Eksamen i Computer Arkitektur og Operativ Systemer

Juni 2022

- **Hovedopgavebesvarelsen skal uploades på digital eksamen.** Besvarelsen skal formatteres som en pdf fil.
- Du kan lave besvarelsen ved at pdf-annotere direkte i opgavearket (dog **ikke** med "notes", der skal clickes på for at åbne). 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 (15 pts.)

Opgave 1:

Den hypotetiske CAOSv1 computer anvender **w=15-bits** til repræsentation af heltal. Til repræsentation af signed heltal anvender den two's complement.
Antag at følgende er erklæret i et C-program:

```
int a1 = INT_MIN; //TMin_w
int a2 = INT_MAX; //TMax_w
int a3 = -8;
unsigned int b1 = UINT_MIN;
unsigned int b2 = UINT_MAX; //UMax_w
```

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

Nr.	Udtryk	Binær repræsentation	Værdi (dec.)
1	a1		
2	a2		
3	b1		
4	b2		

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	Forklaring	Værdi (dec.)
5	b2+b2		
6	a2-a3		
7	a1+a3		
8	b2+a1		
9	(-a2==a2)		
10	(a3<b1)		

Opgave 2:

1. **(5 pts)**
   Givet et 16-bit ord w erklæret i C som `unsigned short w;`. Opstil C-udtryk, der udelukkende ved hjælp af bitvise boolske og shift operationer kan
   1. uddrage de mest betydende byte af ordet w, dvs., hvis w=ABCD, skal svaret være AB.

      ```
      unsigned char x =...
      ```

   2. konstruere ordet w som er lig w hvor alle bittene er inverteret (flipped, dvs., 0 skal være 1, og 1 skal være 0).

      ```
      w = ...
      ```

2 Assembly programmer (18 pts)

Opgave 3:

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

	Udsagn	Sandt	Falsk
1	Instruktionen `movw` flytter 2 bytes.		
2	Hvis `%rax` indholder 0, og `%rdx` indeholder 8, vil instruktionen `subq %rdx,%rax` sætte sign-flag til 1		
3	`RET` instruktionen dekrementerer stack-pointeren.		
4	*guarded-do* er en metode til optimeret oversættelse af if-statements.		
5	%rbx skal gemmes af den kaldte procedure (Callee saved).		

Opgave 4:

1. **(10 pts)** I det følgende er vist et fragment af et (uoptimeret) X86-64 assembly program, som `gcc` har oversat fra et lille C-program. Funktionen `func` tager 3 parametre: første parameter `p1`, anden parameter `p2`, og tredie parameter `p3`. I oversættelsen har `gcc` valgt at allokere et antal lokale variable på stakken, som vi skal benævne `t1`, t2, t3, t4 i den rækkefølge de skrives til i assembly programmet.

Listing 1: Oversat C program

```
1  func:
2  pushq   %rbp
3  movq    %rsp, %rbp
4  movq    %rdi, -24(%rbp)
5  movl    %esi, -28(%rbp)
6  movl    %edx, %eax
7  movw    %ax, -32(%rbp)
8  movl    -28(%rbp), %eax
9  movslq  %eax, %rdx
10 movq    -24(%rbp), %rax
11 addq    %rax, %rdx
12 movswq  -32(%rbp), %rax
13 addq    %rdx, %rax
14 movq    %rax, -8(%rbp)
15 movq    -8(%rbp), %rax
16 popq    %rbp
17 ret
18
19 main:
20 ...
21 call    func
22 ...
```

Bytes (7-0)								
7	6	5	4	3	2	1	0	Adresse
								0xfa658
								0xfa650
								0xfa648
								0xfa640
								0xfa638
								0xfa630
								0xfa628
								0xfa620
								0xfa618
								0xfa610
								0xfa608
								0xfa600
								0xfa5f8
								0xfa5f0

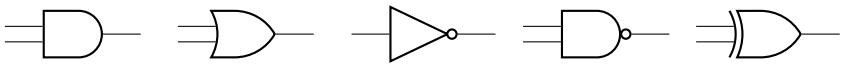
Figuren ovenfor viser desuden et tomt stykke hukommelse, hvori du skal markere et øjebliksbillede af køretidsstakken for ovennævnte program, inden instruktionen i linie 16 udføres. Antag at stackpointer registret (`%rsp`) har værdien `fa628` lige inden `CALL` instruktionen udføres. Byten på addressen `0xfa5f0` findes i sidste række, og højre kolonne, og adressen `0xfa5f7` findes i nederste række venstre kolonne, osv.

Markér i stakken ovenfor placeringen af de bytes, der bruges til de 4 lokale variable og anden data, der vedrører kaldet til `func`, så antallet af markerede bytes svarer til de anvendte ordstørrelser.
En byte markeres med flg. notation:

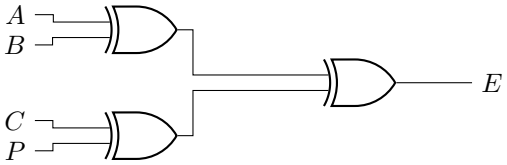
- t1, hvis den gemmer en del af indhold fra variablen t1 (samme for t2 ... t4)
- R, hvis den indeholder en del af gemt instruktionspointer register (`%rip`)
- B, hvis den indeholder en del af gemt base-pointer register (`%rbp`)

2. **(3 pts)**
Funktionen beregner et simplet udtryk. Hvilket ?

3 Processor Realisering (10 pts.)



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. **(3 pts)**
   Opstil et udtryk for output E for det kombinatoriske kredsløb i Figur 2 i boolsk algebra ved brug af de boolske operatorer & | $\sim$ $\hat{}$ (jfv. side 87 i bogen).

   $E = \ldots$

2. **(2 pts)** Givet A=B=C=0 og P=1 hvad bliver da output E ?

   $E = \ldots$

Stage	`OPq rA, rB`
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← valB OP valA Set CC
Memory	
Write back	R[rB] ← valE
PC update	PC ← valP

Stage	`subq %rdx, %rax`
Fetch	icode:ifun ← M_1[...]=... : ... rA:rB ← M_1[...]=... : ... valP ← ... = ...
Decode	valA ← R[...]=... valB ← R[...]=...
Execute	valE ← ... = ... ZF←..., SF←..., OF← ...
Memory	
Write back	R[...] ← ...
PC update	PC ← ...

Tabel 1: Beregningstrin for aritmetiske instruktioner, og instansen `subq %rdx, %rax`

Opgave 6:

1. **(5 pts)** I denne opgave antager vi processor arkitekturen SEQ Y86-64, som beskrevet i lærebogen. Spor effekten af den konkrete instruktion ved at udfylde tabellen til højre i Tabel 1 med konkrete værdier for instruktionen `subq %rdx,%rax`. Antag at instruktionen er placeret i addresse `0x400`, `%rdx` har værdien `0x10`, og `%rax` har værdien `0x0`.

4 Optimering og Instruktionsniveau parallelitet (15 pts.)

Opgave 7:

En $N \times N$ matrice A siges at være symmetrisk hvis for alle i, j gælder at $A_{i,j} = A_{j,i}$, altså elementerne på hver side af diagonalen er ens. Et eksempel er matricen M:

$$M = \begin{bmatrix} 1 & 7 & 3 & 2 \\ 7 & 4 & 5 & 6 \\ 3 & 5 & 0 & 9 \\ 2 & 6 & 9 & 10 \end{bmatrix}$$

Listing 2 viser et C program, der definerer en matrice data-struktur, samt en funktion `isSym_v1`, der checker om en matrice er symmetrisk ved at følge den matematiske definition.

1. **(4 pts)** Betragt `isSym_v1`. Analyser dens adgangsmønster til hukommelsen. Hvordan vil det ændre på programmets cache-venlighed, hvis vi lavede en isSym_v2, hvor de to **for**-løkker i linerne 22-23 blev vendt om? Anvend den svar mulighed, der gælder:
   1. `isSym_v1` er mest cache venlig
   2. `isSym_v2` er mest cache venlig
   3. Det vil ikke give nogen ændring

2. **(4 pts)** I funktionen `isSym_v5` har vi forsøgt at optimere `isSym_v1`. Præcist hvilken form for loop unrolling er anvendt? Angive $x \times y$?

3. **(7 pts)** Hvilke af nedenstående udsagn om optimeringsforsøget af `isSym_v5` er sandt eller falsk.

	Udsagn	Sandt	Falsk
1	`isSym_v1` er omskrevet så procedurekald (optimeringsblocker) i indre løkker er elimineret		
2	`isSym_v1` havde en mulighed for hukommelses-aliasering (optimeringsblocker), som er fjernet		
3	`isSym_v5` har anvendt kode-flytning.		
4	`isSym_v5` har anvendt optimeringen reduceret operator styrke		
5	`isSym_v5` gen-anvender resultat af visse del-udtryk		
6	`isSym_v5` har elimineret mange unødvendige beregninger		
7	`isSym_v5` har elimineret race-conditions i processorens pipeline		

Listing 2: En funktion og forsøg på optimering af denne.

```
1  long M[][4]={    {1,7,3,2},
2                   {7,4,5,6},
3                   {3,5,0,9},
4                   {2,6,9,10} };
5
6  typedef struct Matrix{
7    int D; //Dimension, total D*D elements
8    long *elems;//the elements;
9  } Matrix;
10
11 //returns m[i][j]
12 long get(Matrix*m,int i,int j){
13   return m->elems[m->D*i+j]; //i'th row and j'th column
14 }
15
16 void initMatrix(Matrix*m, int noElems, long * elems) {
17   m->D=noElems;  m->elems=elems;
18 }
19
20 int isSym_v1(Matrix*m){
21   int equals=1;
22   for(int i=0;i<m->D;i++){ //for all rows
23     for(int j=0;j<m->D;j++){
24       //m[i][j]==m[j][i]?
25       equals = (equals && (get(m,i,j)==get(m,j,i)));
26     }
27   }
28   return equals;
29 }
30
31 int isSym_v5(Matrix * m){
32   int equals=1;
33   const int D=m->D;
34   const int limit=D-1;
35   int i; int j;
36
37   for(i=0;i<D && equals;i+=1){
38     for( j=i+1;j<limit && equals;j+=2){
39       //M[i][j]== M[j][i])?  && M[i][j+1]==M[j+1][i] ?;
40       equals=(equals && (m->elems[i*D+j] == m->elems[j*D+i])
41                      && (m->elems[i*D+j+1] == m->elems[(j+1)*D+i]));
42     }
43     for(;j<D && equals;j++)
44      equals=(equals && (m->elems[i*D+j] == m->elems[j*D+i]));
45  }
46  return equals;
47 }
48 int main(){
49   Matrix m;
50   initMatrix(&m,4,(long*) M);
51   printf("Basic: %d\n", isSym_v1(&m));
52   printf("Optim: %d\n", isSym_v5(&m));
53 }
```

5 Operativ Systemer (10 pts.)

Opgave 8: Multiprogrammering

1. **(5 pts.)** Afkryds de egenskaber, der kendetegner multi-threading?
   1. ☐ Flere tråde deler samme process kontrol block
   2. ☐ Flere tråde deler samme adresserum
   3. ☐ Flere tråde deler samme stak
   4. ☐ Kan speede program afviklingshastigheden op på enkelt-kerne maskine
   5. ☐ En tråd kan skrive til en variabel, som en anden tråd allerede ejer en mutex-lås på

Listing 3: Et lille unix program.

```
1  int v=1;
2  int main(){
3    pid_t pid;
4    pid=fork();
5    if(pid==0){
6      v=v+2;
7      printf("A%d\n",v);
8    }
9    else {
10     printf("B%d\n",v);
11     int status;
12     wait(&status);
13     printf("C%d\n",v);
14   }
15   exit(0);
16 }
```

2. **(5 pts.)** Oplist alle mulige outputs, der kan genereres af Unix programmet i Listing 3. Antag at `printf` kører uden afbrydelse.

6 Hukommelses Organisering (15 pts.)

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

- Sidestørrelsen er 128 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	1F	1
01	0E	1
02	0D	1
03	0C	1
04	13	1
05	12	1
06	19	1
07	08	1

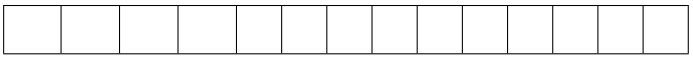
VPN	PPN	Valid
68	06	0
69	17	1
6A	18	0
6B	09	1
6C	0A	1
6D	1B	1
6E	01	1
6F	00	0

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	01	13	1	1B	0A	1	1A	0F	0
1	1A	19	0	01	03	0	00	0E	1
2	00	15	0	01	19	1	1B	01	1
3	00	0C	1	1E	11	0	1B	00	0

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

1. Virtuel Adresse: `0x37A`

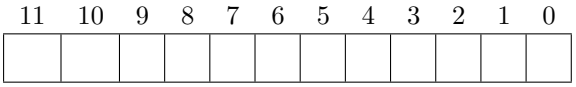
   1. Bits i Virtuelle Adresse



   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: `0x34BC`

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: `0x347F`

   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

7 Concurrency (17 pts.)

Opgave 10: **(15 pts)**
Nedenfor er skitseret en lille sekventielt program, der beregner et histogram over ordlængden i et stykke tekst, som er gemt ordvist i arrayet `input`. Histogrammet gemmes i et array `counts` således at `counts[4]` vil indholde antallet af ord af længde 4. Selve opællingen foretages af funktionen `myWorker`, mens `computeStats` præsenterer resultatet.
Vis vha. **Pthreads og semaforer**, hvordan du kan omskrive programmet til at være flertrådet, så optællingen potentielt kan ske parallelt ved, at hver tråd behandler hver sin portion ("chunk", markeret som `start` og `end` index beregnet af `getChunk`) af input, således:

- Der skal skabes **4 samtidige tråde**, hvoraf 3 workers laver optælling.
- Den 4. tråd skal afvikle `computeStats`. Den skal startes samtidigt med de øvrige tråde, men i denne version afvente, at de 3 workers afslutter deres del af optællingen, inden den påbegynder udskrift (i forberedelse til en potentiel fremtidig version, der skal kunne lave løbende opdateret statistik og udskrift - ikke en del af denne eksamen).
- Der skal være mulighed for høj grad af samtidighed.

Hint: programmet er listet som **verbatim**, så det er nemt at copy-paste ind i din virtuelle maskine til videre-udvikling. Hvis du ikke kan den eksakte syntax, kan du stadig få points med brug af **plausibel pseudo-kode**. I svarene skal du **ikke reproducere** hele programteksten; dine concurrency relaterede ændringer og deres placering skal blot fremstå entydige.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include "common.h"
#include "common_threads.h"
#define MAXWORDLEN 16 //15+1
#define WORKERS 3
#define INPUTLENGTH 57

const char* input[INPUTLENGTH]={"Man","skal", "holde","tungen","lige","i","munden","når",
"man", "skriver","programmer", "med","parallelitet", "eller","med", "en", "bedre",
"term","concurrency" ,"Der","kan","være","mange","fælder","såsom","race","conditions",
"deadlocks","og", "udsultning","Det","er","derfor","vigtigt","at","man","sætter","sig",
"grundigt", "ind","i","hvordan","synkronisering","og","gensidig","udelukkelse",
"fungerer","og", "bedst", "anvendes", "så","ens","programmer","bliver","effektive","og",
"korrekte"};

int counts[MAXWORDLEN];
//Mutex og synkroniserings erklaeringer
void initSemaphores() {}

void getChunk(long workerID, int * start, int * end){
  int chunkSize = INPUTLENGTH / WORKERS;
  *start=workerID*chunkSize;
  *end=(workerID+1)*chunkSize;
  //worker with highest id gets remaining elems
  if(workerID==WORKERS-1) *end=INPUTLENGTH;
}

void *myWorker(void *arg) {
  long id=(long)arg;
  int start=0;int end=0;
  getChunk(id,&start,&end);
  printf("traad id %ld: start %d, end %d\n",id,start,end);
  //COUNT
  for(int i=start;i<end;i++) {
    int wordLen=strlen(input[i]);
    counts[wordLen]++;
  }
  //DONE
  return NULL;
}
void *computeStats(void *arg) {
  long id=(long)arg;
  printf("traad id %ld: \n",id);
  //WAIT for Workers done
  for(int i=0;i<MAXWORDLEN;i++) {
    printf("%d:\t",i);
    for(int j=0;j<counts[i];j++) printf("#");
    printf("\n");
  }
  return NULL;
}
```

```
int main(int argc, char *argv[]) {
  //SEQUENTIAL EXECUTION
  myWorker((void*)0);
  myWorker((void*)1);
  myWorker((void*)2);
  computeStats((void*)4);

  initSemaphores();
  //START AND TERMINATE THREADS
  return 0;
}
```

1. **(4 pts.)** Opstil de nødvendige erklæringer og initialisering af semaforer, samt eventuelle globale delte variable (omkring linien markeret med `//Mutex og synkroniserings erklæringer`).

2. **(6 pts.)** Opstil koden til den opdaterede worker (mellem linierne markeret med kommentarerne `//COUNT` og `//DONE`).

3. **(4 pts.)** Opstil koden til den opdaterede opdatering af synkronisering, som kan indsættes omkring linie `//WAIT for Workers done`.

4. **(3 pts.)** Opstil koden til den opstart og terminering af trådene, som skal erstatte den sekventielle opstart, og kan indsættes omkring linien `//START AND TERMINATE THREADS`.