Group number and student names:

gruppe 22

Aleksander Kolsrud

Emmanuel Medard

Erik Aasen Eskedal

Kevinas Maksevicus

Kristian Skibrek

Tobias Vetrhus

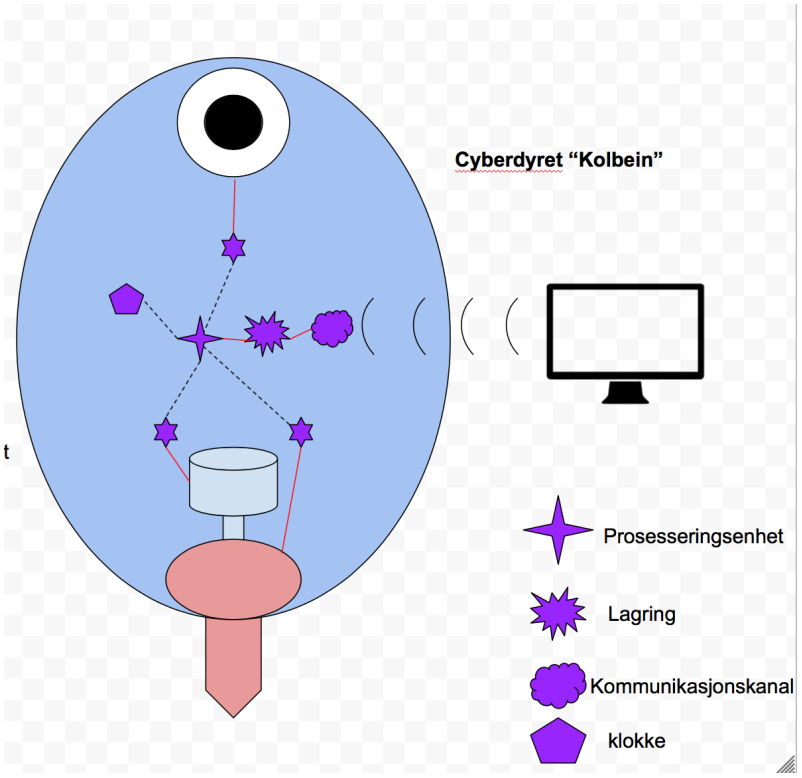
Course Code:	***IS-105***
Course Name:	***Datakommunikasjon og operativsystem***
Responsible teacher/professor:	***Janis Gailis***
Deadline / Due date:	
Number for pages included in the deliverable:	*10*
Title:	*modul 2*

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

Modul 2 IS-105

introduksjon

Denne oppgaven bygger videre på oppgaven vår “modul 1”. Vårt cyberdyr heter Kolbein, og fra modul 1 ser det slik ut:



For å få Kolbein til å gjøre noe, så starter med å sende inn kommandoer i kommunikasjonskanal hvor de blir kompilert til binære data, så sendes de til lagring (RAM), Prosessering (CPU) henter fra lagringen, og ser hvilken output instruksen skal sendes til, eventuelt hvilken input den skal vente på data fra. Prosessoren er koblet til to kontrollere for outputs (muskel for rotasjon og tank for maling) og en sensor (øyet). Outputet som er koblet til kontrollere, som kan sende interrupt signaler tilbake til CPUen.

I forrige modul sa vi at lagringsenheten i Cyberdyret har plass til 32 byte i lagringsenheten sin. Hvis man trenger plass kan ha den flere lagringsenheter hvor hvis den ene blir full, sendes nye kommandoer til ny enhet, eller vi kan utvide den vi har til å romme flere byte med data. Ettersom cyberdyret vårt kun har en RAM minneenhet og ikke noen slags harddisk eller SSD, vil dataene bli borte om cyberdyret ikke er slått på.

Instruksjonene vi introduserte i forrige modul var fram, snu og syn. Vi sa at instruksjonene besto av en byte hvor de to første bitsene var ID-en til instruksen og de seks følgende bitsene var til data om hvor langt de den skulle gå, eller snu seg. 01 var fram 10 var snu, 00 var syn. Med to bits har vi kun mulighet til fire instrukser, Dette var greit helt i starten, men senere vil vi se at cyberdyret har behov for å kunne utføre logiske operasjoner, hoppe til andre adresser og lignende. Det vil det være behov for flere instruksjoner. Derfor utvider vi instruksjons-IDen til fire bit. Dette gir oss mulighet for seksten ulike instruksjoner. Ettersom instruksjonene består av en byte gjør utvidelsen av ID-en at det er mindre plass til data etterpå. Derfor må instruksjoner om at den skal bevege seg lenger, bli brutt ned i flere mindre operasjoner.

I/O enhets-relaterte instruksjoner:
fram: 1000XXXX (får cyberdyret til å gå X skritt fram)
snu: 0100XXXX (får cyberdyret til å snu seg X grader)
mål posisjon: 00001XXX (måler posisjonen og lagrer X adresse)

prosesserings-relaterte instruksjoner:
vent: 0001XXXX (får cyberyret til å ikke gjøre annet enn å vente i x sekunder)
jump: 0010XXXX (får programmet til å hoppe til en annen adressee)
start: 11110000 (starter programmet)
slutt: 00000000 (avslutter programmet)

En annen mulig løsning hadde vært å laste inn instruksjoner og tall separat. Om vi for eksempel vil at cybedyret skal gå framover 10 skritt, så ville vi først hatt en LOAD instruks, som sa at data på en viss adresse skal lagres i et register, så ville neste instruks vært å gå fram så mange skritt som er lagret i registeret. Denne måten ville vært tregere på noen områder, men raskere på andre. Om cyberdyret kun skal gå en kort distanse ville det vært tregere å måtte bruke flere instrukser for å gå framover. Om vi vil at cyberdyret skal gå en veldig lang distanse derimot er det raskere å bare be den gå framover en gang, enn å måtte instruere den om det flere ganger. Siden verdenen til cyberdyret er så liten, har vi bestemt oss for å heller måtte be den om å gå fremover flere ganger.

Hvordan vil så programmet for at cyberdyret lager en firkant se ut.

Adresse	data
01	start
02	gå fram 10
03	snu 90 grader
04	gå fram 10
05	snu 90 grader
06	gå fram 10
07	snu 90 grader
08	gå fram 10
09	slutt

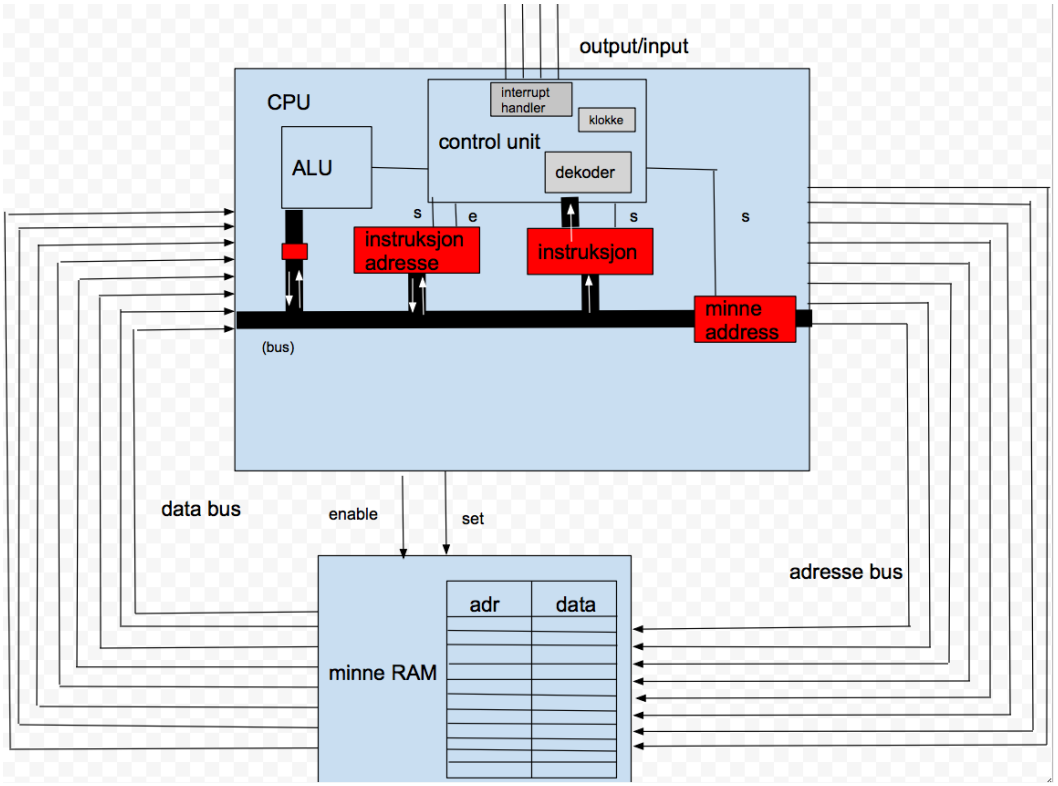
Dette er programmet lagt inn som kompilert kode i RAM, men skrevet på språk som er forståelig for oss. Cyberdyret forstår kun binære tall, men det blir litt for tungt for oss å lese. Ved første øyekast kan dette programmet se helt greit ut det, men det har et fundamentalt problem. Vårt system er synkront med en fast klokkesyklus som kjøres i CPU en. Vi ser bort fra “fetch-decode-execute” prinsippet, som de fleste moderne CPUer opererer etter hvor første klokkesyklus henter ut en instruksjonen, neste syklus dekoder hva instruksjonen vil gjøre, og tredje så utføres instruksjonen, videre repeteres dette (Scott M). Vi forenkler det heller litt og sier at på hver klokkesyklus så hentes, dekodes og utføres en instruksjon fra RAM. Hvis vi da ser på programmet vårt så ser vi at rett etter vi har bedt cyberdyret om å gå fremover, så ber vi det om å snu. Problemet er at “gå fram” er en I/O relatert instruks, disse utføres ikke, i motsetning til prosessering-relaterte instrukser, på en klokkesyklus. En mulig løsning på problemet ville vært at man har en “vent instruks”. Vi sier da at vi vet at et skritt fremover tar for eksempel 3 sekunder, så etter en instruks om å gå ti skritt, vil cyberdyret bli instruert om å vente tretti sekunder. i datatebellen vår ville det sett ut som det her:

Adresse	data
01	start
02	gå fram 10
03	vent 30 sek
04	snu 90 grader
05	vent 30 sek
06	gå fram 10
07	vent 30 sek
08	snu 90 grader
09	vent 30 sek
10	gå fram 10
11	vent 30 sek
12	snu 90 grader
13	vent 30 sek
14	gå fram 10
15	vent 30 sek
16	slutt

Det er en bedre løsning enn den forrige, men det er fortsatt ikke optimalt. Et problem er at vi ikke med sikkerhet kan si at det vil ta 30 sekunder hver gang, noen ganger vil det kanskje ta lenger tid, andre ganger ta mindre tid. Om det tar lenger tid enn beregnet har vi samme problem som i forrige versjon av programmet: den begynner på neste instruks før forrige instruks er utført. Om det tar kortere tid enn beregnet har vi problemet at cyberdyret bare står stille og venter på neste instruks. Løsningen vår blir et *interrupt* signal. Et interrupt signal er et signal som kommer fra en enhet eller software som er koblet til Operativsystemet (computer science). Interrupt signalet sier til prosessoren at den skal stoppe prosessen som foregår, lagre hvor den var, og sette i gang en ny prosess. I vårt cyberdyr kan kontrollere sende et interrupt signal tilbake til prosessorene når den er ferdig med å gå framover.

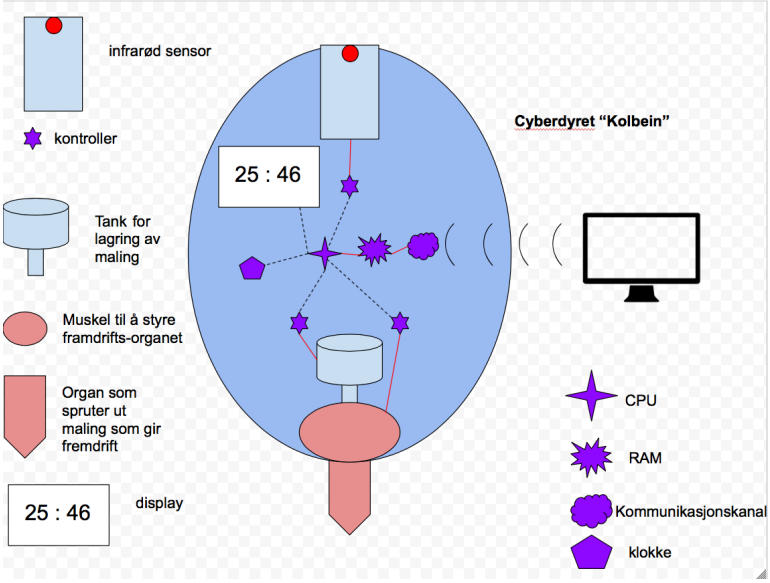
Med interrupt signaler kan vi begynne med *multiprogrammering.*
Multiprogrammering er når man har flere programmer som kjører på et system med kun en prosessor (tech target contributor 2015). En prosessor kan kun utføre en oppgave, eller tråd om gangen. Derfor må prosessoren bytte på å utføre oppgaver fra hvert program. For at dette skal fungere må man ha et *administrativt program.* Hvis cyberdyret vårt skal gjøre to ting samtidig, for eksempel å monitorere malingsnivå og tegne en firkant samtidig vil det være nødvendig med et tredje program (administrativt program), som administrerer at de to andre programmene ikke påvirker hverandre. Administrativt program gjør også at det går an å synkronisere med periferie enheter, for eksempel så kan vi få cyberdyret til å måle hvor den befinner seg mens den går framover.

Når vi kommer inn på multiprogrammering, så er det på tide å introdusere vårt CPU design.

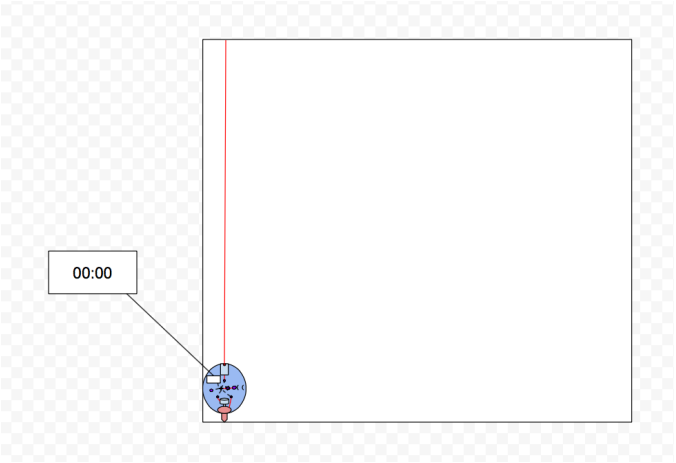


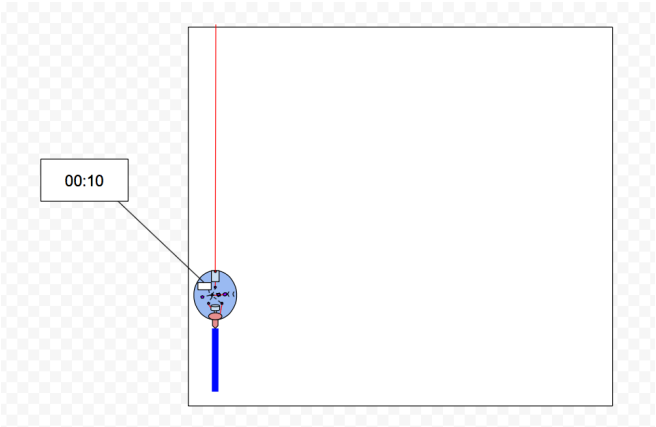
Vår CPU er basert på “The Scott CPU”, en fiktiv og enkel CPU beskrevet i J. Clark Scotts bok *But How Do It Know* (igjen beskrevet i youtube videoen https://www.youtube.com/watch?v=cNN_tTXABUA). På bunnen har vi minnet (RAM). På toppen har vi CPUen. Den består av to hoveddeler kontrollenheten, og ALU (arithmic logic unit). Kontrollenheten står for dekodingen og utførelsen av instruksjoner. Instruksjonene våre er bygget inn i den ved hjelp av grupper av transistorer som danner *gates*. Inne i kontrollenheten er en dekoder enhet, som er satt sammen av kretser bygget opp av gates som gjøre at visse inputs fører til visse outputs. ALU står for all behandling av tall, for eksempel addering og sammenligning av tall. Denne er nødvendig for at prosessoren skal kunne utføre logiske operasjoner som å vente. ALU består også av kretser laget av gates (simple CPU). De røde firkantene representerer registrere, som er midlertidige lagringspunkter for åttebits binære tall. En CPU har gjerne mange registre, vi har valgt å ta med de vi mener er viktigst. Den første er register for lagring av instruksjonen CPUen skal utføre, dette registeret er kalt “instruksjon” på illustrasjonen. “Instruksjon adresse” er navnet på et annet register. Dette registeret står for å sende ut ut på bussen hva adressen til det neste som skal hentes fra RAM er. Dette sendes til registeret “minne address” som er det som sender signal videre med adressen til RAM, hvor det blir kalkulert hvilke data som hører til denne adressen. Noen av linjene som kommer ut fra kontroll-uniten er markert med en “s” eller en “e”. S står for set, når kontroll-uniten sender ut et set-signal til et register, så lagrer den det som er på bussen i det øyeblikket. E står for enable, når kontroll-uniten sender ut enable-signal til et register så sender dette registeret det den har lagret på bussen.

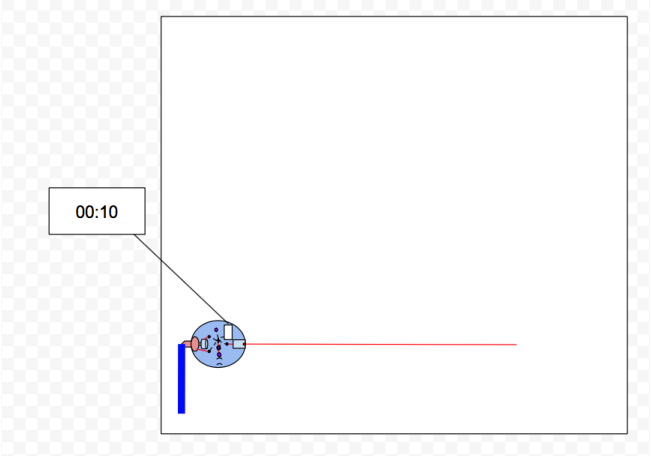
Da er vi tilbake til problemet vårt, hvordan skal vi få cyberdyret vårt til å gå fremover, samtidig som den skal si til oss hvor den befinner seg i verden. Før vi går videre inn på logikken av å programmere den til det, så må vi oppgradere cyberdyret vårt litt. For det første må cyberdyret vårt litt. Cyberdyret må ha en måte å kunne vite hvor den er i rommet. for øyeblikket har den et synsorgan i form av et øye. Cyberdyret vårt er litt for primitivt til å forstå inputet fra et øye. Det kreve ganske sofistikerte maskinlæringsalgoritmer for at det skal kunne forstå hvor den befinner seg utifra bilder som kommer fra øyet. derfor bytter vi ut øyet med en infrarød sensor, som kun måler hvor langt den er fra kanten av verdenen sin den veien den "ser". Vi må også gi cyberdyret vårt et slag display hvor den kan presentere dataene på en måte som gir mening for oss (homo sapiens).

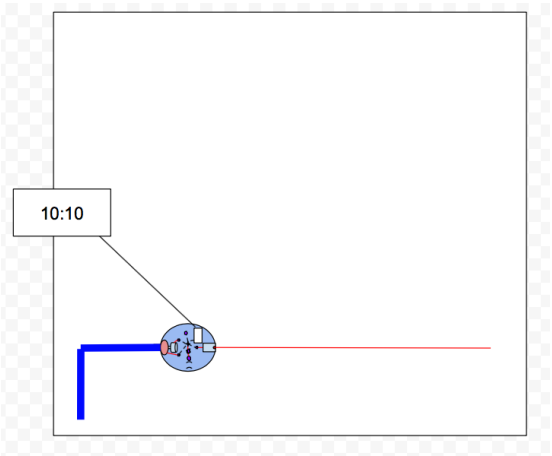


Vi må også sette noen regler for verdenen til Cyberdyret. Vi sier den er 100x100 skritt stor. cyberdyret starter nederst i venstre hjørnet, som den oppfatter som 00:00. Når den beveger seg et skritt fremover vil så merker den infrarøde sensoren at den er et skritt nærmere kanten av verdenen dens og displayet oppdaterer seg til å bli 00:01. Nedenfor illustreres hvordan cyberdyret beveger seg gjennom verdenen sin som en sekvens av illustrasjoner.


00:00


00:10


00:10



Nå har vi grunnlaget som trengs for å skrive et program som får cyberdyret til å lage en firkant, samtidig som den måler og viser på displayet sitt hvor. Et forslag er slik som dette:

	Adresse	data
	01	start
interrupt	02	gå fram 10
	03	jump 08
interrupt	04	snu 90
	05	jump 08
	06	
	07	
	08	mål posisjon 11/12
	09	oppdater display
	10	jump 08
	11	00
	12	00
	13	
	14	
	15	
	16	

programmet består av to deler: den ene delen er delen hvor cyberdyret tegner et kvadrat, den andre er delen hvor den måler posisjonen sin og outputer det til displayet. Programmet starter i adresse 01, så ber den cyberdyret om å gå framover. Mens cyberdyret utfører denne kommandoen så hopper programmet til delen hvor den måler posisjonen. Dette programmet får cyberdyret til å måle hvor langt den er unna kanten av verdenen sin og lagrer det enten i datafelt 11 eller 12 (avhengig av hvilken vei den peker). Denne delen av programmet looper (hoppper tilbake til starten) for alltid ved mindre det kommer et interrupt signal. Det kan komme to typer interrupt signal. Det første fra tanken for maling, som gir cyberdyret fremdrift, det andre er fra muskelen for rotasjon. Et interrupt signal fra muskelen for rotasjon vil få programmet til hoppe til den delen med instrukser for fremgang, og motsatt. Dette er et eksempel på en evig loop. Programmet har ingen måte det kan stoppe på. Cyberdyret vil fortsette å gå rundt i en firkant, helt til cyberdyret går tom for energi eller

maling. Hvis vi vil at programmet skal stoppe på et punkt så kan vi legge til en "slutt-condition" som, hvis den er oppfylt, får programmet til å avslutte. et forslag er for eksempel at vi setter av to datafelt. I det ene feltet har vi det ønskede antallet ganger vi vil at programmet vårt skal kjøre, i det andre har vi et felt som starter på 0, og inkrementeres med 1 hver runde programmet går. I vårt tilfelle blir det ønskede antallet ganger 4 og vi kan legge inkrementeringen rett etter cyberdyret har gått forover. Vi må også ha en instruksjon hvor de to datafeltene blir sammenlignet. Hvis de er like hverandre, så vil programmet avslutte.

kilder

computer science (n.d), *interrupts*
https://www.computerscience.gcse.guru/theory/interrupts

Scott M (n.d), *fetch, decode, evaluate*
https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49284

K-state, (2009) *basics of how operating systems work.*
http://faculty.salina.k-state.edu/tim/ossg/Introduction/OSworking.html

tech target contributor (september 2015) *multiprogramming*
https://whatis.techtarget.com/definition/multiprogramming

Mike (30.10.2015) *simple CPU design*
http://www.simplecpudesign.com/simple_cpu_v1/