Løsningsforslag til øving 5 – TFE4105 Digitalteknikk og Datamaskiner Høsten 2012

Oppgave 1 – "Papirsimulering" av utførende enhet

Styreordsekvens	Registeroperasjon	
011 011 001 0 0010 0 1	R3 ← R3 + R1	; R3 = 01100111
100 100 001 0 1001 0 1	R4 ← R4 ∨ R1	; R4 = 01110100
101 101 001 0 1010 0 1	R5 ← R5 ⊕ R1	; R5 = 01101100
001 001 000 0 1011 0 1	$R1 \leftarrow \overline{R1}$	; R1 = 11011111
001 001 000 0 0001 0 1	R1← R1 + 1	; R1 = 11100000
110 110 001 0 0101 0 1	$R6 \leftarrow R6 + \overline{R1} + 1$	; R6 = 01100001
111 111 001 0 0101 0 1	$R7 \leftarrow R7 + \overline{R1} + 1$	; R7 = 01101001
000 111 000 0 0000 0 1	R0 ← R7	; R0 = 01101001

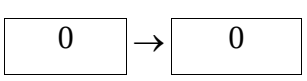
Oppgave 2 – Operand adressering

X=(A+B)*(D-B)/(C-B) og n-adressemaskin (m = antall minnereferanser per instruksjon):

n = 0		n = 1		n = 1½		n = 3	
Assembler	m	Assembler	m	Assembler	m	Assembler	m
PUSH A	2	LD A	2	MOVE R0,D	2	ADD T1,A,B	4
PUSH B	2	ADD B	2	SUB R0,B	2	SUB T2,C,B	4
ADD	1	ST T1	2	MOVE R1,C	2	SUB T3,D,B	4
PUSH B	2	LD C	2	SUB R1,B	2	DIV T4,T2,T1	4
PUSH C	2	SUB B	2	DIV R0,R1	1	MUL X,T3,T4	4
SUB	1	ST T2	2	MOVE R1,A	2		
PUSH B	2	LD D	2	ADD R1,B	2		
PUSH D	2	SUB B	2	MUL R1,R0	1		
SUB	1	DIV T2	2	MOVE X,R1	2		
DIV	1	MUL T1	2				
MUL	1	ST X	2				
POP X	2						
Tot m. ref	19		22		16		20

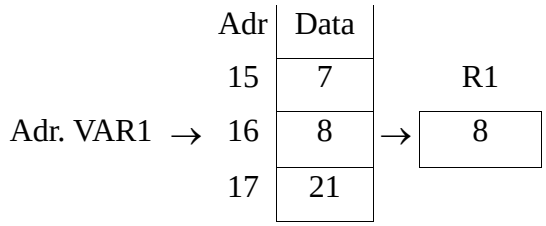
Oppgave 3 – Adresseringsmodi

1) MOV R1,R0



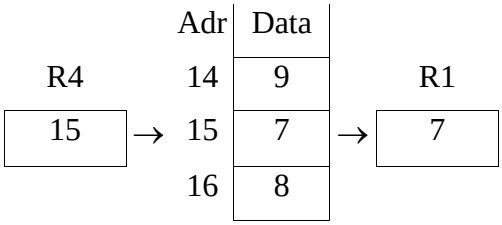
Operanddel er register R0 (kodes 000) og register R1 (kodes 001).

2) MOV R1,VAR1



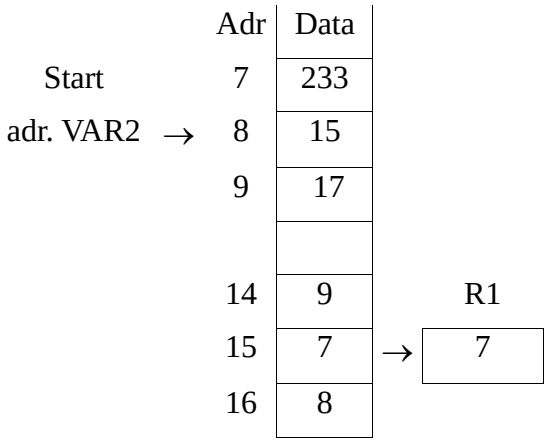
Operanddel er register R1 (kodes 001) og lageradresse VAR1 (16).

3) MOV R1,[R4]



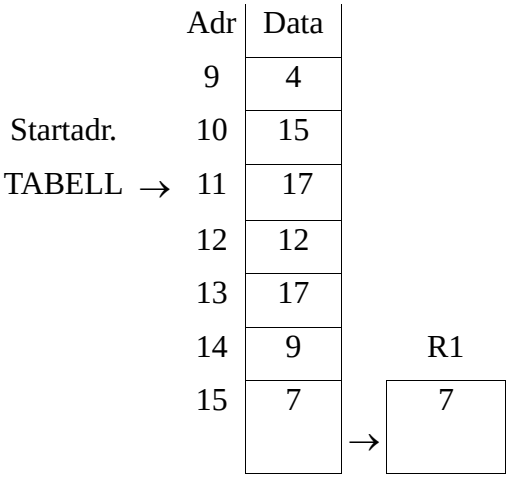
Operanddel er register R1 (001) og register R4 (100).

4) MOV R1,[VAR2]



Operanddel er register R1 (001) og lageradresse VAR2 (8).

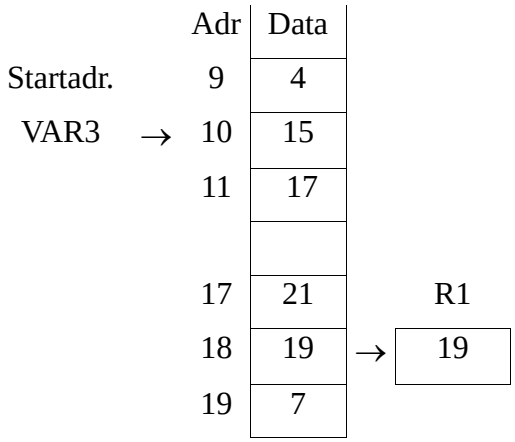
5) MOV R1,TABELL(R2)



Her finnes verdien ved å ta utgangspunkt i starten av tabellen (adr 11) og plusse på R2 (=4), som gir adresse 15. Innholdet i adresse 15 overføres til R1.

Operanddel er register R1 (001) og lageradresse TABELL (10). I tillegg må det oppgis hvilket indeksregister som brukes. Her er indeksregister R2 (010).

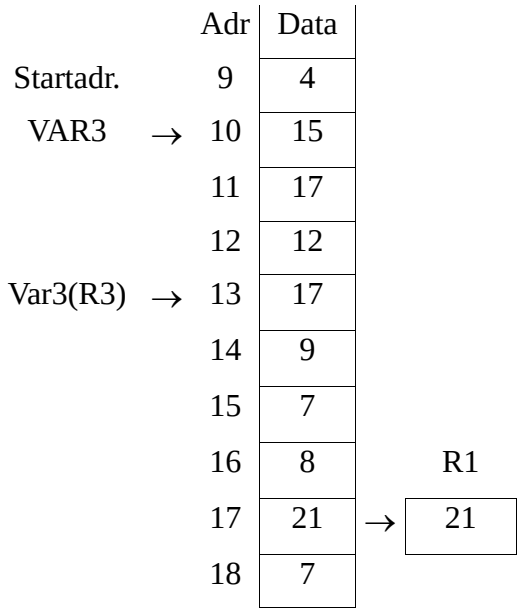
6) MOV R1,[VAR3](R3)



Her finner vi først adressen til var3 (adr 10). Til innholdet i adr 10, som er 15 adderes R3 (3). Dette gir adresse 18 og innholdet (19) overføres til R1.

Operanddel er register R1 (001) og lageradresse VAR3 (10). Indeksregister er R3 (011).

7) MOV R1,[VAR3(R3)]



Her finner vi først Var3(R3), dvs. adressen til Var3 + R3, som gir adr. 13. Innholdet i adr. 13 (17) gir adressen til verdien vi ønsker.

Operanddelen blir den samme som i forrige deloppgave. Det er bare opkoden som skiller denne fra foregående instruksjon.

Oppgave 4 – RISC og CISC

a) Karakteristiske trekk ved RISC:

- Få instruksjoner og adresseringsmodi
- Fast lengde på instruksjonene gir enkel instruksjonsdekoding
- Mange registre
- Gjør ikke operasjoner direkte på hovedlager men bruker registrene, load/store arkitektur
- Styreenheten er ikke-programmert (*"Hardwired control")*
- Instruksjoner utføres i løpet av en klokkesyklus

Karakteristiske trekk ved CISC

- Mange instruksjoner og adresseringsmodi, ofte også noen sære typer som få bruker
- Variabel instruksjonslengde gir komplisert instruksjonsdekoding
- Færre registre enn RISC
- Gjør operasjonene direkte på data i hovedlager
- Styreenheten er ofte mikroprogrammert
- Det kreves flere klokkesykler for å utføre en instruksjon

a) Maskinkode laget for en RISC har som oftest flere instruksjoner enn en tilsvarende CISC kode, men vil oftest utføres hurtigere hvis det er snakk om heltallsberegninger. Hvis vi har et høyt antall komplekse operasjoner i et program kan vi få en situasjon der en tradisjonell CISC har bedre ytelse. Et eksempel kan være programmer for matematiske/vitenskapelige beregninger som bruker mye flyttallsaritmetikk.

Oppgave 5 – Assemblerprogram for RISC

Merk at det er mange mulige løsninger her, vedlagte forslag utnytter mulighetene i branch- og compare-fasilitetene i ISAen fullt ut, samt at det utnytter løkketeller-registeret CTR. Det er mulig å programmere uten dette, men da blir det flere kodelinjer og det må brukes flere kladderegistre. Løsninger uten bruk av CTR godtas også.

Assemblerkoden er lagt til løsningsforslaget som eget vedlegg. Den er programmert ved hjelp av PPC-RISC reference, vedlagt øvingsteksten.

Løsningsforslag øving 5 oppgave 5

Oppgave 5a)

```
1  if:     cmpi 0,r5,1      # cr0 <- z-1
2          beq else         # hopp til else-blokk hvis z == 1
3
4          cmpi 0,r3,0      # cr0 <- x-0
5          bgt incr         # har z != 1, adder hvis x > 0
6          cmp 0,r4,r5      # cr0 <- y-z
7          bgt else         # hopp til else-blokk hvis y > z
8
9  incr:   addi r3,r5,1     # r3 = z + 1
10         b end            # hopp til avslutning (kan skrive blr)
11
12 else:   addi r3,r5,-1    # r3 = z - 1
13
14 end:    blr              # returner til adresse LR
```

Oppgave 5b)

Her benyttes r12 og r13 som kladderegistre, r13 for å lagre indeksen i og r12 for å lagre verdien av tegnet som leses/skrives fra B[]. Statusregisteret CR1 benyttes også for å lagre resultatet av sammenligningen i linje 14-15, slik at ikke resultatet fra sammenligningen i linje 6 overskrives.

```
1          addi r13,0,0     # nullstill indeks til streng B
2          mtctr r3         # kopier løkketeller til CTR
3
4  while:  lbzx r12,r5,r13  # last inn B[i] (i=r13) fra M[r5+r13]
5          addi r13,r13,1   # oppdater strengeindeks
6          cmp 0,r4,r12     # sammenlign A og B[i], lagre i CR0
7          bdnne while      # fortsett hvis CTR != 0 (n) og A-B[i] != 0 (ne)
8
9          # må sjekke hva som skjedde i løkken her:
10         #  - vet hvor langt vi kom fra antall sammenligninger i r13
11         #  - har status fra siste sammenligning A-B[i] i CR0
12         #  - kom vi til siste tegn er det *to* muligheter, avhengig
13         #    av om A=B[i] (fyll inn) eller A!=B[i] (avslutt)
14         cmpi 1,r13,1     # sjekk om første tegn var likt, lagre i CR1
15         beq 1,end        # avslutt hvis første tegn var likt
16         beq null         # hopp til nullfylling hvis siste tegn var likt (CR0)
17         cmp 0,r3,r13     # sammenlign N - strengeindeks
18         beq end          # avslutt hvis ingen like tegn A i B
19
20 null:   addi r13,r13,-1  # dekrementer til posisjon foran funnet tegn
21         mtctr r13        # kopier ny løkketeller til CTR
22         addi r13,0,0     # nullstill indeks til streng B
23         addi r12,0,0     # lag NULL-byte i et register
24
25 for:    stbx r12,r5,r13  # lagre NULL-byte i M[r5+r13]
26         addi r13,r13,1   # oppdater strengeindeks
27         bdnz for         # hopp hvis CTR ikke er telt ned
28
29 end:    blr              # returner til adresse LR
```