UiA

Group number and student names:

gruppe 22

Aleksander Kolsrud

Emmanuel Medard

Erik Aasen Eskedal

Kevinas Maksevicus

Kristian Skibrek

Tobias Vetrhus

Course Code:	***IS-105***
Course Name:	***Datakommunikasjon og operativsystem***
Responsible teacher/professor:	***Janis Gailis***
Deadline / Due date:	
Number for pages included in the deliverable:	*10*
Title:	*modul 1*

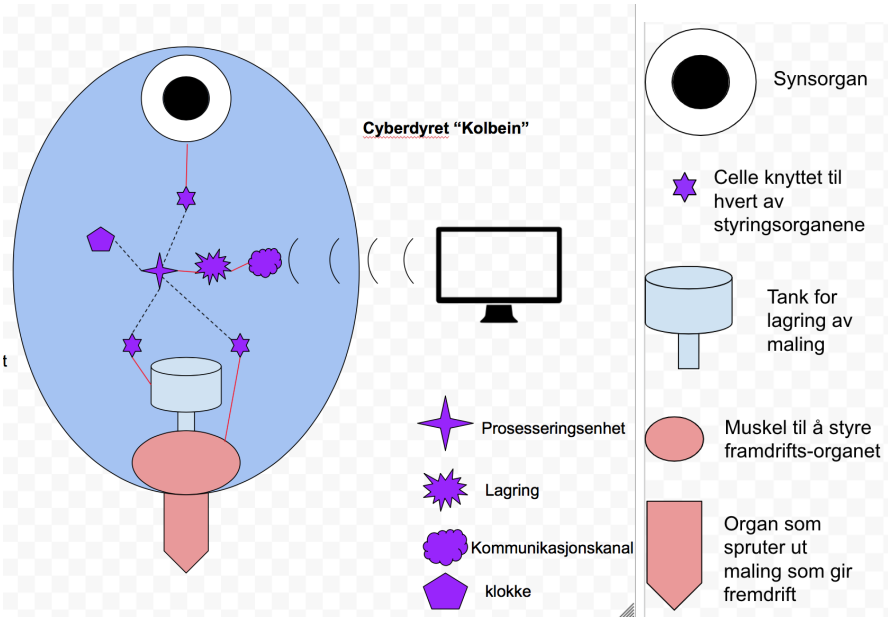
We confirm hereby that we are not citing or using in other ways other's work without stating that clearly and that all of the sources that we have used are listed in the references	Yes	No
We allow the use of this work for educational purposes	Yes	No
We hereby confirm that every member of the group named on this page has contributed to this deliverable/product	Yes	No

Innhold:

Modul 1 IS-105

Introduksjon

Det først vi måtte gjøre var å sette oss ned og tenke igjennom “hva er et cyberdyr?” og “hva trenger et cyberdyr”. Vi slo fast at det er en slags metafor for en datamaskin, og at derfor vil de grunnleggende komponentene i en datamaskin også være de grunnleggende komponentene i et cyberdyr. Akkurat som datamaskiner ville det finnes mange forskjellige typer, og akkurat hvilke komponenter den består av vil være avhengig av hva den skal brukes til.
Så hva skal cyberdyret brukes til? Oppgaven spesifiserte at cyberdyret må kunne lagre kommandoer og et system for å kunne skille mellom forskjellige kommandoer. Ved å bruke en demokratisk prosess skapte gruppa vår cyberdyret Kollbein.



Cyberdyret Kollbein.

Lagring

Det første vi fikk vite at Kollbein måtte kunne gjøre var å lagre ting. Dette løste vi ved å gi Kollbein et dedikert celle til som kan lagre kommandoer. Denne cellen består av 256 mindre deler, som enten kan være aktivert eller avslått. I illustrasjonen er denne cellen representert av en lilla figur markert "lagring". Vi tenkte dette var en god løsning siden det ligner på hvordan datamaskiner lagrer informasjon i virkeligheten. I datamaskiner er informasjon lagret som med de binære verdiene 0 og 1, akkurat som Kollbein har av og på. De binære verdiene kalles bits og åtte bits er én byte. Hvis Kollbein hadde vært en datamaskin, ville vi sagt at han har 32 byte med lagringsplass. Hvorfor vi valgte akkurat denne mengden lagring er egentlig helt vilkårlig. Vi tenkte 32 byte ville være nok av plass til kommandoene vi vil at Kollbein skal utføre.

Kommandoer

Det er ikke nok å bare ha en celle hvor informasjon kan ligge lagret. For at denne lagringscellen skal være til noe nytte må informasjonen kunne brukes til noe. For å få dette til må man ha en måte å putte nye data inn i lagringscellen, noe som kan tolke disse dataene og gjøre dem til signaler til musklene, som fører til at cyberdyret gjør noe. Vi delte dette inn i to deler. Det første kalte vi for "prosesseringsenhet". Vi tok inspirasjon fra prosessorer i

datamaskiner. I en datamaskin er en prosessor en enhet som tolker et programs instruksjoner og genererer output til datamaskinens interface (Sioris, 2018). Dette er akkurat det vi vil at prosessorenheten til Kollbein skal gjøre. Vi vil at den skal ta instruksjonene i lagringsenheten og gjøre det om til output til musklene dens. Det andre vi trengte var en måte å putte kommandoer inn i lagringsenheten. Vi kalte denne delen for "kommunikasjonskanalen", og er der cyberdyret Kollbein mottar signaler/input fra oss, mennesker. Vi har lagt det opp til at kommunikasjonskanalen kan motta trådløse signaler fra for eksempel en datamaskin. Kommunikasjonskanalen kan ta imot disse signalene og sende dem rett inn lagringen. Selv om det er tatt litt inspirasjon fra virkelighetens datamaskiner, så er det ikke helt sånn det fungerer. I virkeligheten er input koblet til prosessoren, som er ansvarlig for å putte informasjonen fra input inn i minnet. For enkelhetens skyld antar vi at i Kollbein, så er kommunikasjonskanalen er koblet direkte til lagringsenheten.

Nå som hardware er på plass, kan vi begynne å tenke på instruksjonene til Kollbein. Målet var at vi skulle kunne be Kollbein om å gå i en firkant med en vilkårlig størrelse på overflaten. Vi tenkte at hvis vi skulle kunne be Kollbein om å gjøre dette så måtte vi ha minst to kommandoer. Vi må kunne be Kollbein om å gå fremover og om å rotere. For å bevege Kollbein fremover tenkte vi at han har muskel som er i form av en tank med malin. Når muskelen aktiveres klemmer den ut maling, som gir Kollbein fremdrift og legger igjen et spor der det har beveget seg. For at Kollbein skal kunne snu seg har han en muskel som kan styre hvor malingen sprutes ut. Derfor konkluderte vi med at Kollbein trenger å forstå to kommandoer for å gå i en firkant.

Kommando	**Kode**
F(X)	**01 XXXXXX**
R(x grader)	**10 XXXXXX**

Den første kommandoen F(X) er kommandoen for å gå fremover. Vi tenkte at tanken kunne ha to verdier, på og av. Når Kollbein leser kommandoen ser han først på starten at kommandoen starter med 01, som er unikt for fremover- kommandoen. Det er etterfulgt av 6 nye bits som er hvor mye han skal gå framover. Vi tenkte oss to mulige mål på framdrift. Det

første var hvor lenge han gir framdrift, og det andre er hvor mye maling kollbein spruter ut. Vi valgte å gå for at Kollbein måler fremdrift i tid.

For at dette skal fungere må Kollbein forstå hva tid er. Det løste vi ved å gi han en klokke som er koblet til prosseseringsenheten. Dette er igjen en parallell til Kollbeins fjerne slektning, datamaskinen, som har en prosessor som er koblet til en klokke. R(X) er kommandoen for å få Kollbein til å snu seg. Vi tenkte at det er lurt at Kollbein skal kunne bevege i mange forskjellige grader, ikke bare rette vinkler, sånn at det kan lage andre former enn firkanter om vi vil det. Vi tenkte derfor at rotasjonsmuskelen kan vris fritt 90 grader til høyre og venstre. I kommandoen er det et 6 bits binært tall som representerer hvor rotasjonsmuskelen skal roteres til. Det er 64 forskjellige kombinasjoner man kan ha med 6 bits. Det høyeste tallet man kan lage er 63. For å gjøre det lettere å regne tenkte vi at den maksimale verdien Kollbein skal forstå er 62. Om Kollbein mottar en kommando om å rotere maks, 10 111110 eller R(62), så vil det vri rotasjonsmuskelen helt mot høyre. Om det mottar en kommando om å rotere minst, 10 000000 eller R(0), vil rotasjonsmuskelen vris helt til venstre. Rotere 31, 10 100001, vil være at rotasjonsmuskelen vris så den står rett bakover. Dette er grunnen til at vi valgte 62 som maks. Om vi lot 63

Om vi tenker at rett bakover er 0 grader, helt til høyre er 90 grader og helt til venstre er -90 grader så vil formelen for å konvertere grader til binære verdier som kollbein forstår 62 * (x + 90) / 180. Om vi for eksempel vil at Kollbein skal vri seg 90 grader til høyre kan vi bruke formelen for å finne ut hva vi skal be Kollbein om å gjøre. 62 * (90 + 90) / 180 = 62. Vi må be kollbein om R(62). Bakdelen ved dette systemet er at man kan ikke putte hvilket som helst tall inn i formelen og få en gyldig verdi, fordi de fleste tall vil produsere desimaltall, og Kollbein forstår kun heltall. En mulig løsning til dette er å utvide kommandoene til Kollbein til å forstå tall fra 0 til 180, eller så han forstår desimaltall. Likevel fungerer systemet fint til det vi vil at Kollbein skal kunne gjøre akkurat nå.

Tegne en firkant

Program

For at Kollbein skal bevege seg i en firkant kan vi gi det disse kommandoene.

Start

01 001010 (gå ti fram)

10 111110 (vri 62 (90 grader til høyre))

01 001010

10 111110

01 001010

10 111110

01 001010

10 111110

Slutt

Vi tenker at Kollbein leser kommandoene en og en fra start til slutt og at de forsvinner fra minnet når han har lest dem. Vi tenker at Kollbein hele tiden leser av hva som er i minne og at han alltid har et "mål" om at minnet skal være tomt. På denne måten har han ikke noe behov for en "start" og "stopp" kommando. Ettersom Kollbein har 32 byte med lagringsplass, har han mer enn nok plass til dette programmet som tar 8 byte med plass.

Hvordan vet Kollbein hva kommandoene betyr?

En ting som bør nevnes er at vi ikke har forklart hvordan Kollbein vet at kommandoene betyr det de gjør. Vi har bare sagt at Kollbein vet hva disse kommandoene betyr. Sånn vi ser det er det to mulige løsninger. Den første er at det ligger lagret i minnet til Kollbein hva disse kommandoene betyr, og hvordan Kollbein skal prosessere dem. Disse kommandoene ligger alltid i minnet, sånn at Kollbein han hente dem fram når de trengs. Problemet med denne løsningen er at vi har allerede sagt at Kollbein har et mål om at minnet skal være tomt, og ville derfor også tømt ut kommando-ßforklaringene. Den andre løsningen er at kommandoene er hardwired inn i prosessorenheten. Prosessorenheten vil da bestå av kombinatorisk logikk som behander kommandoene i minnet (https://www.geeksforgeeks.org/computer-organization-hardwired-vs-micro-programmed-control-unit/). Bakdelen ved denne løsningen er at det er veldig tungvint å legge til nye kommandoer. Om det er behov for mer komplekse handlinger i fremtiden bør vi bytte vekk den hardwirede prosessorenheten.

Rekursiv algoritme

En rekursiv algoritme er en algoritme hvor en funksjon kaller på seg selv med større eller mindre inputverdier (https://www.cs.odu.edu/~toida/nerzic/content/recursive_alg/rec_alg.html) . Vi vil at vår rekursive algoritme skal gjøre at Kollbein produserer 4 kvadrater inni hverandre. Vi tenker at han starter med det innerste kvadratet og jobber seg utover.

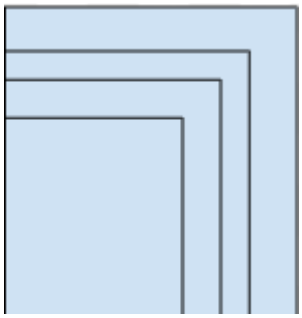
Den rekursive algoritmen vil se slik ut:

```
//variabler som brukes i algoritmen
antallKvadrat = 4;
inkrement = 1;
antallTegnet = 0;


//selve algoritmen
RekursivAlgoritme(sidelengde){
        //kommandoer for å lage firkanten med sidelengde lange sider
        F(sidelengde);
        R(62);
        F(sidelengde);
        R(62);
        F(sidelengde);
        R(62);
        F(sidelengde);
        R(sidelengde);
        //oppdaterer variablen
        antallTegnet += 1;


        //om antall kvadrater vi har tegnet er mindre enn antallet vi vil tegne, kaller funksjonen på seg selv, hvis ikke, så er den ferdig
        if(antallTegnet < antallKvadrater){
                RekursivAlgoritme(sidelengde+inkrement);
```

```
        }
}
```

```
RekursivAlgoritme(2);
```

Denne algoritmen er skrevet på et fiktivt programmeringsspråk for demonstrasjon. Det er basert på programmeringsspråk som Java og C. De som er kjent med disse språkene vil forstå hvordan denne algoritmen fungerer. Linjene som starter med "//" er kommentarer, og er ikke en del av selve programmet. De forklarer hvordan algoritmen fungerer. Kort oppsummert fungerer et ved at vi har en funksjon, som lager en firkant, med input sidelengde lange sider. Når det er gjort kaller funksjonen på seg selv igjen, men med input som er inkrement større. For at en rekursiv algoritme ikke skal fortsette for alltid må den ha en slags exit status. I dette tilfellet er at hvis antall kvadrater Kollbein overstiger antall kvadrater vi vil at Kollbein skal tegne, så kaller ikke funksjonen på seg selv igjen. Til slutt vil vi stå igjen med et resultat som ligner på figuren vist under



Kilder

Sioris S (2018) *What is processor speed, and why does it matter?* https://www.hp.com/us-en/shop/tech-takes/what-is-processor-speed