`, der specificerer de ordrer der er tilgængelige for maskinen. Proceduren for at tilføje ordrer til IJVM er således:

- Skriv for de ønskede ordrer den nødvendige mikrokode.
- Tilføj koden til `ijvm.mal`.
- Tilføj opkoden for de nye ordrer til `ijvm.spec`.
- Kør Mic1-mikroassembleren på den nye `ijvm.mal`-fil for at få bygget et nyt control store til Mic1-simulatoren indeholdende de nye ordrer.

Beskrivelse af funktion	Mnemoteknisk navn	Opkode
Vestreskift	`ishl`	`0x78`
Aritmetisk højreskift	`ishr`	`0x7A`
Logisk højreskift	`iushr`	`0x7C`

Tabel 1: Tabel over de ordrer der ønskes implementeret, deres mnemotekniske navne samt deres opkoder.

`ijvm.spec` udvides med en tekststreng på formen "*opkode mnemoteknisk navn*". For ordren ishl vil udvidelsen til ijvm.spec således være: `0x78 ishl`

Venstreskift - `ishl`

Et venstreskift på én bit rykker alle bits i en sekvens én plads til venstre med undtagelse af den bit, der er yderst til venstre i sekvensen. Denne bit bliver fjernet fra sekvensen, mens der fra højre indtræder et 0. Det aritmetiske venstreskift er således identisk med det logiske venstreskift, da der ikke tages hensyn til hverken den mest eller mindst betydende bit. Venstreskiftshandlingen svarer således til en fordobling af tallet jf. Tabel 2, hvori dette vises ved at betragte venstreskiftsoperationen på 4-bit tal.

Bit-position:	3	2	1	0	
	0	1	1	0	6
$<< 1$	1	1	0	0	12

	3	2	1	0	
	1	1	0	0	12
+	1	1	0	0	12
=	1	0	0	0	8

Tabel 2: Til venstre ses et venstreskift af et tal. Det ses, at dette er ækvivalent med en fordobling af tallet. På højre side ses en tabel, hvor de to øverste rækkers tal lægges sammen. Dette resulterer i et overløb, men den resulterende bitstreng er identisk med den bitstreng, der ville komme ud af et venstreskift. Det at fordoble et tal svarer altså, på trods af overløb, til et venstreskift. Det er dog bemærkelsesværdigt, at decimalrepræsentationen i overløbssituationer ikke vil stemme overens med den forsøgte fordobling. Dette er imidlertid den adfærd, der forventes.

Til udvidelsen af IJVM med denne instruktion implementeret på Mic1, ønskes en ordre der skifter det næstøverste element på stakken et antal gange til venstre, som er specificeret i de fem mindst betydende bits i ordet på toppen af stakken. Den foretagne implementering benytter at et venstreskift af en bitstreng svarer til en fordobling af det tal hvis bitstreng skiftes til venstre. Implementeringen af ordren `ishl` kan ses i Listing 1. En redegørelse for og forklaring af koden findes under figuren.

Listing 1: `ishl`

```
1  ishl = 0x78:
2                          # It is possible to use the value of
3                          #  the MBRU register to create the mask.
4      H = MBRU >> 1       # MBRU = 1111000, H = 111100
5      H = H >> 1          # H = 11110
6      H = H + 1           # H = 11111
7                          # Read second-to-top word from stack and
8                          #  set SP to point to this position.
9      MAR = SP = SP - 1; rd
10     TOS = TOS AND H     # Apply bit mask to the top word from
11                         #  the stack which signifies the number
12                         #  of shifts to perform.
13     H = OPC = MDR       # Load the value to be shifted.
14
15 ishl_loop:              # If the counter (TOS) is 0, end loop.
16     Z = TOS; if (Z) goto ishl_end; else goto ishl_shift
17 ishl_shift:
18     H = OPC = OPC + H   # Perform an arithmetic left-shift by
19                         #  multiplying H and OPC. H and OPC is
20                         #  loaded with the result to prepare for
21                         #  possible further multiplications.
22
23     TOS = TOS - 1       # Decrease the counter by one.
24     goto ishl_loop
25 ishl_end:               # Customary push of result to the TOS
26                         #  register and the stack in memory.
27                         # MAR = SP is not necessary, due to
28                         #  MAR already being set to SP earlier.
29     MDR = TOS = H; wr; goto main
```

Først skabes en bit-maske[1] til at isolere de fem mindst betydende bits i det næstøverste ord på stakken, da det ønskes, at der kun skal være mulighed for at lave 31 venstreskift ad gangen, da bredden af sekvensen, der skal skiftes er 32-bit, hvilket vil gøre alt over 31 skift redundant, da bitstrengen i så fald vil være bestående af udelukkende 1- eller 0-værdier. Da kun de fem mindst betydende bits er ønsket, benyttes bitmasken `11111`[2]. Denne bitmaske kan produceres udelukkende ved brug regneoperationer fra ALU'en. I denne implementering benyttes dog værdien af registret `MBRU`, der indeholder opkoden (uden fortegn) for den senest udførte instruktion, da denne værdi her er `0x78`, hvilket svarer til det binære tal `1111000`. Ved brug af to højreskift og dernæst en forøgelse

[1] En bit-maske kan bruges som et af de to argumenter til operationen `AND` for at isolere et bestemt sæt af positioner af bits i en bitsekvens.

[2] IJVM benytter sig af 32-bit (4-byte) ord. Derfor er den fulde repræsentation af bitmasken `00000000 00000000 00000000 00011111`. Som det er sædvanligt vil det være implicit at der fyldes op med 0'er fra venstre når en bitsekvens ikke fylder hele 32-bit størrelsen.

med 1, er dette tal omformet til tallet `11111`, der er den ønskede bitmaske. Der foretages en læsning af det næstøverste element på stakken og bitmasken påføres det øverste element på stakken, der er beliggende i `TOS`-registret og derfor ikke forudsætter en læsning fra hukommelsen.

Efter påføringen af bitmasken er læsningen fra hukommelsen færdig. Værdien af denne læses ind i registrene `H` og `TOS` for at forberede den forekommende fordobling. Nu træder maskinen ind i en løkke. Inde i løkken adderes værdierne i `OPC` og `H`, hvorefter summen læses ind i både `OPC` og `H` igen. Da værdierne i `H` og `OPC` er ens, svarer dette til en fordobling af det tal, der ligger i dem begge. Værdien af `TOS`-registret, der indeholder antallet af gange der skal skiftes til venstre, reduceres herefter med én. Denne løkke fortsætter indtil værdien af `TOS` er lig nul, hvorefter det venstreskiftede tal, der er gemt i `H`, skrives tilbage til toppen af stakken i hukommelsen, samt til `TOS`-registret. Efter dette er ordren afsluttet. De to argumenter er fjernet fra stakken og på toppen af stakken ligger nu det venstreskiftede tal.

Aritmetisk højreskift - `ishr`

Et aritmetisk højreskift med en størrelse på 1 bit er et skift af bits én plads til højre, hvor den mest betydende bits værdi beholdes ved at kopiere værdien af denne til den nye mest betydende bit, hvorved fortegnet for det tal der skiftes til højre beholdes. Den bit der løber ud over højre side bliver dog stadig kastet væk.

Det aritmetiske højreskift er allerede implementeret i IJVM for et skift på 1 bit. Det kan udvides ved at indsætte det i en simpel løkke på samme måde som det blev gjort for at udføre venstreskiftet flere gange. Koden til implementeringen af `ishr`-ordren på Mic1 ses nedenfor i Listing 2.

Listing 2: `ishr`

```
1  ishr = 0x7A:
2                          # It is possible to use the value of
3                          #  the MBRU register to create the mask.
4      H = MBRU >> 1       # MBRU = 1111010, H = 111101
5      H = H >> 1          # H = 11110
6      H = H + 1           # H = 11111
7                          # Read second-to-top word from stack and
8                          #  set SP to point to this position.
9      MAR = SP = SP - 1; rd
10     TOS = TOS AND H     # Apply bit mask to the top word from
11                         #  the stack.
12
13     H = MDR             # Load the value to be shifted into H.
14
15 ishr_loop:              # If the counter (TOS) is 0, end loop.
16     Z = TOS; if (Z) goto ishr_end; else goto ishr_shift
17 ishr_shift:             # The actual shifting takes place here
18     H = H >> 1          # The value is shifted arithmetically
19                         #  right by 1-bit using the ALU.
20     TOS = TOS - 1       # The counter is decreased.
21     goto ishr_loop
22 ishr_end:               # Customary push of result to the TOS
23                         #  register and the stack in memory.
24     MDR = TOS = H; wr; goto main
```

Som ved implementeringen af venstreskiftet skabes først en bitmaske til at isolere de fem mindst betydende bits. Dette gøres igen ud fra værdien af `MBRU`, hvor værdien denne gang er `1111010`. Bitsekvensen `11111`, kan altså igen skabes ved at skifte to gange til højre og lægge én til. Herefter følges samme procedure som ved implementeringen af venstreskiftet. Bitmasken påføres og det næstøverste ord på stakken læses ind i registret `H`. Igen trædes der ind i en løkke, der afsluttes når værdien af `TOS`-registret er nul. Denne gang udføres der dog et aritmetisk højreskift i løkken ved brug af den shifter, der er til stede i Mic1'en. Løkken endes når alle skift er udført, hvorefter den skiftede værdi lægges øverst på stakken.

Logisk højreskift - `iushr`

Det logiske højreskift adskiller sig fra det aritmetiske højreskift, da det logiske højreskift ikke tager hensyn til den mest betydende bit, som blot bliver et `0`. Med det logiske højreskift betragtes den skiftede værdi altså mere som en bit*sekvens* end et tal. Ved et logisk højreskift på 1 bit flyttes alle bits én plads til højre, hvor den nye mest betydende bit, der kommer ind fra venstre *altid* er et nul og

den bit, der løber ud over højresiden kastes væk.

Den udarbejdede implementering udfører det egentlige logiske højreskift ved at benytte sig af det aritmetiske højreskift i Mic1'en samt en bitmaske til at sætte den mest betydende bit til at være nul. I Listing 3 ses koden, der udgør implementeringen af `ishr`-ordren for Mic1'en.

Som i de to tidligere eksempler skabes bitmasken `11111` ved brug af værdien af `MBRU`. Denne gang kræver det kun to højreskift for at skabe den bitmaske, der påføres værdien i `TOS`-registret, som indeholder antallet af skift. Denne gang er den efterfølgende procedure dog anderledes. Efter påføringen af den første bitmaske skabes der ny en bitmaske til fjernelsen af den mest betydende bit. Den ønskede bitmaske har derfor den binære værdi `01111111 11111111 11111111 11111111`. Værdien skabes på følgende vis:

Der startes ud med værdien `11111` fra den gamle bitmaske. Denne forøges denne med én så værdien bliver `100000`. Værdien fordobles to gange så værdien er `10000000`. Denne værdi udsættes for tre logiske 8-bit venstreskift ved brug af Mic1'ens indbyggede skifter. Dette giver værdien `10000000 00000000 00000000 00000000`. Værdien, der opbevares i `OPC`, inverteres bitvist hvorved den ønskede bitmaske opnås.[3] Efterfølgende indlæses `H` med den værdi, der skal skiftes til højre.

Maskinen træder nu ind i en løkke ligesom i de andre to implementeringer, hvor løkken fortsættes indtil der er skiftet det ønskede antal gange, hvilket er sket når `TOS`-registret har værdien nul. Inde i løkken udsættes værdien, der skal skiftes logisk til højre, for et *aritmetisk* højreskift ved brug af Mic1'ens skifter, hvorefter den skiftede værdi påføres den tidligere nævnte bitmaske gennem en `AND`-operation med de to værdier som argumenter. Dette sætter den mest betydende bit til `0`. Således omdannes det aritmetiske skift til et logisk skift ved brug af bitmasken. Løkken udføres indtil det ønskede antal logiske højreskift er påført startværdien, hvorefter resultatet af disse skift lægges på toppen af stakken og indlæses i `TOS`-registret.

[3] En fratrækning af 1 fra værdien vil også være en mulighed, da resultatet af dette her er ækvivalent med en invertering.

Listing 3: `iushr`

```
1  iushr = 0x7C:
2      H = MBRU >> 1          # MBRU = 1111100, H = 111110
3                             # Set H to the value of OPC to prepare
4                             # for the creation of a new bit-mask
5                             # later in the instruction.
6      OPC = H = H >> 1       # H = 11111
7                             # Read second-to-top word from stack and
8                             #  set SP to point to this position.
9      MAR = SP = SP - 1; rd
10     TOS = TOS AND H        # Apply bit mask to the top word from
11                            #  the stack.
12                            # Create new bit mask from the previous
13                            #  bit mask.
14     H = OPC = OPC + 1      # H = 100000
15     H = OPC = H + OPC      # H, OPC = 01000000
16                            # The value of OPC after the following
17                            #  instructions is listed in the
18                            #  comments of each line:
19     OPC = H + OPC          # 00000000 00000000 00000000 10000000
20     OPC = OPC << 8         # 00000000 00000000 10000000 00000000
21     OPC = OPC << 8         # 00000000 10000000 00000000 00000000
22     OPC = OPC << 8         # 10000000 00000000 00000000 00000000
23     OPC = inv(OPC)         # 01111111 11111111 11111111 11111111
24                            # If anything is AND with OPC, it will
25                            #  remove the most significant bit of
26                            #  that value with OPC.
27
28     H = MDR                # Load H with the value to be shifted
29 iushr_loop:
30     Z = TOS; if (Z) goto iushr_end; else goto iushr_shift
31 iushr_shift:
32     H = H >> 1             # Do a 1-bit arithmetic right-shift
33     H = H AND OPC          # Use the bit-mask to remove the most
34                            #  significant bit, effectively making
35                            #  the arithmetic right-shift into a
36                            #  logical right-shift.
37     TOS = TOS - 1          # Decrease the counter by one.
38     goto iushr_loop
39 iushr_end:                 # Customary push of result to the TOS
40                            #  register and the stack in memory.
41     MDR = TOS = H; wr; goto main
```

Test af implementeringerne

Inden de færdige implementeringer kan afprøves er det nødvendigt at samle dem ved brug af mikroassembleren `mic1-asm`. Mikrokoden til de tre ordrer placeres i filen `ijvm_ext.mal` sammen med det allerede eksisterende IJVM-instruktionssæt fra `ijvm.mal` og filen samles med assembleren `mic1-asm`. Ordrerne tilføjes til `ijvm.spec` så fortolkeren kan genkende de nye ordrer. Hvis Mic1-simulatoren køres med det nyligt oprettede *control store*—i form af den samlede fil `ijvm_ext.mic1`—kan de nye ordrer bruges på lige fod med standardsættet af ordrer.

Ordrerne testes ved brug af IJVM-programmerne `test_ishl.j`, `test_ishr.j`, `test_iushr.j`, der alle tre er inkluderet sammen med dette dokument. Alle tre programmer følger samme struktur, hvilket medfører at koden for `test_ishl.j`, der er vist i Listing 4, således er repræsentativ for de to andre programmer, hvor `ishl`-ordren blot er erstattet af `ishr` og `iushr`. Ordrerne testes ved at køre de tre programmer med forskellige argumenter. Testene er valgt for at vise hvorledes ordrerne opfører sig i tilfælde, hvor der er mulighed for returnering af forkerte værdier hvis implementeringen ikke er korrekt udført.

Listing 4: `test_ishl.j`

```
1 .method main
2 .args   3
3 .define a = 1
4 .define b = 2
5     iload a
6     iload b
7     ishl
8     ireturn
```

Venstreskift - `ishl`

Ordren der udfører et venstreskift er afprøvet med de tests der er vist i Tabel 3. Det bemærkes som tidligere vist, at venstreskiftet svarer til en fordobling af tallet der skiftes når der ikke tages hensyn til overflow. Ved overflow vil venstreskiftet stadigvæk udføres, men tallet der kommer ud som resultat af skiftet vil ikke svare til en fordobling af det oprindelige tal.

	Decimal repræsentation	Binær repræsentation	Antal skift
1.	0	00000000 00000000 00000000 00000000	0/1
	0	00000000 00000000 00000000 00000000	—
2.	16	00000000 00000000 00000000 00010000	2
	64	00000000 00000000 00000000 01000000	—
3.	-16	11111111 11111111 11111111 11110000	2
	-64	11111111 11111111 11111111 11000000	—
4.	-2147483648	10000000 00000000 00000000 00000000	1
	0	00000000 00000000 00000000 00000000	—
5.	-1	11111111 11111111 11111111 11111111	1
	-2	11111111 11111111 11111111 11111110	—
6.	8	00000000 00000000 00000000 00001000	34 (2)
	32	00000000 00000000 00000000 00100000	—

Tabel 3: Tests udført med ordren `ishl`. For hver test er der vist både en decimal og binær repræsentation af det tal der skal skiftes samt tallet efter at det er skiftet det viste antal gange. Hver test er tildelt et nummer som står yderst til venstre. Den første test er udført for både nul og ét skift. De to tests er samlet i én test da de gav samme resultat og havde samme formål.

Den første test udføres for at vise at implementeringen ikke indfører uhensigtsmæssige tal ved skiftet, hverken når det fortælles at der skal skiftes nul eller én gang. Anden og tredje test er valgt for at vise at skiftet virker som forventet for både positive og negative tal. Fjerde test har til hensigt at vise at venstreskiftet også virker når der skabes overløb. Den decimale repræsentation af tallet stemmer dog ikke overens med en fordobling af tallet grundet overløbet, men dette er forventet adfærd i overløbssituationer. Femte test viser at venstreskiftet også virker efter hensigten når alle bits i ordet er 1. Den sjette test viser, at der kun tages hensyn til de fem mindst betydende bits i det ord, der fortæller hvor mange gange det oprindelige ord skal skiftes.

Alle tests har givet de forventede resultat, hvilket viser at implementeringen virker efter hensigten.

Aritmetisk højreskift - `ishr`

Ordren der udfører et aritmetisk højreskift er afprøvet med de tests der er vist i Tabel 4.

	Decimal repræsentation	Binær repræsentation	Antal skift
1.	0	00000000 00000000 00000000 00000000	0/1
	0	00000000 00000000 00000000 00000000	—
2.	16	00000000 00000000 00000000 00010000	2
	4	00000000 00000000 00000000 00000100	—
3.	-16	11111111 11111111 11111111 11110000	2
	-4	11111111 11111111 11111111 11111100	—
4.	1	00000000 00000000 00000000 00000001	1
	0	00000000 00000000 00000000 00000000	—
5.	-2147483648	10000000 00000000 00000000 00000000	31
	-1	11111111 11111111 11111111 11111111	—
6.	8	00000000 00000000 00000000 00001000	34 (2)
	2	00000000 00000000 00000000 00000010	—
7.	17	00000000 00000000 00000000 00010001	2
	4	00000000 00000000 00000000 00000100	—
8.	-17	11111111 11111111 11111111 11101111	2
	-5	11111111 11111111 11111111 11111011	—

Tabel 4: Tabel over tests udført med ordren `ishr`.

De første tre tests' formål er identiske med dem af de første fire tests for `ishl`. Den fjerde test er skabt for at vise at bits kan blive skubbet ud over højresiden af bitstrengen under skiftet. Femte test viser at ordren `ishr` sørger for at lave fortegnsforlængelse for at holde sig i overensstemmelse med definitionen af det aritmetiske højreskift. Sjette test viser ligesom sjette test for `ishl`, at de fem mindst betydende bits i antallet af skift isoleres korrekt. Syvende og ottende test viser at det aritmetiske højreskift virker for tal der ikke går op i to, når disse betragtes som en bitstreng. For tal der går op i potenser af to gælder det, at disse halveres ved brug af det aritmetiske højreskift.

Samtlige otte tests gav det forventede resultat, hvorledes det kan konkluderes at implementeringen af ordren opfører sig korrekt.

Logisk højreskift - `iushr`

Ordren der udfører et logisk højreskift er afprøvet med de tests der er vist i Tabel 5.

	Decimal repræsentation	Binær repræsentation	Antal skift
1.	0	00000000 00000000 00000000 00000000	0/1
	0	00000000 00000000 00000000 00000000	—
2.	16	00000000 00000000 00000000 00010000	2
	4	00000000 00000000 00000000 00000100	—
3.	-16	11111111 11111111 11111111 11110000	2
	1073741820	00111111 11111111 11111111 11111100	—
4.	1	00000000 00000000 00000000 00000001	1
	0	00000000 00000000 00000000 00000000	—
5.	8	00000000 00000000 00000000 00001000	34 (2)
	2	00000000 00000000 00000000 00000010	—

Tabel 5: Tabel over tests udført med ordren `iushr`.

Formålene for tests 1 til 4 er ens med de tests der har samme numre fra testene af `ishr`. Det er bemærkelsesværdigt at det logiske skift i den tredje test *ikke* laver fortegnsforlængelse hvorved det negative tal laves til et positivt tal. Dette er den forventede adfærd for det logiske højreskift. Den femte test viser, at de fem mindst betydende bits af ordet, der viser antallet af skift, udvindes på korrekt vis.

Igen har alle tests haft det forventede resultat og implementeringen af `iushr`-ordren betragtes på baggrund af disse som korrekt udført.

Konklusion

Alle tre ordrer blev indført i mikroprogrammet til Mic1-maskinen ved brug af mikroinstruktionssproget MAL. Disse ordrer blev testet og det viste sig, at alle tre ordrer gennemførte de fremlagte tests uden forekomster af uventede resultater.