Eksamen i Computer Arkitektur og Operativ Systemer

Juni 2021

- **Hovedopgavebesvarelsen skal uploades på digital eksamen.** Besvarelsen skal formatteres som en pdf fil.
- Du kan lave besvarelsen ved at pdf-annotere direkte i opgavearket. Alternativt kan du udfærdige besvarelsen med et tekstbehandlingsprogram, der kan producere en pdf fil. I din besvarelse behøver du ikke at gentage opgaveteksten, men det er tilstrækkeligt tydeligt at identificere svaret på en opgave ved brug af nummeridentifikationen fra opgavesættet.
- Det kan være en god ide at **læse opgaverne igennem først**, 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:

Den hypotetiske CAOSv1 computer anvender **w=14-bits** til repræsentation af heltal. Til signed heltal anvendes two's complement.
Antag at følgende er erklæret i et C-program:

```
unsigned int x1 = UINT_MIN;
unsigned int x2 = UINT_MAX; //UMax_w
int y1 = INT_MIN; //TMin_w
int y2 = INT_MAX; //TMax_w
int y3 = 1;
```

1. **(4 pts)** Beregn de konkrete værdier af nedenstående udtryk; angiv tallets binære repræsentation og decimale værdi.

Nr.	Udtryk	Binær repræsentation	Værdi (dec.)
1	x1		
2	x2		
3	y1		
4	y2		

2. **(6 pts)** Beregn de konkrete værdier af nedenstående udtryk; angiv tallets decimale værdi og angiv kort forklaring til hvordan resultatet fremkom. Ved boolske udtryk kan du anvende 0 for falsk, og 1 for sand.

Nr.	Udtryk	Resultat	Forklaring
5	y2+y3		
6	y1-y3		
7	y1+x1		
8	x2+y3		
9	(y1+y2==0)		
10	(-y1==y1)		

Solution:

Nr.	Udtryk	Værdi i decimal	kommentar
1	x1	0	$B2U_{14}$(00 0000 0000 0000)= 0
2	x2	16383	$B2U_{14}$(11 1111 1111 1111)=16383
3	y1	-8192	$B2T_{14}$(10 0000 0000 0000)= -8192
4	y2	8191	$B2T_{14}$ (01 1111 1111 1111)= 8191
5	y2+y3	-8192	overløb i positiv retning Tmax+1==Tmin
6	y1-y3	8191	overløb i negativ retning Tmin-1==Tmax
7	y1+x1	8192	blandet, beregnes som unsigned!
8	x2+y3	0	blandet, beregnes som unsigned, overflow
9	(y1+y2==0)	0 false	1 negativt tal mere end positive
10	(-y1==y1)	1 true	(-TMin=TMin)

Opgave 2:

1. **(5 pts)**
   Lad $x = 200$ decimalt og $y =$ 0xAA hexadecimalt, begge 8-bit unsigned. Udfyld denne tabel af bitvise udtryk med svar i hexadecimalt:

Nr.	Udtryk	Svar 0x...
1	`x | y`	
2	`x ^ ~y`	
3	`y <<4`	
4	`x >>3 (logisk)`	
5	`x >>3 (aritmetisk)`	

Solution:

x	=	1100 1000
y	=	1010 1010
`x | y`	=	1110 1010 = EA = 234

x	=	1100 1000	x	= 1100 1000
y	=	1010 1010	$\sim$y	= 0101 0101
`x ^ ~y`	=		=	1001 1101 = 9D = 157

y	=	1010 1010
`y <<4`	=	1010 0000 = A0 = 160

x	=	1100 1000
`x >>3 (logisk)`	=	0001 1001 =19 =25

x	=	1100 1000
`x >>3 (aritmetisk)`	=	1111 1001= F9 = 249

Nr.	Udtryk	Svar 0x...
1	`x | y`	EA
2	`x ^ ~y`	9D
3	`y <<4`	A0
4	`x >>3 (logisk)`	19
5	`x >>3 (aritmetisk)`	F9

Solution:

2 Assembly programmer

Opgave 3:

1. **(5 pts)**
   Angiv for hver udsagn om det er sandt eller falsk.

	Udsagn	Sandt	Falsk
1	Instruktionen `movl` flytter 4 bytes.		
2	Lad x være et heltal. Udtrykket 3*x kan i assembler beregnes som `x<<3` (vha. instruktionen `shl`)		
3	`RET` instruktionen skubber retur addressen på stakken.		
4	%rdi skal gemmes af den kaldende procedure (Caller saved).		
5	%rsi skal gemmes af den kaldte procedure (Callee saved).		

Opgave 4:

I det følgende er vist et stykke X86-64 assembly program, som gcc har oversat fra en lille C-funktion. Programlistingen viser den disassemblerede udgave af funktionen `det`.

Listing 1: Disassembleret oversat C program.

```
1  det:
2  0x401617 <+0>:  movq   0x8(%rsi),%rax
3  0x40161b <+4>:  imul   (%rdi),%rax
4  0x40161f <+8>:  movq   (%rsi),%rdx
5  0x401622 <+11>: imul   0x8(%rdi),%rdx
6  0x401627 <+16>: subq   %rdx,%rax
7  0x40162a <+19>: js     0x40052e <det+23>
8  0x40162c <+21>: retq
9  0x40162e <+23>: negq   %rax
10 0x401631 <+26>: jmp    0x40052c <det+21>
```

Hukommelsen:	
Addresse	Værdi
0x601030	6
0x601038	5
0x601040	8
0x601048	7

1. **(5 pts)** Angiv C-funktionens prototype (dvs retur- og parameter typer). Fx. `int func(char*,int,long);` hvis funktionen `func` returnerer en int, og tager en char pointer som første parameter, en int som anden parameter, og long som tredie parameter, osv.)

   Angiv ligeledes den/de kontrol-strukturer funktionen anvender.

   Prototype erklæring = ...

   Kontrol strukturer = ...

2. **(10 pts)**
   Figuren ovenfor viser desuden et stykke hukommelse med et antal heltallige værdier. Lav et beregningsspor (execution trace) ved at udfylde nedenstående tabel når funktionen kaldes med `det(0x601040,0x601030)`. Dvs. angiv effekten på tilstanden ved at hånd-eksekvere instruktionerne én efter én. I en linie skal du angive den tilstand, der gælder *inden* instruktionen udføres; dens resultat (ændring af register-værdier) skal dermed stå i linien under. Hvis der ingen ændring er, kan du lade feltet stå tomt.

Instruktion		Tilstand (ved start)				
PC	instruktion	%rdi	%rsi	%rdx	%rax	SF (signflag)

Solution:

	Udsagn	Sandt	Falsk
1	Instruktionen `movl` flytter 4 bytes.	T	
2	Lad x være et heltal. Udtrykket 3*x kan i assembler beregnes som `x<<3` (vha. instruktionen `shl`)		F
3	`RET` instruktionen skubber retur addressen på stakken.		F
4	%rdi skal gemmes af den kaldende procedure (Caller saved).	T	
5	%rsi skal gemmes af den kaldte procedure (Callee saved).		F

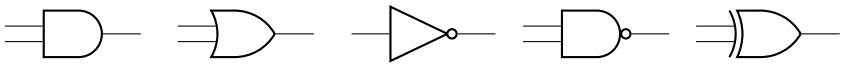
Solution:

```
long det(long*,long*);
```
Argument 1 type = long* (long pointer)
Argument 2 type = long* (long pointer)
Retur type = long
Kontrol = If-then selektion

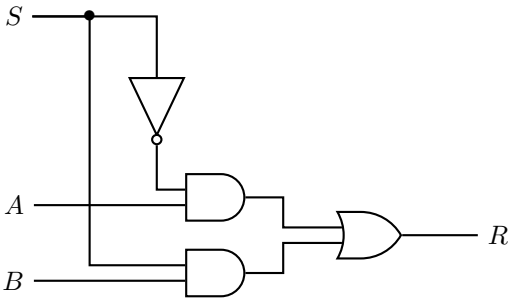
Solution:

Instruktion		Tilstand (ved start)				
PC	instruktion	%rdi	%rsi	%rdx	%rax	SF (signflag))
401617	movq	601040	601030			
40161b	imul				5	
40161f	movq				5*8=40	0
401622	imul			6		
401627	subq			6*7=42		0
40162a	js				40-42=-2	1
40162e	negq					
401631	jmp				2	0
40162c	retq					

3 Processor Realisering



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 anvender disse til opbygning af et digitalt logisk kredsløb.

1. **(5 pts)** Udfyld denne sandhedstabel for kredsløbet i Figur 2:

A	B	S	R
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

Opgave 6: I denne opgave antager vi processor arkitekturen SEQ Y86-64, som beskrevet i lærebogen.

1. **(5 pts)** Spor effekten af udførelsen af den konkrete instruktion ved at udfylde nedenstående tabel med specifikke konkrete værdier for instruktionen `push %rsi`, hvor det er givet, at `%rsp` har værdien `0x348`, `%rsi` har værdien 0x3, og at program tælleren PC har værdien `0x77774402`.

Stage	`pushq %rsi`
Fetch	icode:ifun $\leftarrow$ M_1[...]=... rA:rB $\leftarrow$ M_1[...]=... valP $\leftarrow$ ... = ...
Decode	valA $\leftarrow$ R[...] =... valB $\leftarrow$ R[%rsp] = ...
Execute	valE $\leftarrow$ valB + (-8) =...
Memory	M_8[...] $\leftarrow$ ...
Write back	R[%rsp] $\leftarrow$ ...
PC update	PC $\leftarrow$ ...

Solution:

A	B	S	R
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

Solution:

Stage	`pushq %rsi`
Fetch	icode:ifun ← M_1[0x77774402]=A:0 rA:rB ← M_1[0x77774403]=6:F valP ← 0x77774402+2 = 0x77774404
Decode	valA ← R[6 el.%rsi]=0x3 valB ← R[4 el. %rsp] =0x348
Execute	valE ← valB + (-8) =0x340
Memory	M_8[0x340] ← 0x3
Write back	R[4 el. %rsp] ← 0x340
PC update	PC ← 0x77774404

4 Optimering og Instruktionsniveau parallelitet

Opgave 7:

Nedenfor er angivet en lille C-funktion `calc`, der foretager en beregning over elementerne i et array med N vektorer (repræsenteret som en struct `vec_t`. Funktionen leverer resultatet via resultat *pointeren* i parameter listen.

Listing 3: En funktion og forsøg på optimering af denne.

```
1  typedef struct vec {
2    long x;
3    long y;
4  } vec_t;
5  #define N 10000
6  vec_t v[N];
7
8  void calc(vec_t *v, const int Length, long* calcResult){
9    int i;
10   *calcResult=0;
11   for(i=0;i<Length;i++){
12     *calcResult+=v[i].x*v[i].x + v[i].y*v[i].y;
13   }
14  }
15 int main(){
16   long res;
17   calc(v,N,&res);
18 }
```

1. **(10 pts)** I et forsøg på at optimere denne funktion ønsker vi at lave en 3 gange udfoldning af løkken med 3 opsamlingsvariable (3x3 loop unrolling). Angiv den nødvendige koden hertil i funktionen `calc3x3`:
   Hint: Hvis du er usikker på den eksakte syntaks, kan du stadig angive løsningen som pseudo-kode, evt. forklarende tekst.

2. **(5 pts)** Eksperimenter på en testmaskine viser at en program-transformation, som anvender en enkelt lokal akkumulator variabel (fx. 2×1 unrolling), giver en betragtelig hastighedsgevinst, men at en 3x3 udfoldning ikke giver nogen betydelig gevinst derudover.
Forklar kort med egne ord, hvorfor førstnævnte gevinst nåes, og en sandsynlig forklaring på at yderligere gevinst ikke opnåes.
Hint: Tænk i termer af "optimization blockers", og mulighed/begrænsning for parallel beregning.

Solution:

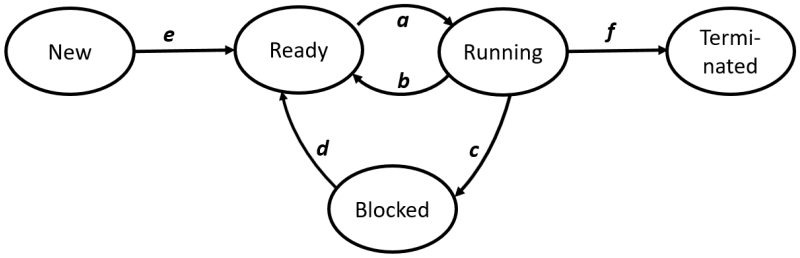
Listing 4: Erklæringer.

```
1  long calcOptim(vec_t *v, int Length, long*calcResult){
2    long res0=0;
3    long res1=0;
4    long res2=0;
5    const int limit=Length-2;
6    int i;
7
8    //*calcResult=0;
9    for(i=0;i<limit;i+=3){
10     res0+= v[i].x * v[i].x + v[i].y * v[i].y ;
11     res1+= v[i+1].x * v[i+1].x + v[i+1].y * v[i+1].y ;
12     res2+= v[i+2].x * v[i+2].x + v[i+2].y * v[i+2].y ;
13   }
14
15   for(;i<Length;i++){
16     res0+=v[i].x * v[i].x + v[i].y * v[i].y ;
17   }
18
19   res0=res0+res1+res2;
20   *calcResult=res0;
21 }
```

Solution:

- Ved at optælle i en lokal variabel, fjerner vi **hukommelses-aliaseringen** mellem resultat pointeren og vektoren. Dette er vigtigt 1) for at akkumuleringen kan ske i et register uden dyr mellemliggende læsning/skrivning til hukommelsen, og 2) ligeledes at nødvendigt for at multiplikationerne i en udfoldning ($v[i]$ og $v[i+1]$) ske parallelt, da aliaseringen kan forårsage at resultatet skrives til $v[i+1]$.
  Optimeringen her er sikker og korrekt da der ikke kan opstå aliasering da resultatet er sat op i `main` til at blive leveret i en lokal variabel, som er adskilt fra arrayet!!
- I 3x3 udfoldningen er der lagt op til 6 parallelle multiplikationerne og 6+1 summer. Men det er sandsynligt, at processor-kernen, som afvikler programmet har færre multiplikations-enheder, og at vi har nået dens produktivitetsgrænse. Endvidere bliver der også pres på load-enhederne til indlæsning af 6 longs fra hukommelsen pr iteration. (Eksempel arkitekturen i lærerbogen har faktisk kun een enhed til heltalsmultiplikation: fuldt pipelinet med latency 3)

5 Operativ Systemer



Figur 3: Tråd-tilstandsmodel

Opgave 8: Multiprogrammering/Multitasking

1. **(5 pts.)** Afkryds de egenskaber, der kendetegner multi-programmering?
   1. ☐ processor afvikler flere programmer tilsyneladende samtidigt
   2. ☐ muliggør at operativ systemet kan programmeres af flere udviklere
   3. ☐ muliggør større udnyttelsesgrad af ydre enheder
   4. ☐ muliggør større udnyttelsesgrad af processoren
   5. ☐ er kun en fordel når processoren har flere kerner (til parallel beregning)
   6. ☐ tillader multiple brugere

2. **(5 pts.)** Betragt trådtilstandsmodellen i figur 3. Angiv med bogstaverne $a - f$ den tilstandsovergang, der svarer til nedenstående situationer:

Nr.	Overgang	Beskrivelse
1		En kørende tråd afbrydes af et timer interrupt, som angiver, at dens timeslice er udløbet
2		Tråden kalder wait på en semafor med værdien 0
3		Tråden kalder en længerevarende synkron I/O operation
4		Tråden kalder yield
5		Pagefault indtræffer ved adgang til en side, som ikke er resident i RAM.

Solution:
1,3,4,6

Solution:
1b, 2c, 3c, 4b, 5c, (5f)

6 Hukommelses Organisering

Opgave 9: **(15 pts)**
I denne opgave skal du oversætte virtuelle til fysiske adresser i nedenstående mini hukommelsesystem givet ved:

- Sidestørrelsen er 32 bytes
- Virtuelle adresser er 14 bits lange
- Fysiske adresser er 12 bits lange
- Der er tilknyttet en 3-vejs TLB (translation lookaside buffer) med 4 sets.
- Hukommelsen kan udlæses byte-vist.
- Sidetabel og TLB har følgende indhold:

VPN	PPN	Valid
00	7F	1
01	6E	1
02	5D	1
03	4C	1
04	3B	1
05	2A	1
06	19	1
07	08	1

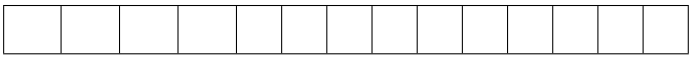
VPN	PPN	Valid
A8	06	1
A9	17	1
AA	28	0
AB	39	1
AC	4A	1
AD	5B	1
AE	01	1
AF	00	0

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	01	3B	0	2B	4A	1	2A	06	0
1	2A	17	1	01	2A	0	00	6E	1
2	00	5D	0	01	19	1	2B	01	1
3	2A	39	1	2E	BC	0	2B	00	0

Vis hvordan følgende (uafhængige) virtuelle adresser oversættes til fysiske adresser.

1. Virtuel Adresse: `0x150A`

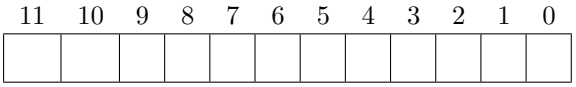
   1. Bits i Virtuelle Adresse



   2. Adresseoversættelse

Parameter	Værdi
VPN	
TLB Indeks	
TLB tag	
TLB Hit (J/N)	
Page fault (J/N)	
PPN	

   3. Bits i Fysisk Adresse



2. Virtuel Adresse: `0x15EA`

1. Bits i Virtuelle Adresse

13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Adresseoversættelse

Parameter	Værdi
VPN	
TLB Indeks	
TLB tag	
TLB Hit (J/N)	
Page fault (J/N)	
PPN	

3. Bits i Fysisk Adresse

11	10	9	8	7	6	5	4	3	2	1	0

3. Virtuel Adresse: `0x00CF`

1. Bits i Virtuelle Adresse

13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. Adresseoversættelse

Parameter	Værdi
VPN	_____
TLB Indeks	_____
TLB tag	_____
TLB Hit (J/N)	_____
Page fault (J/N)	_____
PPN	_____

3. Bits i Fysisk Adresse

11	10	9	8	7	6	5	4	3	2	1	0

Solution:
0x150A = 010101000.01010
VPN= A8
TLI=00
TLB TAG=2A
TLB MISS (valid=0)
PageTable[A8]: PPN 06
Pagefault:0
PA=0000110.01010=CA

0x15EA = 010101111.01010
VPN: AF
TLI=3
TLBT=2B
TLB MISS= (valid=0)
Pagetable[AF]: valid=0
Pagefault
(No PPN/PA)

0xCF=000000110.01111
VPN: 6
TLBI=2
TAG=01
TLP HIT
Pagefault:No
PPN 19
PA:0011001.01111 = 032F

1590 = 0101011 00.10000 =

VPN: AC
TLBI:0
TLBT:2B
TLB Hit
Pagefault:No
PPN: 4A
PA:1001010.10000

ANSWERS FOR 6 BITS VPO:
Many forget that the page size is a paramter of a VM system; in this assignment 5 bits whereas the book example uses a different page size (in real life a page is around 4 kilobytes). With that (big) mistake, the solutions become:

0x150A = 00010101 00 001010
VPN= 54
TLI=00
TLB TAG=15
TLB MISS (no match)
PageTable[54]: PF
Pagefault
No PA

0x15EA = 10101 11 101010
VPN: 57
TLI=3
TLBT=15
TLB MISS= (no match)
pageTable[57]: PF
Pagefault
No PA

0xCF=11 001111
VPN: 3
TLBI=3
TAG=0
TLB MISS
PPN: PageTable[3]=4C
PA: 01001100 101111 = 132F

1590 = 010101 10 010000 =
VPN: 56
TLBI:2
TLBT:15
TLB MISS
PageTable[57]=PF
PageFault
No PA.

7 Concurrency

Opgave 10: **(15 pts)**
Nedenfor er skitseret en lille Pthreads program, med 3 tråde, der (potentielt) parallelt skal beregne summen af elementerne i et (hypotetisk lille) array. Hver tråd skal bruge den samlede sum som et del-resultat i en videre beregning, som ikke er en del af opgaven. En tråd gemmer sin delsum i et andet 3-element array på index svarende til sit `id`, overført som parameter. For at virke korrekt, skal det sikres at hver tråd afventer de 2 andres del-summer i linie 22 inden de beregner den samlede sum. Altså, inden en tråd fortsætter i linie 24, skal den sikre at de 2 øvrige har afsluttet beregningen op til linie 19.

Vis vha. Pthreads mutexes og condition variable hvordan du kan opnå den krævede synkronisering. *Hint:* programmet er listet som verbatim, så det er nemt at copy-paste ind i din virtuelle maskine til videre-udvikling.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "common.h"
#include "common_threads.h"

const int N=3;
long arr[9]={1,2,3, 4,5,6, 7,8,9};
long sum[3]={0,0,0};

//Mutex og condition erklaeringer

void *mythread(void *arg) {
  long id=(long)arg;
  printf("Traad %ld: start\n", id);
  long total=0;
  for(int i=0;i<N;i++)
    total+= arr[id*N+i];
  sum[id]=total;
  printf("Traad %ld: foer: %ld\n", id, total);

  //22 Traaden skal vente her paa de andre traade

  total=0;
  for(int i=0;i<N;i++) total+=sum[i];
  printf("Traad %ld: efter: %ld\n", id, total);
  //fortsat beregning; ikke en del af opgaven.
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t t1, t2,t3,t4;
  Pthread_create(&t1, NULL, mythread, (void*)0);
  Pthread_create(&t2, NULL, mythread, (void*)1);
  Pthread_create(&t3, NULL, mythread, (void*)2);
  Pthread_join(t1, NULL);
  Pthread_join(t2, NULL);
  Pthread_join(t3, NULL);
  return 0;
}
```

1. Opstil de nødvendige erklæringer og initialisering af mutex og condition variable, samt eventuelle globale delte variable.

2. Opstil koden til synkronisering, som kan indsættes omkring linie 22

3. Beskriv kort virkemåden af din løsning, eller løsningsidé

Listing 7: Erklæringer.

```
1 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
2 pthread_cond_t waiters = PTHREAD_COND_INITIALIZER;
3 int noArrived=0;
```

Listing 8: Synkroniseringskode.

```
1
2 Pthread_mutex_lock(&mutex);
3 noArrived++;
4
5 if(noArrived==N) {
6   Pthread_cond_broadcast(&waiters);
7 } else {
8     while(noArrived<N) {
9       Pthread_cond_wait(&waiters,&mutex);
10    }
11 }
12 Pthread_mutex_unlock(&mutex);
```

Ideen er at lade en global variable optælle antallet af ankomne tråde. Sålænge antallet ikke har nået 3, skal trådene blokeres vha. `cond_wait`. Den sidst ankomne vækker alle øvrige vha `cond_broadcast`. Optælling af antallet af ankomne tråde skal ske udeleligt, og derfor beskyttes med en lås. Endvidere skal brug af wait og (bør) signal/broadcast på en condition variable ske mens låsen holdes. Vi husker at vække ventende så der ikke bliver baglås. Kaldet til broadcast sker kun når antallet af fremmødte er N, så unødvendig vækning/kald til `condbroadcast` undgåes.