

Computer Arkitektur Eksamen

Juni 2018

- **Hovedopgavebesvarelsen skal uploades på digital eksamen.** Besvarelsen skal formateres som en pdf fil, som bedes navngivet '`studienummer.pdf`'.
- Skriv dit studienummer på forsiden af besvarelsen.
- I din besvarelse behøver du ikke at gentage opgaveteksten. Det er tilstrækkeligt tydeligt at identificere svaret på en opgave ved brug af nummeridentifikationen fra opgavesættet. Du må selv vælge hvilket tekstbehandlingssystem, du anvender. Alternativt kan du pdf-annotere direkte i opgavearket.
- Det kan være en god ide at læse opgaverne igennem, inden du begynder besvarelsen, så du kan vurdere hvor du evt. skal prioritere for at samle flest points.
- Hvis du mener, at der er fejl i en opgave, eller at du mangler en oplysning, så skriv din antagelse for din løsning ned sammen med løsningen.
- Dette eksamenssæt er delt op i 7 dele, der i alt giver 100 points.

1 Repræsentation og manipulation af information

Opgave 1:

1. **(4 pts)** Udfyld denne tabel, således at rækkerne har den samme **6-bit** integer i de forskellige repræsentationer. Til signed anvendes "twos complement".

Nr.	Binær	Hexadecimal	Unsigned	Signed
1	011101			
2		0x11		
3			60	
4				-2

2. **(5 pts)** Lad $x = 133$ decimalt og $y = 0x33$ hexadecimalt, begge 8-bit unsigned. Udfyld denne tabel af bitvise udtryk med svar i hexadecimalt:

Nr.	Udtryk	Svar 0x...
1	$x \mid y$	
2	$x \& y$	
3	$x \wedge y$	
4	$\sim x \wedge \sim y$	
5	$x \gg 3$ (logisk)	
6	$x \gg 3$ (aritmetisk)	

Opgave 2: Antag Signed/Unsigned Integers repræsenteret i 32 bits. Til signed anvendes two's complement.

1. (5 pts) Antag at følgende er erklæret i et C-program:

```
int a1, a2;
unsigned int b1, b2;
```

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`, eller potenser af 2).

Nr.	Udsagn	Sandt?	Modeksempel?
1	<code>b1 >= 0</code>		<code>b1 =</code>
2	<code>a1 >= 0</code>		<code>a1 =</code>
3	Hvis <code>a1 < 0</code> og <code>b1 = INT_MAX</code> , så vil <code>b1 >= a1</code>		<code>a1 =</code>
4	Udtrykket <code>((-a1) > (-a2))</code> kan altid simplificeres til <code>(a1 < a2)</code>		<code>a1 =</code> <code>a2 =</code>
5	Hvis <code>b1 >= 0</code> og <code>b2 >= 0</code> , så vil <code>b1 + b2 >= 0</code>		<code>b1 =</code> <code>b2 =</code>
6	Hvis <code>a1 < 0</code> og <code>a2 > 0</code> , så vil <code>a1 - a2 < 0</code>		<code>a1 =</code> <code>a2 =</code>

2 Assembly programmer

Opgave 3:

1. (5 pts)

Angiv for hver udsagn om det er sandt eller falsk.

	Udsagn	Sandt	Falsk
1	Instruktionen <code>lea 8(%rax, %rbx, 8), %rax</code> gemmer adressen <code>8*%rbx+%rax</code> i <code>%rax</code> .		
2	Instruktionen <code>subq \$32, %rsp</code> de-allokerer 32 bytes fra stakken.		
3	<code>%rbx</code> skal gemmes af den kaldte procedure (Callee saved).		
4	<code>%rsi</code> skal gemmes af den kaldende procedure (Caller saved).		
5	Instruktionen <code>cmpq</code> udfører en sammenligning mellem værdierne af to registre ved brug af bitvis logisk 'og' (&).		

2. (5 pts) I det følgende er vist to X86-64 assembly programmer, oversat fra samme C-funktion `calc`. Ligeledes er vist et øjebliksbillede af hukommelsen. Hvilken værdi returnerer funktionen når den kaldes med argumenterne `calc(0x601060, 10)` ?

Listing 1: Oversættelse 1

```
calc:
    movl    $0, %eax
    movl    $0, %edx
    jmp     .L2
.L3:
    addq    (%rdi,%rdx,8), %rax
    addq    $2, %rdx
.L2:
    cmpq    %rsi, %rdx
    jl      .L3
    ret
```

Listing 2: Oversættelse 2

```
calc:
    testq   %rsi, %rsi
    jle     .L4
    movl    $0, %eax
    movl    $0, %edx
.L3:
    addq    (%rdi,%rdx,8), %rax
    addq    $2, %rdx
    cmpq    %rdx, %rsi
    jg      .L3
    ret
.L4:
    movl    $0, %eax
    ret
```

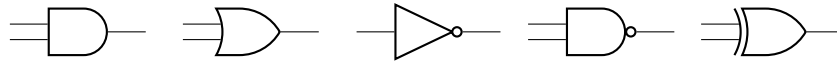
Hukommelsen:	
Adresse	Værdi (dec)
0x601060	1
0x601068	2
0x601070	3
0x601078	4
0x601080	5
0x601088	6
0x601090	7
0x601098	8
0x6010a0	9
0x6010a8	10
0x6010b0	11
0x6010b8	12

3. (4 pts) C-funktionen anvender en velkendt kontrol-struktur. Hvilken kontrol struktur er der tale om, og hvilken oversættelsesmetode (translation method) har compileren anvendt til oversættelse til assembly for de to listings:

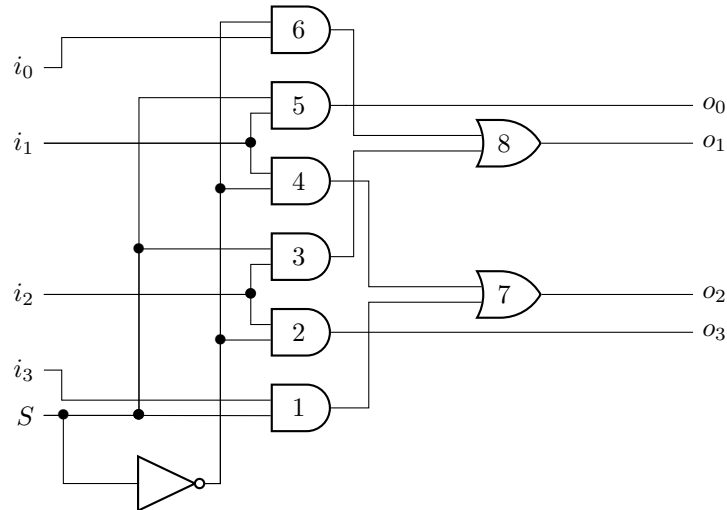
Kontrol-struktur	
------------------	--

Variant	Oversættelsesmetode
Oversættelse 1	
Oversættelse 2	

3 Digital Logik



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



Figur 2: Et digitalt logisk kredsløb.

Opgave 4: Figur 1 viser nogle logiske gates. Figur 2 viser et digitalt logisk kredsløb.

- (5 pts) Betragt det kombinatoriske kredsløb i Figur 2. **Antag at S modtager input værdien 0.** Skriv formelen for output værdierne $o_0 \dots o_3$ som et udtryk i boolsk algebra ved brug af de boolske operatorer $\&$ og \sim (jfv. side 87 i bogen), og reducer udsagnet.

- | | |
|------------------|-----------|
| 1. $o_0 = \dots$ | $= \dots$ |
| 2. $o_1 = \dots$ | $= \dots$ |
| 3. $o_2 = \dots$ | $= \dots$ |
| 4. $o_3 = \dots$ | $= \dots$ |

2. (4 pts) Udfyld denne sandhedstabel for kredsløbet i Figur 2:

S	i_0	i_1	i_2	i_3	o_0	o_1	o_2	o_3
0	0	0	0	0				
0	1	0	0	0				
0	0	1	0	0				
0	1	1	0	0				
0	0	0	1	0				
0	1	0	1	0				
0	0	1	1	0				
0	1	1	1	0				
0	0	0	0	1				
0	1	0	0	1				
0	0	1	0	1				
0	1	1	0	1				
0	0	0	1	1				
0	1	0	1	1				
0	0	1	1	1				
0	1	1	1	1				

3. (3 pts) Hvilken operation implementerer kredsløbet?

--

4 Y86-64 processor arkitektur

Stage	OPq rA, rB	Stage	addq %rdi, %r9
Fetch	icode:ifun \leftarrow M ₁ [PC] rA:rB \leftarrow M ₁ [PC+1] valP \leftarrow PC+2	Fetch	icode:ifun \leftarrow M ₁ [...] = ... rA:rB \leftarrow M ₁ [...] = ... valP \leftarrow ... = ...
Decode	valA \leftarrow R[rA] valB \leftarrow R[rB]	Decode	valA \leftarrow R[...] = ... valB \leftarrow R[...] = ...
Execute	valE \leftarrow valB OP valA Set CC	Execute	valE \leftarrow ... = ... ZF \leftarrow ..., SF \leftarrow ..., OF \leftarrow ...
Memory		Memory	
Write back	R[rB] \leftarrow valE	Write back	R[...] \leftarrow ...
PC update	PC \leftarrow valP	PC update	PC \leftarrow ...

Tabel 1: Beregningstrin for aritmetiske instruktioner, og instansen `addq %rdi, %r9`

Opgave 5: I denne opgave antager vi processor arkitekturen Y86-64, som beskrevet i lærebogen.

1. (5 pts)

Spor effekten af den konkrete instruktion ved at udfylde tabellen til højre i Tabel 1 med konkrete værdier for instruktionen `addq %rdi,%r9`. Antag at instruktionen er placeret i adresse `0x82`, `%rdi` har værdien `0x100`, og `%r9` har værdien `0x108`.

2. (7 pts)

Af effektivitetshensyn vil vi tilføje en specialiseret instruktion `iaddq V, rB`, der adderer en immediate værdi `V`, til et register `rB`, således at `rB` kommer til at indeholde summen af `rB` og `V`.

Vi kan bruge byte encodingen `0x64`, som instruktions encoding for `iaddq`, med de efterfølgende bytes til at angive værdien.

Udfyld nedenstående tabel med de nødvendige beregninger.

Stage	iaddq V, rB
Fetch	icode:ifun \leftarrow ... rA:rB \leftarrow ... valC \leftarrow ... valP \leftarrow ...
Decode	valB \leftarrow ...
Execute	valE \leftarrow
Memory	
Write back	... \leftarrow ...
PC update	PC \leftarrow valP

5 Caching

Opgave 6:

1. (4 pts)

Betragt følgende C program, som beregner summen af elementerne i et 2-D array.

```
long arr_sum(long a[M][N]) {
    long sum, i, j;
    for(i=0; i<M; i++){
        for(j=0; j<N; j++){
            sum+=a[i][j];
        }
    }
}
```

Hvordan vil du karakterisere programmets lokalitetsadfærd (sæt 'X')?

- ☐ 1. Kun god temporal lokalitet.
☐ 2. Kun god spatial lokalitet.
☐ 3. Både god spatial og temporal lokalitet.
☐ 4. Hverken god spatial eller temporal lokalitet.

2. (8 pts)

I det følgende er vist en 2-vejs associative cache med 8 sets og en blokstørrelse på 8 bytes (byte med offset 0 er vist til venstre). Addresser er 8 bit. Antag i tilfælde af cache miss at den pågældende cache line fyldes med værdien 0xAABBCCDDAABBCCDD.

	Line 1										Line 2									
Set	V	Tag	Value (Hex)								V	Tag	Value (Hex)							
0	0	3	12	13	14	15	A0	A1	A2	A3	1	2	C1	C2	C3	C4	C5	C6	B6	C6
1	1	2	A0	B0	C0	D0	98	97	80	81	1	3	12	23	34	45	12	34	56	78
2	1	3	00	00	00	00	B9	B1	B2	B8	1	2	76	65	54	43	65	65	87	86
3	1	0	00	00	00	00	00	01	02	03	1	3	00	01	02	03	00	00	00	00
4	0	3	22	23	24	25	B0	B1	B2	B3	1	2	BF	BE	BD	BC	BB	BA	B9	B8
5	1	2	A1	B1	C1	D1	88	87	86	85	1	3	11	11	11	11	01	01	01	01
6	0	3	A2	B2	B7	B8	A3	B3	B7	C1	1	2	E6	E5	E4	E3	E5	E5	E7	E6
7	1	0	FA	FB	FC	FD	FE	F1	F2	F3	1	3	D8	D9	DA	DB	AC	DC	DE	DF

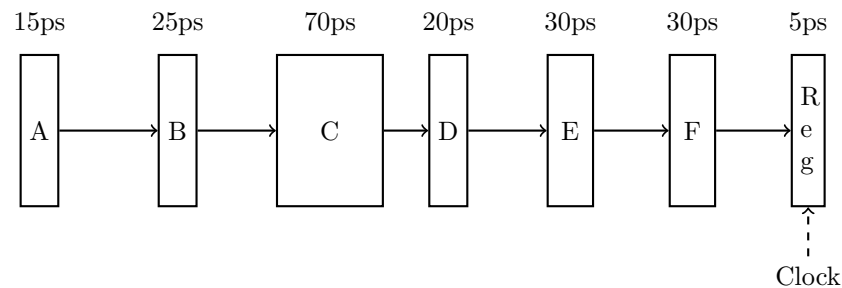
- Udfyld nedenstående tabel med angivelse af hvilke adresse bits, der skal bruges til hhv.
 - block offset (angiv med et 'B')
 - set index (angiv med et 'I')
 - tag (angiv med et 'T')

7	6	5	4	3	2	1	0

- Udfyld nedenstående tabel når cache-systemet læser en byte for den angivne serie af referencer (adresser). Tag, Set og Offset skal angives som decimal tal, og byte-værdi i Hex.

Adresse	Tag	Set	Offset	Værdi
0x8B				
0xB7				
0xE5				
0x4E				
0xE1				
0xFF				

6 Pipelines



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

Opgave 7:

- (4 pts)** Figur 3 viser et digitalt kredsløb. Vi ønsker at øge dets throughput vha. en 3-trins pipeline. Optimer throughput for det digitale kredsløb i Figur 3, ved at indsætte yderligere 2 hardware registre mellem de funktionelle dele. Antag at hardware registre har delay på 5ps.

- Indiker med et "R" de positioner i nedenstående streng, hvor du vil indsætte hardware registre. Svaret kan gives som en streng med navnene på de funktionelle dele, a la "X R YZ R".

A B C D E F

- Hvad bliver den resulterende throughput?

- Hvad bliver latency ?

- (7pts)**

Antag en 5-trins pipeline arkitektur, der *ikke* har nogen form for videresendelse af data (data-forwarding) a la PIPE⁺. Den opererer altså med følgende trin

F Fetch

D Decode

E Execute

M Memory

W Writeback

Nedenstående viser et fragment af et Y86 assembler program med tilhørende naive pipeline timing.

	Program	Timing					
1	mrmovq (%rax), %rdx	F	D	E	M	W	
2	subq %rdi, %rax		F	D	E	M	W
3	addq %r8, %rax			F	D	E	M W
4	mrmovq (%rax), %rdi				F	D	E M W

- Angiv med ovennævnte timing, hvilke data-afhængigheder, der giver anledning til data-hazard:

	Spørgsmål	Sandt	Falsk
1	Der er data hazard imellem %rax i linie 1 og %rax i linie 2	<input type="checkbox"/>	<input type="checkbox"/>
2	Der er data hazard imellem %rax i linie 1 og %rax i linie 3	<input type="checkbox"/>	<input type="checkbox"/>
3	Der er data hazard imellem %rax i linie 1 og %rax i linie 4	<input type="checkbox"/>	<input type="checkbox"/>
4	Der er data hazard imellem %rax i linie 2 og %rax i linie 3	<input type="checkbox"/>	<input type="checkbox"/>
5	Der er data hazard imellem %rax i linie 2 og %rax i linie 4	<input type="checkbox"/>	<input type="checkbox"/>
6	Der er data hazard imellem %rax i linie 3 og %rax i linie 4	<input type="checkbox"/>	<input type="checkbox"/>
7	Der er data hazard imellem %rdi i linie 2 og %rdi i linie 4	<input type="checkbox"/>	<input type="checkbox"/>

- Antag at arkitekturen anvender *stalling* til at håndtere data-hazard. Vis via et pipeline diagram, hvordan arkitekturen kan opnå korrekt timing af instruktionerne.

	Instruktion	Timing														
1	mrmovq (%rax), %rdx															
2	subq %rdi, %rax															
3	addq %r8, %rax															
4	mrmovq (%rax), %rdi															

7 Praktisk Opgave

I den virtuelle maskine ligger den krypterede opgave

exam-08-06-2018-files.gpg

hvis du har downloadet den på forhånd. Den findes også på

<http://people.cs.aau.dk/~bnielsen/exam-08-06-2018-files.gpg>.

Den kan hentes med firefox browseren i den virtuelle maskine, eller i en terminal med kommanderne:

1. `cd Skrivebord/download-materiale/`
2. `wget http://people.cs.aau.dk/~bnielsen/exam-08-06-2018-files.gpg`
3. `gpg-zip --decrypt exam-08-06-2018-files.gpg`

Password til denne opgave er:

V2Wy8Um7ax

som du bliver bedt om at indtaste når filen dekrypteres. Opgaven bliver pakket ud i mappen

Skrivebord\download-materiale\exam-08-06-2018-files.gpg

Alternativt kan den hentes i bilaget på digital eksamen som zip fil.

Opgave 8:

1. (15 pts) Filen

binarybomb

er et eksekverbart terminal program. Programmet beder dig indtaste første bogstav i dit fornavn. Anvend et lille bogstav i intervallet 'a'-'z', Hvis dit navn begynder med andet tegn, så vælg et i intervallet, som du mener matcher bedst. Du har således en "personlig" variant af programmet. Programmet udskriver en velkomst hilsen med bogstavet og dets decimal værdi i ascii ('a' har værdien 97, videre til 'z' med værdien '122').

- Programmet spørger dernæst efter 2 tal som input. Disassembler filen med `objdump -d binarybomb`.

Angiv dit begyndelsesbogstav, den tilhørende decimal værdi, samt de 2 inputs, som det accepterer. Decimalværdien for dit begyndelsesbogstav bliver overført som første parameter til de relevante funktioner.

Bogstav &Dec. værdi	
Input 1	
Input 2	

- I hvilken adresse starter `main` funktionen?

- Hvad er objekt kode indkodningen for instruktionen, der starter i adresse `0x40072d`?

2. (10 pts) Nedenfor er angivet en lille C-funktion, der foretager en beregning over elementerne i to arrays *a* og *b*, hver af længden *Length*. Omskriv funktionen `calc`, så den benytter 2 gange udfoldning af løkken og to opsamlingsvariable (2 x loop unrolling with two accumulator variables). Funktionen kan passende kaldes `calc2x2`.

```
long calc(long*a, long*b, const int Length){
    long res=0;
    int i;
    for(i=0;i<Length;i++){
        res+=a[i]*b[i];
    }

    return res;
}
```

I filen findes et test-program `unrolling-tester.o` til virtuelle maskine, som kan hjælpe dig med at udvikle og teste din løsning. Du kan skrive din løsning i funktionen `calc2x2` i filen `unroll.c`, og teste den med kommandoerne:

```
prompt> gcc unroll.c unroll-tester.o -o unroll
prompt> ./unroll
```

Du kan angive svaret nedenfor (også selvom det ikke virker, eller kører i den virtuelle maskine):

```
long calc2x2(long*a, long*b, const int length){
    long res=0;

    return res;
}
```

Hvad udskriver programmet, når du kører test-programmet?