DAT 103
Datamaskiner og operativsystemer (Computers and Operating Systems)

Øvelse 2 – Digitale kretser, Addresser og Assembly – **Hele øvelsen er obligatorisk**

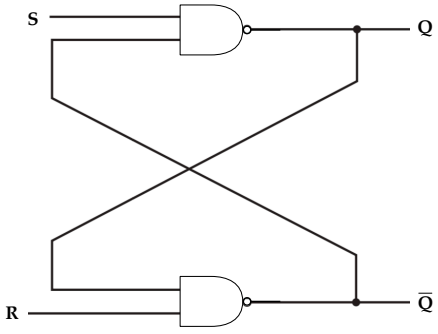
Dette er en øvelse hvor alle oppgavene er obligatoriske med innlevering. Hjelp til øvingen gis på laben i uke 39 – 41. Krav til den obligatoriske øvelsen:

- Dere kan arbeide sammen i grupper på inntil 4 personer.
- Gruppen leverer samlet på Canvas.
- Innleveringsfrist er **Mandag, 12. oktober, kl 12:00**.
- Dere må løse alle oppgavene.

Innhold

1 Digitale kretser

Følgende figur illustrerer en implementering av en S-R-lås ved bruk av NAND-porter i stedet for NOR-porter.



Ta utgangspunkt i følgende tabell som viser serien med inngangsverdier for S og R over tid t. Fullfør tabellen ved å fylle inn verdiene til Q.

t	0	1	2	3	4	5	6	7
S	0	1	1	1	0	1	1	0
R	1	0	1	0	1	1	0	0
Q								

2 Adresser

1. Ta utgangspunkt i en prosessor som henter og utfører en instruksjon som krever beregning av en enkelt operand. Angi antall hukommelsetilganger som kreves for hver av følgende adresseringsmoduser.

   (a) Immediate (Umiddelbar)
   (b) Direct (Direkte)
   (c) PC relative (PC relativ)
   (d) Register

2. Ta utgangspunkt i en datamaskin der data som vises til høyre opptrer i hovedhukommelsen, som starter på adresse 100. Anta at prosessoren utfører instruksjonen i adresse `100`. `AM` i instruksjonen angir en adresseringsmodus, og angir (om nødvendig) et referanseregister `RR`. Anta at når det brukes, inneholder `RR` verdien 199. Det finnes også et basisregister `BR` som inneholder verdien 99.

   Skriv opp den effektive adressen og operanden som skal lastes for hver av følgende adresseringsmoduser

   (a) Immediate 200
   (b) Direct 200
   (c) Base-register 200
   (d) PC-relative 200
   (e) Indirect 200
   (f) Register `RR`
   (g) Register indirect `RR`

100	`Load`	`AM`
101	Next Instruction	
⋮	⋮	
199	1000	
200	400	
201	401	
⋮	⋮	
299	899	
300	900	
301	901	
⋮	⋮	
399	999	
400	1000	
401	1001	
⋮	⋮	

3 Assembly programmering

Oppsett

I denne øvingen anbefaler jeg at dere bruker NASM (Netwide Assembler) for å oversette x86 assembly-kode til maskinkode. Det enkleste er å gjøre denne oppgaven i en virtuell Linux-maskin, for eksempel, Ubuntu.

Det enkelste er å gjøre dette fra kommandolinjen i bash. NASM er tilgjengelig i Ubuntu sine repositories, så det er bare å gi kommandoen: `sudo apt-get install nasm` . Dette vil installere siste versjon av NASM.

NASM oversetter assembly-kode til relokerbar maskinkode som ikke er kjørbar. For å gjøre denne kjørbar må vi lenke sammen all relokerbar kode til utførbart programkode. Dere vil normalt bare ha en fil, så dette kan virke litt unødvendig, men slik er prosessen for å lage et program. For å lenke bruker dere `ld` som allerede finnes i Linux og skal være tilgjengelig for bruk. For å sjekke om den finnes, kan du gi kommandoen `whereis ld` .

For å se hva som skjer under utføring, skal vi kjøre programmet i en debugger. Vi skal bruke gdb (Gnu Debugger `www.gnu.org/software/gdb`). Denne skal også være tilgjengelig på Linux.

Oppgave 1 – Hello World

Lag en fil `hello.asm` med en tekst-editor og skriv inn følgende innhold:

```
; Program som skriver ut Hello World

; Konstanter
   cr equ 13      ; Vognretur
   lf equ 10      ; Linjeskift

section .data     ; Datasegment
   melding db 'Hello World!',cr,lf
   lengde equ $ - melding

section .text      ; Kodesegment
global _start
_start:
   mov edx,lengde
   mov ecx,melding
   mov ebx,1
   mov eax,4       ; sys_write
   int 80h
   mov ebx,0
   mov eax,1       ; sys_exit
   int 80h
```

Denne koden må først assembleres (oversettes til maskinkode), med `nasm`, og deretter lenkes, med `ld`.

- Kommando for å oversette til relokerbar maskinkode og samtidig generere debug-informasjon:

  `nasm -f elf -F dwarf -g hello.asm`

- Denne kommandoen lager filen `hello.o` som inneholder maskinkoden. For å gjøre filen kjørbar må den lenkes med `ld`:

  `ld -m elf_i386 -o hello hello.o`

  Lenkeren lager et kjørbart program med navnet `hello`. Kjør det og se om det virker.

- Til slutt kjører du det ved hjelp av debug-er:

  `gdb -tui hello`

Når `gdb` er startet, gir du kommandoen `layout regs` for å se registrene. Så setter du et bruddpunkt med kommandoen `b _start` og starter programmet ditt med kommandoen `r`. Programmet vil nå starte og stopper på det bruddpunktet du har satt. Kjør det stegvis med kommandoen `s` til det er ferdig. Legg merke til registrenes verdier underveis.

Slå opp i man og finn forklaring til alle opsjonene som er brukt i eksempelet. Du skal levere kildekoden din, bilde av debug-ingen, og forklaringen av opsjoner.

Oppgave 2 – Summer og skriv ut tall

Nedenfor er vist et enkelt assemblyprogram.

Kompiler og test ut programmet. Programmet leser to sifre adskilt med ett eller flere mellomrom. Programmet summerer tallene og skriver ut summen. Svaret vil kun stemme for summer mindre enn `10` da programmet kun skriver ut ett siffer i svaret.

```
; Inndata Programmet leser inn to sifre skilt med ett eller flere mellomrom
; Utdata Programmet skriver ut summen av de to sifrene,
; forutsatt at summen er mindre enn 10.

; Konstanter
    cr equ 13 ; Vognretur
    lf equ 10 ; Linjeskift
    SYS_EXIT  equ 1
    SYS_READ  equ 3
    SYS_WRITE equ 4
    STDIN     equ 0
    STDOUT    equ 1
    STDERR    equ 2

; Datasegment
section .bss
    siffer resb 4

; Datasegment
section .data
    meld db "Skriv to ensifrede tall skilt med mellomrom.",cr,lf
       db "Summen av tallene maa vaere mindre enn 10.",cr,lf
    meldlen equ $ - meld
    feilmeld db cr,lf, "Skriv kun sifre!",cr,lf
    feillen equ $ - feilmeld
    crlf db cr,lf
    crlflen equ $ - crlf

; Kodesegment med program
section .text

global _start
_start:
    mov edx,meldlen
    mov ecx,meld
    mov ebx,STDOUT
    mov eax,SYS_WRITE
    int 80h
```

```
    ; Les tall, innlest tall returneres i ecx
    ; Vellykket retur dersom edx=0
    call lessiffer
    cmp edx,0 ; Test om vellykket innlesning
    jne Slutt ; Hopp tilavslutning ved feil i innlesing
    mov eax,ecx ; Første tall/siffer lagres i reg eax

    call lessiffer
    ; Les andre tall/siffer
    ; vellykket: edx=0, tall i ecx
    cmp edx,0 ;Test om vellykket innlesning
    jne Slutt
    mov ebx,ecx ; andre tall/siffer lagres i reg ebx

    call nylinje
    add eax,ebx
    mov ecx,eax
    call skrivsiffer ; Skriv ut verdi i ecx som ensifret tall

Slutt:
    mov eax,SYS_EXIT
    mov ebx,0
    int 80h

; -------------------------------------------------------
skrivsiffer:
    ; Skriver ut sifferet lagret i ecx. Ingen sjekk på verdiområde.
    push eax
    push ebx
    push ecx
    push edx
    add ecx,'0' ; converter tall til ascii.
    mov [siffer],ecx
    mov ecx,siffer
    mov edx,1
    mov ebx,STDOUT
    mov eax,SYS_WRITE
    int 80h
    pop edx
    pop ecx
    pop ebx
    pop eax
    ret

; -------------------------------------------------------
lessiffer:
    ; Leter forbi alle blanke til neste ikke-blank
    ; Neste ikke-blank returneres i ecx
    push eax
    push ebx

Lokke:
    ; Leser et tegn fra tastaturet
    mov eax,SYS_READ
    mov ebx,STDIN
    mov ecx,siffer
    mov edx,1
    int 80h
    mov ecx,[siffer]
```

```
    cmp ecx,' '
    je Lokke
    cmp ecx,'0' ; Sjekk at tast er i område 0-9
    jb Feil
    cmp ecx,'9'
    ja Feil
    sub ecx,'0' ; Konverter ascii til tall.
    mov edx,0 ; signaliser vellykket innlesning
    pop ebx
    pop eax
    ret ; Vellykket retur

Feil:
    mov edx,feillen
    mov ecx,feilmeld
    mov ebx,STDERR
    mov eax,SYS_WRITE
    int 80h
    mov edx,1 ; Signaliser mislykket innlesning av tall
    pop ebx
    pop eax
    ret ; Mislykket retur

; ------------------------------------------------------------
; Flytt cursor helt til venstre på neste linje
nylinje:
    push eax
    push ebx
    push ecx
    push edx
    mov edx,crlflen
    mov ecx,crlf
    mov ebx,STDOUT
    mov eax,SYS_WRITE
    int 80h
    pop edx
    pop ecx
    pop ebx
    pop eax
    ret

; End _start
```

Dere skal i denne oppgaven endre programmet slik at programmet også fungerer om summen av de to sifrene blir større enn `10`.

Eksempel på kjøring kan være:

```
sumnumber
6 8
14
```

Summen av to heltall som hver er på ett siffer hver, blir maksimalt `18`. Det betyr at programmet må skrive ut to sifre i svaret. Det enkleste er at dere alltid skriver resultatet som et tosifret tall i området `00` til og med `18`.

Kompiler og kjør programmet i en debug-er. Legg spesielt merke til programteller og stabel i forbindelse med kall til subrutinene.

Oppgave 3 – Oversett Java-kode til assembly-kode

Oversett følgende Java-kode til assembly-kode.

```
/**
 * Kode som gjør noe mystisk.
 *
 * @param args ikke i bruk
 */
public static void main(String[] args) {
    int a = 0;
    for (int i = 0; i < 20; i++) {
        if (i < 10) {
            a++;
        } else {
            a--;
        }
    }

    System.out.println(a);
    System.exit(0);
}
```

Tips: Husk at du må angi datatype når du angir verdi til en variabel direkte. For eksempel, hvis du har deklarert `a` i bss (f.eks. ved: `a resb 1`), og ønsker å initiere denne variabelen til `0`, kan du oppnå dette ved: `mov [a], byte 0`