HØGSKOLEN I BERGEN

BERGEN UNIVERSITY COLLEGE

EKSAMENSOPPGAVE

Emnekode: DAT103

Emnenavn: Datamaskiner og operativsystemer

Utdanning/kull/klasse: 2. klasse data/informasjonsteknologi

Dato: 17. desember 2014

Eksamensform: Skriftlig

Eksamenstid: 9.00 – 13.00 (4 timer)

Antall eksamensoppgaver: 4

Antall sider (inkludert denne): 15

Antall vedlegg: Ingen

Tillatte hjelpemidler: Enkel kalkulator

Fagansvarlig: Henrik Lieng og Atle Geitung

Merknader: Ingen

Oppgave 1 – Aritmetiske og logiske operasjoner (25 %)

Oppgave A (45 % av oppgave 1)

Spørsmålene i denne oppgaven er av typen *flervalg*, der bare et alternativ er korrekt. Angi det alternativet du mener er korrekt.

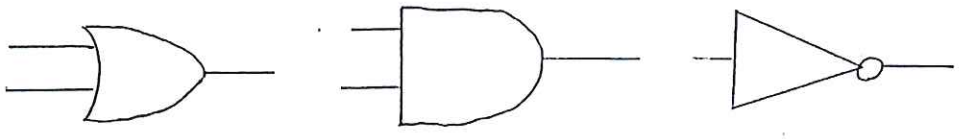
NB: hvis du mener alternativene for et spørsmål er tvetydige, kan du angi ekstra informasjon i besvarelsen din. Dette blir tatt i betraktning hvis du angir feil alternativ. Du kan fortsatt bare angi et alternativ og det gis ikke bonuspoeng for ekstra informasjon hvis du har angitt korrekt alternativ. Dette gjelder alle flervalgsspørsmålene i denne eksamenen.

i. Hva er hovedformålet til ALU enheten (Arithmetic-Logic Unit) og hvilket tallsystem bruker den for å representere tall?
   a. ALU enheten utfører aritmetiske og logiske operasjoner. Den baserer seg på binære tall.
   b. ALU enheten utfører aritmetiske og logiske operasjoner. Den baserer seg på heksadesimale tall.
   c. ALU enheten utfører aritmetiske og logiske operasjoner, men IKKE boolske operasjoner. Den baserer seg på binære tall.

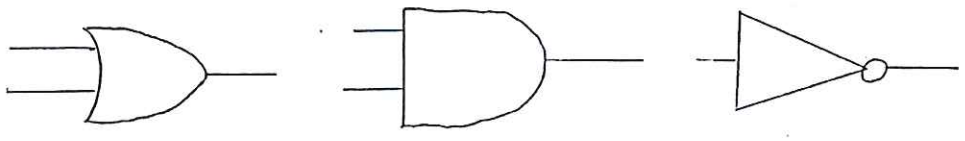
ii. Merk av det alternativet under som angir en gyldig representasjon av negative tall. Representasjonen skal være slik at den kan enkelt implementeres på en datamaskin.
   a. -1010
   b. 1010, der den mest signifikante biten angir om tallet er negativt (1 betyr negativ).
   c. Det er umulig å representere negative tall direkte. Et spesifikt flagg (Negative Flag – NF), som er implementert ved et register, angir om et tall er negativt.

iii. Boolsk algebra er definert (aksiomatisk) ved tre fundamentale operasjoner. Hvilke?
   a. OG (AND), ELLER (OR), og IKKE (NOT).
   b. Addisjon, multiplikasjon, og negasjon. OG, ELLER, og IKKE er automatisk definert fra addisjon, multiplikasjon, og negasjon operasjonene.
   c. IKKE-OG (NAND), IKKE-ELLER (NOR), og IKKE (NOT). AND og OR er definert fra NAND og NOR.

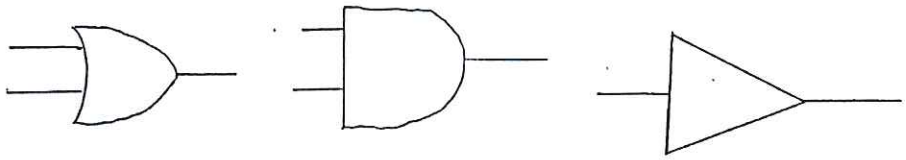
iv. Angi de grafiske symbolene for OG (AND), ELLER (OR), og IKKE (NOT).
   a. Symboler, i rekkefølgen ELLER, OG, og IKKE:



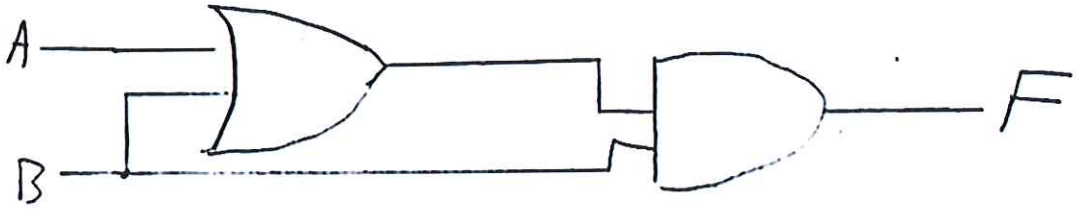
   b. Symboler, i rekkefølgen OG, ELLER, og IKKE:



c. Symboler, i rekkefølgen ELLER, OG, og IKKE:



v. Angi sannhetstabellen for følgende krets:



a. Korrekt tabell:

A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

b. Korrekt tabell:

A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

c. Korrekt tabell:

A	B	F
0	0	0
1	0	0
0	1	1
1	1	1

vi. Angi sannhetstabellen for følgende uttrykk: $F = A \cdot B + A \cdot \bar{B}$

a. Korrekt tabell:

A	B	F
0	0	0
1	0	0
0	1	1
1	1	1

b. Korrekt tabell:

A	B	F
0	0	0
1	0	1
0	1	0
1	1	1

c. Korrekt tabell:

A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

Oppgave B (15 % av oppgave 1)

Forklar kort (maksimalt fem setninger) hvordan datamaskinen utfører subtraksjonsoperasjonen, der tall er representert ved toerkomplement representasjonen.

Bruk fremgangsmåten du har angitt til å utføre følgende tre operasjoner:

$$0111 - 0100$$

$$00000111 - 00001010$$

$$0111 - 0111$$

Merk at tallene over er binære tall. Ved titallsrepresentasjon, skal du dermed utføre 7-4, 7-10, og 7-7. Resultatet skal være angitt ved toerkomplement representasjonen.

Oppgave C (15 % av oppgave 1)

Bruk Kernaughdiagram til å forenkle følgende uttrykk (hvis mulig):

$$F = \bar{A}B + AB$$

$$F = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$$

$$F = AB\bar{C}\bar{D} + \bar{A}\bar{B}CD + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D}$$

Oppgave D (15 % av oppgave 1)

Vi har angitt følgende diagrammer:

Diagram 1:

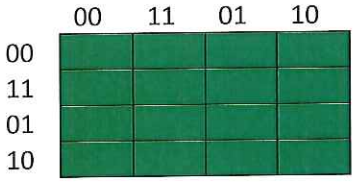


Diagram 2:

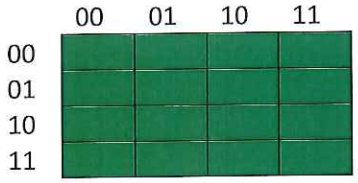
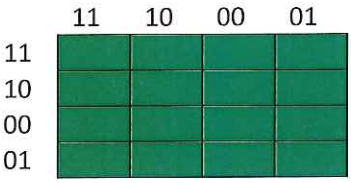


Diagram 3:



Hvorfor kan grupperinger i Kernaughdiagrammer forenkles til enklere utrykk?

Diagrammene over viser potensielle oppsett for Kernaugh-basert forenkling. Minst et av diagrammene er ugyldig og minst et er gyldig. Angi hvilke av diagrammene som er gyldige og hvilke som er ugyldige Kernaughdiagrammer. Begrunn.

Oppgave E (10 % av oppgave 1)

Bevis følgende tre identiteter:

$$A \cdot \bar{B} + A \cdot B = A$$

$$A + A = A$$

$$A \cdot A = A$$

Disse identitetene kan bevises direkte fra aksiomene. Husk at boolsk algebra er definert ved følgende aksiomer:

Assosiativitet: $A + (B + C) = (A + B) + C$

$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

Kommutativitet: $\mathrm{A} + \mathrm{B} = \mathrm{B} + \mathrm{A};\ \mathrm{A} \cdot \mathrm{B} = \mathrm{B} \cdot \mathrm{A}$

Absorpsjon: $\mathrm{A} + (\mathrm{A} \cdot \mathrm{B}) = \mathrm{A};\ \mathrm{A} \cdot (\mathrm{A} + \mathrm{B}) = \mathrm{A}$

Identitet: $\mathrm{A} + 0 = \mathrm{A};\ \mathrm{A} \cdot 1 = \mathrm{A}$

Distributivitet: $\mathrm{A} + (\mathrm{B} \cdot \mathrm{C}) = (\mathrm{A} + \mathrm{B}) \cdot (\mathrm{A} + \mathrm{C})$

$\mathrm{A} \cdot (\mathrm{B} + \mathrm{C}) = (\mathrm{A} \cdot \mathrm{B}) + (\mathrm{A} \cdot \mathrm{C})$

Komplement: $\mathrm{A} + \overline{\mathrm{A}} = 1;\ \mathrm{A} \cdot \overline{\mathrm{A}} = 0$

Oppgave 2 – Programkontroll (25 %)

Oppgave A (45 % av oppgave 2)

Spørsmålene i denne oppgaven er av typen *flervalg*, der bare et alternativ er korrekt. Angi det alternativet du mener er korrekt.

i. Angi et typisk instruksjonsformat som datamaskinen kan relativt enkelt tolke.
   a. En instruksjon kan dekodes i en streng med bits. For eksempel, kan et ord (word) brukes som en 16 bit instruksjon: [type(4 bits) operand1(6 bits) operand2(6 bits)].
   b. Instruksjoner til datamaskinen gjøres enklest via programmeringsspråk, som Java og C++. En enkel instruksjon i Java: System.out.println("Hello World! ");
   c. Direkte instruksjoner til datamaskinen kan bare gjøres i assembly. Eksempel: add a,b; vil fortelle datamaskinen at den skal legge sammen verdiene relatert til a og b, og lagre resultatet i a.

ii. Datamaskinen støtter ulike adresseringsmoduser. Hva er forskjellen mellom umiddelbar, direkte, og indirekte adresseringsmodus?
   a. **Umiddelbar**: instruksjonen blir utført umiddelbart, uten at maskinen gjør minnereferanse.
      Direkte: instruksjonen blir utført direkte, uten at noen minnereferanser blir utført.
      Indirekte: instruksjonen blir utført indirekte, der maskinen trenger å hente data fra minne for å fullføre instruksjonen.
   b. **Umiddelbar**: verdien er angitt direkte i instruksjonen.
      Direkte: verdien er lokalisert i minne. Instruksjonen angir den relaterte adressen.
      Indirekte: verdien er lokalisert i minne. Instruksjonen angir en lokasjon i minne som inneholder adressen til den faktiske verdien.
   c. **Umiddelbar**: angir at maskinen skal gå i umiddelbar modus. Dette vil angi høyeste prioritet til kommende instruksjoner (de blir utført umiddelbart).
      Direkte: angir at maskinen skal gå i direkte modus. Instruksjonene får lavere prioritet, men blir utført direkte av maskinen.
      Indirekte: angir at kommende instruksjoner skal utføres indirekte. Instruksjonene kan dermed bli utført i et klynget system (cluster).

iii. Hva er hovedformålet med registre som brukes som flagg?
   a. Dette er generelle boolske registre, som kan brukes når som helst og opererer på boolske variabler.
   b. Flagg brukes hovedsakelig til synkronisering av prosesser og tråder. For eksempel, kan Peterson sin løsning bli implementert ved slike flagg.
   c. Flagg brukes stort sett som betingede koder. Verdiene til et flagg settes som et resultat av en operasjon (f.eks. CMP instruksjonen vil sette ulike flagg basert på verdiene til operandene).

iv. Hva inneholder programtelleren (program counter)?
   a. Programtelleren inneholder adressen til neste instruksjon som skal hentes.
   b. Programtelleren inneholder ID til nåværende program som blir utført.
   c. Programtelleren inneholder adressen til nåværende instruksjon som utføres.

v. Hva er poenget med å utføre instruksjoner i parallell (instruction pipelining)?
   a. Å utnytte de ulike komponentene i CPU'en mer optimalt og effektivt.

b. Å øke hastigheten på utførelsen av minnereferanser.
c. Å utnytte moderne CPU'er, med flere CPU kjerner, mer optimalt og effektivt.

vi. Løkkebufferen er en liten og hurtig buffer i CPU'en som inneholder et gitt antall instruksjoner. Disse er de siste instruksjonene som har blitt utført. Hva er fordelen med denne bufferen angående betinget forgreining (conditional branching)?
   a. Det er ingen fordeler med denne bufferen siden instruksjonen allerede ligger i hovedminne.
   b. Hvis en forgreining er relatert til en instruksjon i nær fortid, er instruksjonen i forgreiningen sannsynligvis i bufferen. Tiden det tar å hente denne instruksjonen vil derfor minke.
   c. Generelt, vil tiden for å hente instruksjoner som nylig har blitt utført minke. Det er imidlertid umulig å forhåndshente instruksjoner relatert til betingelser. Løkkebufferen vil derfor ikke være til hjelp for instruksjoner relatert til betingete forgreininger.

vii. Tekst segmentet (text segment) er en av de fundamentale delene av det virtuelle adresserommet til et program. Hva inneholder dette segmentet?
   a. Tekst relatert til filer i systemet.
   b. Kjørbare instruksjoner.
   c. Konstante data og variabler.

Oppgave B (15 % av oppgave 2)

Beskriv de ulike stegene i instruksjonssyklusen. Du må ta med minst 4 hovedsteg.

Merk at en fullstendig beskrivelse av instruksjonssyklusen vil ta for seg 10 ulike oppgaver. Du trenger bare en kort beskrivelse av de ulike oppgavene (1 til 2 setninger).

Oppgave C (15 % av oppgave 2)

Konverter følgende assembly kode til Java (eller pseudo) kode:

```
mov ax,1
mov bx,0
hopp:
mov cx, [variabel_a]
cmp bx,cx
jnz hopp
mov [variabel_a], ax
mov [variabel_b], byte 5
mov [variabel_a], bx
```

Oppgave D (15 % av oppgave 2)

Konverter pseudo koden under til assembly kode.

```
ax = 5;
cx = 7;
dx = 0;
bx = 0;
while(dx == 0) {
   if(bx > ax && bx < cx) {
      dx = 1;
   } else {
      bx++;
   }
}
```

Du kan anta at vi bare arbeider med registre her og du trenger dermed ikke bruke variabler lagret i minne.

Oppgave E (10 % av oppgave 2)

Konverter assembly koden under til maskinkode. Angi eventuelle antagelser du tar, som for eksempel instruksjonsformat. Tips: oppgaven kan løses med et 16 bit instruksjonsformat.

```
hopp:
mov cx, [variabel_a]
cmp bx,cx
jnz hopp
mov [variabel_a], ax
mov [variabel_b], variabel_c
mov [variabel_a], bx
```

Oppgave 3 – Prosesser og tråder (25 %)

Oppgave A (45 % av oppgave 3)

Spørsmålene i denne oppgaven er av typen *flervalg*, der bare et alternativ er korrekt. Angi det alternativet du mener er korrekt.

i. Hva er en prosess?
   a. En instruksjonssekvens for å løse en gitt oppgave.
   b. Et program under utføring.
   c. Et kjørende program i CPU.

ii. Hva er hensikten med kontrollblokken (Process Control Block - PCB) til en prosess?
   a. Å kontrollere funksjonaliteten til prosessen.
   b. Å holde på midlertidig data relatert til prosessen.
   c. Å lagre utelukkende verdier relatert til CPU registre for å muliggjøre kontekstbytte.

iii. Hva er forskjellen mellom langtidsplanleggeren og korttidsplanleggeren?
   a. Langtidsplanleggeren bestemmer prosesser som blir angitt en lang tid (time slot) på CPU'en. Korttidsplanleggeren bestemmer prosesser som blir angitt en kort tid på CPU'en.
   b. Langtidsplanleggeren laster inn jobber som skal kjøres til hovedminne. Korttidsplanleggeren bestemmer hvilken prosess som skal kjøres på CPU'en.
   c. Begge planlegger hvilke prosesser som skal kjøres på CPU'en. Forskjellen er at korttidsplanleggeren blir kjørt minst hver 100. millisekund, mens det kan ta minutter mellom langtidsplanleggerens kjøringer.

iv. Angi potensiell utskrift til standard ut ved kjøring av følgende kodesnutt:

```
pid = fork();
if(pid == 0)
   printf("Hello");
fork();
fork();
printf("1");
```

   a. Hello111
   b. HelloHello11111111
   c. Hello11111111

v. Hva er en zombie prosess?
   a. En terminert prosess som tar over adresseområdet til en eksisterende prosess.
   b. En prosess som har angitt exit(), men har ikke blitt terminert.
   c. En barneprosess som har blitt terminert, men forelder prosessen har ikke kalt funksjonen wait().

vi. Hvorfor er ressursdeling mellom tråder enklere sammenliknet med å bruke flere prosesser?
   a. Fordi tråder bruker et forenklet meldingssystem for å dele ressurser.
   b. Fordi tråder relatert til en prosess deler automatisk minne og ressurser.
   c. Fordi tråder deler kontrollblokken. Dvs. at midlertidig data, som data relatert til ressurser, deles automatisk.

vii. Hvordan muliggjør moderne operativsystemer kjøring av brukertråder samtidig?
   a. Ved å håndtere brukertråder som standard prosesser.
   b. Ved å avbilde brukertråder til kjernetråder som planlegges av OS.

c. Ved å avbilde alle brukertråder til en kjernetråd.

viii. Hva er hovedpoenget med trådbasseng?

a. Å forenkle bruken av tråder for applikasjonsutviklere.
b. Trådbasseng øker ytelsen til systemet.
c. Trådbasseng gjør det mulig å håndtere flere tråder samtidig.

Oppgave B (15 % av oppgave 3)

I prosessplanlegging, vil *ventetid* angi tiden en prosess har ventet i klarkøen. Angi gjennomsnittlig ventetid for prosessene i tabellen under ved FCFS (First-Come, First-Served) og SJF (Shortest-Job-First) algoritmene. Vi antar at SJF bruker erstatning (den er "preemptive"); dvs. at vi bruker SRTF (Shortest-Remaining-Time-First). Vis utregningen din i besvarelsen.

Prosess ID	Ankomsttid	Utføringstid
0	0	3
1	2	3
2	4	15
3	6	5
4	7	4

Ankomsttiden er det tidspunktet prosessen ankommer klarkøen og utføringstiden (burst time) er den tiden prosessen trenger å kjøre i CPU.

Oppgave C (15 % av oppgave 3)

I prosessplanlegging, vil *behandlingstid* angi tiden det tar å utføre en gitt prosess. Angi gjennomsnittlig behandlingstid for prosessene i tabellen under.

Som planleggingsalgoritme bruker vi flernivå tilbakeleveringskø, med følgende nivåer:

1. Prioritet med Round-Robin, der:
   - valg av prosess bestemmes ved hjelp av prioritet
   - lave verdier har høyere prioritet
   - kjøretid i CPU er begrenset med en Round-Robin tidskvant, q=3.
2. Round-Robin, med tidskvant q=6 for alle prosesser
3. FCFS.

Vi antar at alle prosesser ankommer klarkøen ved samme tidsenhet (tid=0).

Prosess ID	Prioritet	Utføringstid
0	0	10
1	1	5
2	2	15
3	3	20
4	4	4

Oppgave D (15 % av oppgave 3)

Angi utskriften av følgende kodesnutt:

```
tall = 0; // global variabel av typen int
pid = fork();
if(pid == 0) {
   pthread_create(thread, attributes, runner, NULL);
   pthread_join(thread,NULL);
   printf("A: %d\n ",tall);
} else if(pid > 0) {
   printf("B: %d\n ",tall);
}
} // slutt på metode
void *runner(void *param) { // metode relatert til ny tråd
   tall = 5;
   pthread_exit(0);
}
```

Begrunn svaret ditt.

Hva er forskjellen mellom `pthread_create` og `fork()` med tanke på minnehåndtering?

Oppgave E (10 % av oppgave 3)

Angi pseudo og/eller C kode for følgende metode:

```
opprettProsesser(int antall, char* program)
```

Metoden skal opprette et gitt antall (`antall`) nye prosesser (ved å bruke fork).

De nye prosessene skal få sine adresseområder overskrevet med et program angitt med `program`.
Du kan anta at følgende kommando gjør denne operasjonen: `exec(program);`

Begrunn hvorfor koden din produserer nøyaktig det angitte antall prosesser.

Du kan anta at forelder prosessen (som kaller `opprettProsesser`) ikke trenger å vente på at barneprosessene avslutter.

Oppgave 4 – Samtidighet og synkronisering (25 %)

Oppgave A (45 % av oppgave 4)

Spørsmålene i denne oppgaven er av typen *flervalg*, der bare et alternativ er korrekt. Angi det alternativet du mener er korrekt.

i. Hva menes med kappløp (race condition) i forbindelse med samtidige prosesser?
   a. Samtidige prosesser som "kapper" om globale eller delte data, som for eksempel mutex låser.
   b. Dette er en betingelse mht. samtidige prosesser: for at prosesser skal kunne kjøre samtidig, må de "kappe" om CPU kjerner.
   c. Dette angår prosesser eller tråder som arbeider på samme oppgave, der de samtidige prosessene eller trådene kapper om å finne korrekt svar først. Et eksempel er søk i et større bibliotek, der tråder kan kappe om å finne et gitt element.

ii. En betingelse for korrekt løsning på "The Critical-Section" problemet er at prosesser eller tråder må oppnå gjensidig eksklusjon (mutual exclusion) i en kritisk seksjon. Hva menes med dette?
   a. Prosesser må ekskludere andre prosesser med å gå inn i kritiske seksjoner. Dvs. at alle prosesser må ekskludere alle andre prosesser med å gå inn i en kritisk seksjon.
   b. Bare en prosess eller tråd kan utføre instruksjoner i en kritisk seksjon om gangen.
   c. Hvis en prosess får tilgang til en kritisk seksjon, kan den for alltid forhindre andre prosesser å gå inn i den samme kritiske seksjonen.

iii. Hvorfor trenger vi typisk støtte fra maskinvaren når vi synkroniserer tilstandsbaserte tråder?
   a. Maskinvaren kan garantere at operasjoner på delte data kan utføres atomisk.
   b. Bare maskinvaren kan korrekt synkronisere tråder ved hjelp av signaler.
   c. Bare maskinvaren kan korrekt synkronisere tråder ved hjelp av flagg.

iv. Hvordan kan mutex låser bli brukt for å garantere at bare en tråd har tilgang til en kritisk seksjon om gangen?
   a. Å relatere en mutex lås til hver tråd. Tråden som først låser den kritiske seksjonen med låsen sin, vil ha eksklusiv tilgang til seksjonen.
   b. Å relatere en mutex lås til den kritiske seksjonen. Den tråden som har tilgang til låsen, har automatisk tilgang til den kritiske seksjonen. Andre tråder venter i en ventekø til låsen blir ledig.
   c. Å relatere en mutex lås til den kritiske seksjonen. Den tråden som har tilgang til låsen, har automatisk tilgang til den kritiske seksjonen. Andre tråder spinner i CPU til låsen blir ledig.

v. Hva er *hovedforskjellen* mellom mutex låser og semaforer?
   a. Mutex låser vil eksplisitt garantere atomiske operasjoner, ved bruk av LOCK. På den andre siden, vil semaforer implisitt garantere atomiske operasjoner, ved bruk av wait().
   b. Hvis en lås ikke er ledig, vil mutex låser tvinge forespørrende tråder til å spinne i en while løkke i CPU'en, mens semaforer vil tvinge tråder i søvn (sleep).
   c. Mutex låser kan bare implementeres i assembly, mens semaforer kan implementeres i høy-nivå språk som Java.

vi. Hvorfor putter moderne OS hode i sanden mht. vranglås? (dvs: moderne OS lar vranglås skje og gjør ingen ting for å forhindre det; hvorfor?)
   a. Vranglås kan verken oppdages eller forebygges. OS må derfor la vranglås skje. Det er programutvikler sin oppgave å unngå vranglås.
   b. Algoritmer for håndtering av vranglås krever enten masse ressurser eller kunnskap om systemet (som hvor mange ressurser prosesser trenger). Disse ulempene overstyrer fordelene med tanke på håndtering av vranglås i OS.
   c. Algoritmene for håndtering av vranglås garanterer ikke korrekt håndtering i alle tilfeller. Mer forskning på dette emne er nødvendig for at utviklere av moderne OS velger å implementere korrekt håndtering av vranglås.

vii. Hva er forskjellen mellom deadlock (vranglås) og livelock?
   a. Ved livelock, kjører tråder i CPU, men de gjør ingen progresjon. Ved deadlock, vil tråder vente på hverandre i en syklus.
   b. Ved livelock, vil tråder avbrytes hvis de har ventet for lenge. Dette vil normalt ikke skje ved deadlock.
   c. Ved livelock, vil tråder fortsette å kjøre hvis de kommer i vranglås. Ved deadlock, vil trådene bli drept hvis de kommer i vranglås.

Oppgave B (15 % av oppgave 4)

Angi ressurs-allokeringsgrafen for følgende fem tråder og tre ressurser:

	A	B	V
	X Y Z	X Y Z	X Y Z
T0	0 1 0	0 0 0	0 0 0
T1	2 0 0	1 1 0	
T2	1 0 0	0 1 0	
T3	2 1 0	1 0 1	
T4	0 0 1	0 0 1	

I tabellen er hver rad (T0 til T4) relatert til en tråd (foruten kolonnen til høyre). Det er tre ressurser X, Y, og Z.

Kolonnen relatert til A angir antall ressurser som brukes av en gitt tråd. For eksempel, bruker T0 null instanser av ressurs X, en instans av ressurs Y, og null instanser av ressurs Z.

Kolonnen relatert til B angir antall ressurser som en gitt tråd forespør. For eksempel, forespør T1 en instans av ressurs X, null instanser av ressurs Y, og en instans av ressurs Z.

Kolonnen relatert til V angir antall tilgjengelige instanser av en gitt ressurs. I dette tilfellet er det ingen ledige ressurser.

Oppgave C (15 % av oppgave 4)

Bruk algoritmen som har blitt gjennomgått i faget for å finne ut som systemet angitt i forrige oppgave er i vranglås. Fremgangsmåten skal kort beskrives. Angi arbeidsvektoren din for hver iterasjon.

Oppgave D (15 % av oppgave 4)

Koden under angir en løsning på "Readers-Writer" problemet ved bruk av semaforer. Konverter denne løsningen til en monitor. Koden skal være av type pseudo og/eller C.

Tips: neste side viser en mal for en typisk monitor.

```
Globale data:
semafor rw_CS = 1;
semafor CS = 1;
int read_count = 0;
Kode for WRITER:
do {
   wait(rw_CS);
   write data
   signal(rw_CS);
} while(true);
Kode for READER:
do {
   wait(CS);
   if(read_count==0)
      wait(rw_CS);
   read_count++;
   signal(CS);
   read DATA
   wait(CS);
   read_count--;
   if(read_count==0)
      signal(rw_CS);
   signal(CS);
} while(true);
```

Oppgave E (10 % av oppgave 4)

Pseudokoden på neste side angir en ukorrekt implementasjon av Producer-Consumer problemet. Gjør minimale modifikasjoner, med referanser til linjenumre, for å gjøre koden korrekt i henhold til en signal-and-wait monitor. Begrunn.

```
1: Monitor ProducerConsumer {
2:     int in, out, buff[N];
3:     condition full, empty;
4:     { in = out = 0;} // init
5:     function produce(item) {
6:         if( (in-out) == 0 ) wait(full);
7:         buff[in] = item;
8:         if( (in+1)%N == out ) signal(empty);
9:         in = (in + 1) % N;
10:    }
11:    function consume() {
12:        if( (in+1)%N == out ) wait(empty);
13:        item = buff[out];
14:        if( (in-out) == 0 ) signal(full);
15:        out = (out + 1) % N;
16:    }
17: }
```