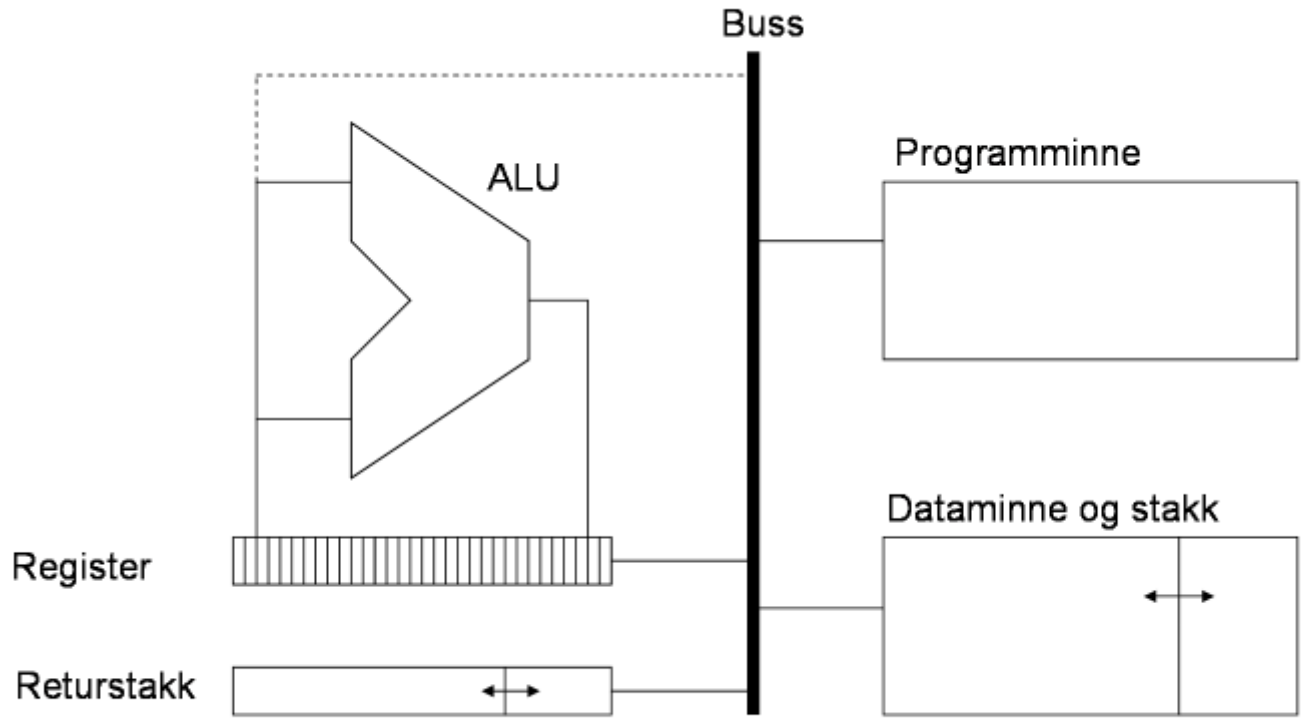
Dark load-store-maskin



Figur 1: Load-store arkitektur i Dark

Dette dokumentet beskriver arkitekturen til load-store-maskina som benyttes i Dark. Figur 1 viser hvordan den ser ut. Det finnes 32 registre i maskinen. I alle instruksjoner som jobber med data må minst en av operandene være et register og resultatet blir satt inn i et av registrene. Tre av registrene har spesielle anvendelsesområder og bør anvendes forsiktig:

- Register 0 (null) har alltid verdien null og kan ikke endres
- Register 1 (en, «zero») er reservert for assembleren
- Register 31 (brukes til «sp») brukes til stakkpekeren og peker alltid på nåværende stakktopp i nåværende stakk.

Syntaksdefinisjon

- Noe som er skrevet i `courier` er et nøkkelord som må være med.
- Noe som er skrevet i SMALL CAPS er et ord eller uttrykk som må settes, men programmereren kan selv bestemme den rette verdien f.eks. variabelnavn. Variabelnavnet kan ikke være det samme som et av nøkkelordene.
- Noe som er skrevet i *kursiv* er noe som eventuelt kan forekomme
- Noe som er skrevet i normal stil i semantiskbeskrivelse er en oppdatering av et internt (usynlig) register.
- {a|b} betyr enten a eller b.
- NUMMER er hvilket som helst nummer.
- VARIABEL er hvilken som helst variabel.

- Om en stakk(f.eks. returstakk) står på venster side av en ← betyr det at verdien legges på stakken. Om den står til høyre for pilen betyr det at verdien hentes fra toppen av stakken og tas bort.

Symbolske adresser

Syntaks:
ETIKETT
Semantikk:
ETIKETT ← minneposisjon
Lag en LABEL som viser hvor i minne et gitt kodestykke begynner. Disse LABEL-ene kan man skrive hvor som helst foruten inne i instruksjonene, f.eks. så går det fint å skrive snurra ret.

Register

$

Syntaks:
$ NUMMER
Sifferne angir hvilket register man bruker.

sp

Syntaks
`sp`
Angir at man vil bruke register 31.

zero

Syntaks
`zero`
Angir at man vil bruke register 0.

Register- og minneinstruksjoner

load

Syntaks:
`load` REGISTER, { REGISTER, NUMMER| NUMMER| VARIABEL}
Semantikk:
register ← {mem[register+ NUM]| NUM| VAR}
Målregisteret får enten verdien som er angitt som et nummer, innholdet i variabelen eller innholdet i minneposisjonen som man finner ut fra registeret + nummeret.

mov

Se **mv**

mv

Syntaks:
`mov|mv` REGISTER, REGISTER
Semantikk:
register ← register
Verdien av det andre registeret kopieres inn i det første.

store

Syntaks:
`store` REGISTER,{ REGISTER| NUMMER| VARIABEL}
Semantikk:
{mem[register+ NUMMER]| VARIABEL} ← register
Registeret lagres i variabelen eller i minneposisjonen som pekes ut av registeret + nummeret.

Variabeldefinisjoner

data

Syntaks:
`data` *nummer* NAVN
Om et nummer er gitt kommer varibelen ha denne verdien i begynnelsen av eksekveringen av programmet, ellers så er ikke verdien definert.

Kommentarer

Syntaks
; *tekst*
All tekst som står etter et semikolon(;) blir tolket som kommentarer

Aritmetiske instruksjoner

add

Syntaks
`add` REGISTER, REGISTER,{ REGISTER| NUMMER| VARIABEL}
Semantikk
register ← register + {register| NUMMER| VARIABEL}
Målregisteret får summen av andre og siste verdi.

and

Syntaks
`and` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk
register ← register $\wedge$ {register| NUMMER| VARIABEL}
Målregisteret får verdien til den logiske AND-en mellom andre og siste verdi

band

Syntaks
`band` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk
register ← register .$\wedge$ {register| NUMMER| VARIABEL}
Målregisteret får verdien til den bitvise logiske AND-en mellom andre og siste verdi.

bnot

Syntaks
`bnot` REGISTER, REGISTER
Semantikk register ← .$\neg$ register
Den bitvise logiske NOT av register to legges inn i målregisteret.

bor

Syntaks
`bor` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk
register ← register .$\vee$ {register| NUMMER| VARIABEL}
Målregisteret får verdien til den bitvise logiske OR-en mellom andre og siste verdi.

bxor

Syntaks
`bxor` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk
register ← register .$\oplus$ {register| NUMMER| VARIABEL}
Målregisteret får verdien til den bitvise logiske XOR-en mellom andre og siste verdi.

dec

Syntaks
`dec` REGISTER
Semantikk
register ← register − 1
Målregisteret blir dekrementert(minsket) med 1.

div

Syntaks:
`div` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk:
register ← register / {register| NUMMER| VARIABEL}
Målregisteret får svaret av divideringen mellom andre og siste verdi.

inc

Syntaks
`inc` REGISTER
Semantikk
register ← register + 1
Målregisteret blir inkrementert(øker) med 1.

mod

Syntaks:
`mod` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk:
register ← register {register| NUMMER| VARIABEL}
Målregisteret får resten etter divisjon mellom den andre og siste verdien.

mul

Syntaks:
`mul` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk:
register ← register ∗ {register| NUMMER| VARIABEL}
Målregisteret får produketet av den andre verdien og den siste verdien.

not

Syntaks:
`not` REGISTER, REGISTER
Semantikk:
register ← ¬ register
Logisk NOT av det andre registeret legges i det første.

or

Syntaks:
`or` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}
Semantikk:
register ← register ∨ {register| NUMMER| VARIABEL}
Målregisteret får verdien til den logiske OR mellom det andre og siste verdiene.

sub

Syntaks:
`sub` REGISTER, REGISTER, { REGISTER| NUMMER| VARIABEL}

Semantikk:
register ← register − {register| NUMMER| VARIABEL}
Målregisteret får differansen av det andre registeret og den siste verdien.

xor

Syntaks:
`xor` REGISTER, REGISTER,{ REGISTER| NUMMER| VARIABEL}
Semantikk:
register ← register ⊕ {register| NUMMER| VARIABEL}
Målregisteret får verdien til XOR-en mellom det andre registeret og den siste verdien.

Instruksjoner som kontrollerer programflyt

call

Syntaks
`call` ETIKETT
Semantikk
returstakk ← pc
pc ← ETIKETT
Legger adressen til nåværende instruksjon på retrustakken og hopper til neste instruksjon som ETIKETT peker på.

end

Syntaks
`end` |ETIKETT
Semantikk
pc ← ETIKETT
Slutt på kildekodefilen. Om ETIKETT er gitt skal prosessoren begynne å eksekvere på denne instruksjonen, ellers starter prosessoren med den absolutt første instruksjonen.

jeq

Syntaks:
`jeq` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}
Om verdiene i registrene er like hopper programmet til den symbolske adresse ETIKETT, eller skjer det ingenting.

jge

Syntaks:
`jge` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}

Om verdien i det første registeret er større eller lik verdien i det andre registeret så hopper programmet til ETIKETT, eller skjer det ingenting.

jgt

Syntaks:
`jgt` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}
Om verdien i det første registeret er større en verdien i det andre registeret hopper programmet til ETIKETT, eller skjer det ingenting.

jle

Syntaks:
`jle` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}
Om verdien i det første registeret er minder eller lik verdien i det andre registeret hopper programmet til ETIKETT, ellers skjer ingenting.

jlt

Syntaks:
`jlt` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}
Om verdien i det første registeret er minder en verdien i det andre registeret hopper programmet til ETIKETT, ellers skjer ingenting.

jmp

Syntaks:
`jmp` ETIKETT
Semantikk:
{|pc ← ETIKETT}
Eksekveringen fortsetter med instruksjonen via ETIKETT

jne

Syntaks:
`jne` REGISTER, REGISTER, ETIKETT
Semantikk:
{|pc ← ETIKETT}
Om verdiene i registrene er ulike hopper programmet til ETIKETT, ellers skjer ingenting.

ret

Syntaks:
`ret`

Semantikk:
pc ← returstakk
Ekekveringen hopper tilbake fra subrutine.

stop

Syntaks:
`stop`
Semantikk:
–
Eksekveringen avsluttes.