

## Computer Arkitektur 2015 (CART'15): Test

22. Maj 2015

**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.

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

Binær	Hexadecimal	Unsigned	Signed
01001101	0x4D	77	77
1010 0001	0xA1	161	-95
1111 1110	0xFE	254	-2
1110 1111	0xEF	239	-17

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

Udtryk	Svar
$x \& y$	0x08
$x \mid y$	0xB8
$x \wedge y$	0xB0
$\sim x \mid \sim y$	0xF7
$y \ll 2$	0x60

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

- (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?
$x1 \geq 0$	Ja	
$y1 \geq 0$	Nej	$y1 = \text{INT\_MIN}$
Hvis $y1 < 0$ og $x1 = \text{INT\_MAX}$ så vil $y1 > x1$	Ja	
Hvis $y1 < 0$ og $x1 = y1$ udføres, så vil $x1 > 0$	Ja	
Hvis $y1 < 0$ og $y2 = -y1$ udføres, så vil $y2 > 0$	Nej	$y1 = \text{INT\_MIN}$
Hvis $y1 < 0$ og $y2 < 0$ , så vil $y1 + y2 < 0$	Nej	$y1 = \text{INT\_MIN}; y2 = \text{INT\_MIN}$

## 2 Assembly programmer

### Opgave 3:

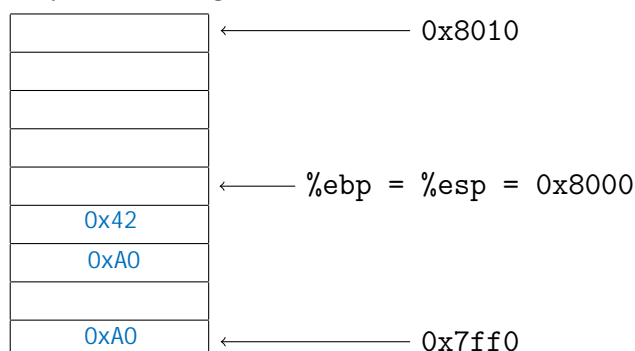
- (6 pts) Forbind disse x86 assembly instruktioner (GNU syntax) med deres ækvivalente C-udtryk (antag `eax` eksisterer som variabel).

Assembly	C-udtryk
<code>mov 0x42, %eax</code>	<code>eax = eax * 4 + 66;</code>
<code>add 0x42, %eax</code>	<code>eax = eax + 42;</code>
<code>lea 42(%eax), %eax</code>	<code>eax = eax + 0x42;</code>
<code>lea 0x42(,%eax, 4), %eax</code>	<code>eax = eax * 5 + 66;</code>
<code>lea 0x42(%eax, %eax, 4), %eax</code>	<code>eax = 66;</code>

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

`mul` og `div` instruktioner bruger flere clock cycles end shift og addition

Det står direkte et eller andet sted i biblen :-)

## 2. (5 pts)

Omskriv disse beregninger til ikke at bruge `mul` og `div` instruktionerne:

```
mul $2, %eax  
    shl $1, %eax    (eller add %eax, %eax )
```

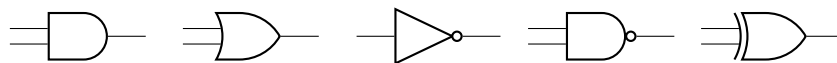
```
mul $3, %eax  
    shl $1, %eax  
    add %eax, %eax
```

```
mul $5, %eax  
    shl $1, %eax  
    add %eax, %eax
```

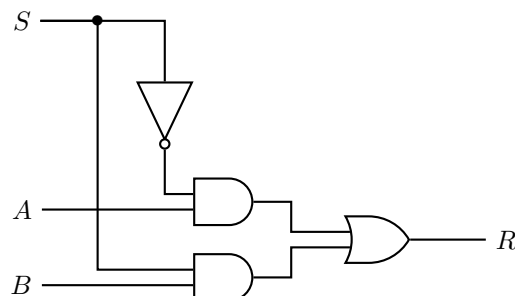
```
mul $16, %eax  
    shl $4, %eax
```

```
div $2, %eax  
    sar $1, %eax    (eller shr $1, %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 formelen for output værdien  $R$  for kredsløbet i Figur 2 som et C-udtryk:

`R = (!S && A) || (S && B)`

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

Erstat AND-gates med NANT-gates efterfulgt af NOT-gates

## 4 Y86 processor arkitektur

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

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.

- (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 $\leftarrow M_1[PC]$ rA:rB $\leftarrow$ valP $\leftarrow PC+1$
Decode	valA $\leftarrow R[\%ebp]$ valB $\leftarrow$
Execute	valE $\leftarrow valA + 4$
Memory	valM $\leftarrow M_4[valA]$
Write back	R[%esp] $\leftarrow valE$ R[%ebp] $\leftarrow valM$
PC update	PC $\leftarrow valP$

Svarer til:

```
esp = ebp
ebp = Memory[esp]
esp = esp + 4
```

Der skal udføres en  
beregning i  
Execute:

ebp + 4

Og en memory  
tilgang:

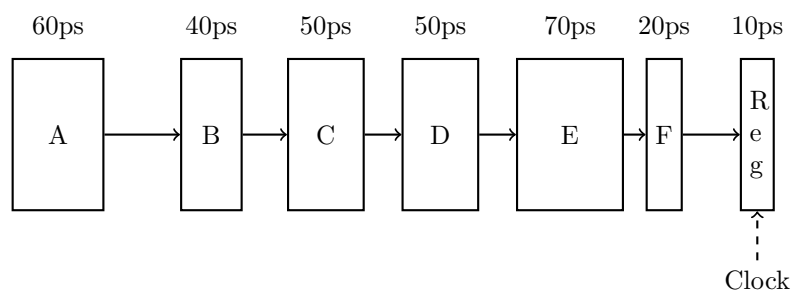
Member[ebp]

rA og rB bruges kun i forbindelse med argumenter (operands) til instruktioner. Leave instruktionen har ingen operands.

Fordi instruktionen leave kun fylder 1 byte

Det er ikke nødvendigt at hente register %esp, da %esp = %ebp

## 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?

300 ps

Summen af de 6 funktionelle dele

Hvad er throughput for det digitale kredsløb i Figur 3?

3,33 GIPS

1 divideret med latency gange med 1000

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

Trial and error :-)

Hvad bliver den resulterende throughput?

6,25 GIPS

Tager udgangspunkt i enheden med den højeste latency. Altså  $(1/160) \cdot 1000$

Hvad bliver latency reduceret til?

160 ps

## 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?

64 bytes

For at løse denne opgave læs kapitel 6.3 i biblen.

Adresse bits  $m=8$   
Set bits  $s=\log_2(4)=2$   
Block bits  $b=\log_2(8)=3$   
Tag bits  $t=m-(s+b)=3$

De første 3 bits i adressen er tag, de næste 2 bits er til set index og de sidste 3 bits går til block offset

3 pts) Antag at følgende adresser bliver tilgået, startende fra en tom cache:

1. 0x64 011 00 100 tag=3 set=0 block\_offset=4
2. 0x70 011 10 000 tag=3 set=2 block\_offset=0
3. 0x00 000 00 000 tag=0 set=0 block\_offset=0
4. 0x01 000 00 001 tag=0 set=0 block\_offset=1
5. 0x78 011 11 000 tag=3 set=3 block\_offset=0
6. 0x02 000 00 010 tag=0 set=0 block\_offset=2
7. 0xA0 101 00 000 tag=5 set=0 block\_offset=0

Hvad indeholder cachen derefter?

	Valid?	Tag	Valid?	Tag
Set 0	1	5	1	3
Set 1				
Set 2	1	3		
Set 3	1	3		

## 7 Praktisk Opgave

I den virtuelle maskine ligger den krypterede opgave

test22-05-2015.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\test22-05-2015

**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;
}
```

```
int adder(int *a, int *b) {
    int sum = *a + *b;
    if (*a > 0 && *b > 0 && sum < 0)
        return 0;
    if (*a < 0 && *b < 0 && sum > 0)
        return 0;
    *a = sum;
    return 1;
}
```

Mere eller mindre en kopi af løsningen til Problem 30 i kapitel 2

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).