Computer Arkitektur 2016 (CART'16): Test

10. Maj 2016

Skriv dit studienummer på alle sider. Hvis du vedlægger yderligere sider til besvarelsen, hvilket burde være unødvendigt, skal du skrive hvor mange du har vedlagt og skrive dit studienummer på dem også.
Dele af besvarelsen skal afleveres elektronisk på udleveret USB-stick. Filerne skal anbringes i en mappe med studienummer som navn, og navngives som angivet. USB-stick'en skal placeres i den udleverede kuvert. Der skal skrives navn og studienummer på kuverten.

Denne opgave er delt op i 7 dele, der i alt giver 100 points.

1 Repræsentation og manipulation af information

Opgave 1: Antag integers repræsenteret i 8 bits.

1. **(8 pts)** Udfyld denne tabel, således at rækkerne har den samme 8-bit integer i de forskellige repræsentationer.

Binær	Hexadecimal	Unsigned	Signed
01001101			
	0xA1		
		254	
			-17

2. **(5 pts)** Lad $x = 0$xA8 hexadecimalt og $y = 24$ decimalt, unsigned. Udfyld denne tabel af bitvise udtryk med svar i hexadecimalt:

Udtryk	Svar
`x & y`	
`x | y`	
`x ^ y`	
`~x | ~y`	
`y << 2`	

Opgave 2: Antag Signed/Unsigned Integers repræsenteret i 32 bits.

1. **(8 pts)** Antag at følgende er erklæret i et C-program:

```
unsigned int x1, x2;
int y1, y2;
```

Hvilke af disse udsagn er altid sande? For de falske udsagn: giv værdier til variablerne der modbeviser udsagnet (brug gerne konstanterne `INT_MAX`, `INT_MIN`, `UINT_MAX`, `UINT_MIN`, eller potenser af 2).

Udsagn	Sandt?	Modeksempel?
`x1 >= 0`		
`y1 >= 0`		
Hvis `y1 < 0` og `x1 = INT_MAX` så vil `y1 > x1`		
Hvis `y1 < 0` og `x1 = y1` udføres, så vil `x1 > 0`		
Hvis `y1 < 0` og `y2 = -y1` udføres, så vil `y2 > 0`		
Hvis `y1 < 0` og `y2 < 0`, så vil `y1 + y2 < 0`		

2 Assembly programmer

Opgave 3:

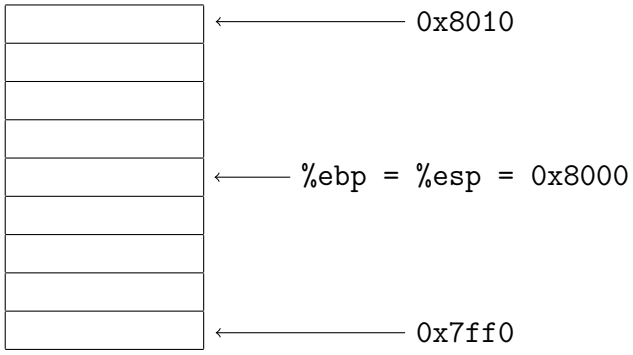
1. **(6 pts)** Forbind disse x86 assembly instruktioner (GNU syntax) med deres ækvivalente C-udtryk (antag `eax` eksisterer som variabel).

Assembly		C-udtryk	
`mov 0x42, %eax`	•	•	eax = eax * 4 + 66;
`add 0x42, %eax`	•	•	eax = eax + 42;
`lea 42(%eax), %eax`	•	•	eax = eax + 0x42;
`lea 0x42(,%eax, 4), %eax`	•	•	eax = eax * 5 + 66;
`lea 0x42(%eax, %eax, 4), %eax`	•	•	eax = 66;

2. **(6 pts)** Antag at `%ebp = %esp = 0x8000`. Hvilke værdier er skrevet på stacken efter følgende assembly-instruktioner er blevet udført?

```
push 0x42
sub 12, %esp
mov 0xA0, 0(%esp)
pop %eax
mov %eax, 4(%esp)
```

Udfyld denne figur af stacken med værdierne:



Opgave 4: x86 instruktionen for multiplikation, `mul`, og for division, `div`, undgås ofte af compileren.

1. **(1 pts)** Hvorfor genererer compileren ofte multiplikation og division uden at bruge `mul` og `div` instruktionerne?

2. **(5 pts)**
   Omskriv disse beregninger til ikke at bruge `mul` og `div` instruktionerne:

   mul \$2 , %**eax**

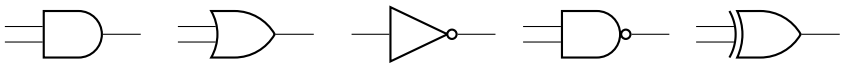
   mul \$3 , %**eax**

   mul \$5 , %**eax**

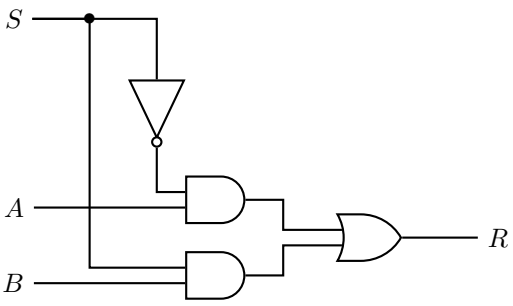
   mul \$16 , %**eax**

   div \$2 , %**eax**

3 Digital Logik



Figur 1: *And*, *Or*, *Not*, *Nand* og *Xor* gates.



Figur 2: Et digitalt logisk kredsløb.

Opgave 5: Figur 1 viser nogle logiske gates. Figur 2 viser et digitalt logisk kredsløb.

1. **(4 pts)** Skriv formlen for output værdien R for kredsløbet i Figur 2 som et C-udtryk:

2. **(4 pts)** Tegn et ækvivalent kredsløb, men uden at bruge *And*-gates. Brug de gates der er i Figur 1.

4 Y86 processor arkitektur

Stage	`popl rA`
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
Decode	valA ← R[%esp] valB ← R[%esp]
Execute	valE ← valB + 4
Memory	valM ← M_4[valA]
Write back	R[%esp] ← valE R[rA] ← valM
PC update	PC ← valP

Tabel 1: Beregninger i de forskellige stages for instruktionen `popl rA`.

Opgave 6: I denne opgave antager vi processor arkitekturen Y86, som beskrevet i lærebogen.

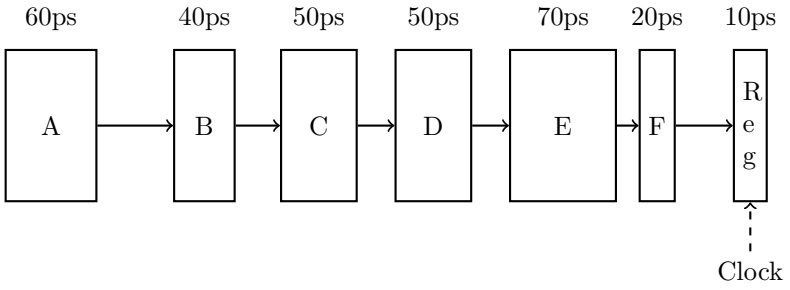
1. **(10 pts)** Antag at vi vil tilføje x86 instruktionen `leave`, der kan bruges til at forberede stacken før en funktion returnerer. Den har samme effekt som disse Y86 instruktioner:

   ```
   rrmovl %ebp, %esp
   popl %ebp
   ```

   Vi kan bruge 1-byte encodingen `0xD0`, som instruktions encoding for `leave`.
   Udfyld denne tabel med de nødvendige beregninger (brug Tabel 1 som inspiration):

Stage	`leave`
Fetch	icode:ifun ← rA:rB ← valP ←
Decode	valA ← valB ←
Execute	valE ←
Memory	valM ←
Write back	R[ ] ← R[ ] ←
PC update	PC ←

5 Pipelines



Figur 3: Et digitalt kredsløb delt op i 6 funktionelle dele, med delays i pico-sekunder for de enkelte dele.

Opgave 7:

1. **(3 pts)** Hvad er latency for det digitale kredsløb i Figur 3?

   Hvad er throughput for det digitale kredsløb i Figur 3?

2. **(5 pts)** Optimer throughput for det digitale kredsløb i Figur 3, ved at indsætte højst 2 hardware registre mellem de funktionelle dele. Antag at hardware registre har delay på 10ps.
   Indiker med pile hvor du vil indsætte hardware registre:

   A B C D E F

   Hvad bliver den resulterende throughput?

   Hvad bliver latency reduceret til?

6 Caching

Opgave 8: Antag et system med 8-bit adresser, og en 2-Way Set Associative Cache med 4 cache sets, og med cache block size på 8 bytes.

1. **(2 pts)** Hvor mange bytes kan cachen cache?

2. **(8 pts)** Antag at følgende adresser bliver tilgået, startende fra en tom cache:
   1. 0x64
   2. 0x70
   3. 0x00
   4. 0x01
   5. 0x78
   6. 0x02
   7. 0xA0

   Hvad indeholder cachen derefter?

	Valid?	Tag	Valid?	Tag
Set 0				
Set 1				
Set 2				
Set 3				

7 Praktisk Opgave

I den virtuelle maskine ligger den krypterede opgave

test10-05-2016.gpg

hvis du har downloadet den på forhånd, som du burde. Password til denne opgave er:

AyRoactas3

som du bliver bedt om at indtaste når du kører update scriptet. Opgaven bliver pakket ud i mappen

Skrivebord\download-materiale\test10-05-2016

Opgave 9:

1. **(25 pts)**
   En bank har brug for et regnskabsprogram, til at summere en række indtægter og udgifter. Indtægter og udgifter er altid tal mellem -2000000000 kr. ($-2 \cdot 10^9$) og 2000000000 kr. ($2 \cdot 10^9$).
   Banken har besluttet at programmet skal kunne køre på deres gamle mainframe, og derfor må programmet ikke bruge datatyper større end 32 bit, dvs. `int` i C på den virtuelle maskine. Hvis summen ikke kan repræsenteres på 32 bit skal programmet indikere en fejl.
   Du skal skrive en C-funktion ud fra dette skelet (filnavn: `adder.c`):

```
int adder(int *a, int *b) {
        *a = *a + *b;
        return 1;
}
```

således at funktionen returnerer 1 hvis resultatet kan repræsenteres eksakt i a, og returnerer 0 ellers.
C-programmet skal kunne compile med `gcc -O0 -o adder adder.c addertester.c` (ingen optimeringer), hvilket vil resultere i et program `adder` der tester basal funktionalitet. Du kan med fordel tilføje dine egne testcases i `addertester.c`.
Du skal aflevere filen

adder.c

og den skal placeres på USB-stick i en mappe med dit studienummer (1234578 i dette eksempel):

USB-Drev:\1234578\adder.c

Det nemmeste er at kopiere kildekoden (CTRL+C) i den virtuelle maskine, oprette tekstfilen direkte på USB-drevet og paste kildekoden (CTRL+V).