

Registrene er spesielle heltallsvariabler som er ekstra tett koblet til regneenheten. Vi skal bruke disse registrene:

%EAX %ECX %EDX %EBP %ESP

<u>func:</u>	<u>movl</u>	<u>\$0,%eax</u>	<u># Initier til 0.</u>
Navnelapp	Instruksjon	Parametre	Kommentar

.globl xxx	Navnet xxx skal være kjent utenfor filen
movl <v ₁ >, <v ₂ >	Flytt <v ₁ > til <v ₂ >.
cdq	Omform 32-bits %EAX til 64-bits %EDX:%EAX.
leal <v ₁ >, <v ₂ >	Flytt <v ₁ >s <i>adresse</i> til <v ₂ >.
pushl <v>	Legg <v> på stakken.
popl <v>	Fjern toppen av stakken og legg verdien i <v>.
negl <v>	Skift fortegn på <v>.
addl <v ₁ >, <v ₂ >	Adder <v ₁ > til <v ₂ >.
subl <v ₁ >, <v ₂ >	Subtraher <v ₁ > fra <v ₂ >.
imull <v ₁ >, <v ₂ >	Multipliser <v ₁ > med <v ₂ >.
idivl <v>	Del %EDX:%EAX med <v>; svar i %EAX; rest i %EDX.
andl <v ₁ >, <v ₂ >	Logisk AND.
orl <v ₁ >, <v ₂ >	Logisk OR.
xorl <v ₁ >, <v ₂ >	Logisk XOR.
call <lab>	Kall funksjon/prosedyre i <lab>.
enter \$<n ₁ >, \$<n ₂ >	Start funksjon/prosedyre på blokknivå <n ₂ > med <n ₁ > byte lokale variabler.
leave	Rydd opp når funksjonen/prosedyren er ferdig.
ret	Returner fra funksjon/prosedyre.
cmpl <v ₁ >, <v ₂ >	Sammenligning <v ₁ > og <v ₂ >.
jmp <lab>	Hopp til <lab>.
je <lab>	Hopp til <lab> hvis =.
sete <v>	Sett <v>=1 om =, ellers <v>=0.
setne <v>	Sett <v>=1 om ≠, ellers <v>=0.
setl <v>	Sett <v>=1 om <, ellers <v>=0.
setle <v>	Sett <v>=1 om ≤, ellers <v>=0.
setg <v>	Sett <v>=1 om >, ellers <v>=0.
setge <v>	Sett <v>=1 om ≥, ellers <v>=0.

4.5.2 Registre

Vi vil bruke disse registrene:

%EAX er det viktigste arbeidsregisteret. Alle uttrykk eller deluttrykk skal produsere et resultat i %EAX.

%ECX er et hjelperegister som brukes ved aritmetiske eller sammenligningsoperatorer eller til indeks ved oppslag i arrayer.

%EDX brukes til arrayadresser og som hjelperegister ved tilordning og divisjon.

%ESP peker på toppen av kjørestakken.

%EBP peker på den aktuelle funksjonens parametre og lokale variabler.

4.5.2.1 Navn

Hovedprogrammet, funksjoner og prosedyrer beholder sitt Pascal2016-navn men med en endelse så vi unngår dobbeltdeklarasjoner: `proc$name_nnn`.

Eksterne navn benyttes ved kall på biblioteksprosedyrer; navnet på startpunktet, dvs `main`, er også et eksternt navn. Slike navn skrives som de er i Linux, mens de trenger en understreking («`_`») foran navnet i Windows and Mac OS X.

Parametre trenger ikke navn i assemblerkoden siden de er gitt utfra posisjonen i parameterlisten: Første parameter har tillegg («offset») 8, andre parameter 12, tredje parameter 16 etc.

Variabler trenger heller ikke navn siden de også ligger på stakken. Nøyaktig hvor de ligger på stakken må kompilatoren vår regne seg frem til; dette avhenger av de andre lokale variablene i samme funksjon eller prosedyre.

Ekstra navn har vi behov for når assemblerkoden skal hoppe i løkker og annet. De får navn `.L0001`, `.L0002`, osv.

4.5.3 Oversettelse av uttrykk

Hovedregelen når vi skal lage kode for å beregne uttrykk, er at resultatet av alle uttrykk og deluttrykk skal ende opp i `%EAX`. Dette gjør kodegenereringen svært mye enklere, men vi får ikke alltid den mest optimale koden.

gcd.pas

```
1  program GCD;
2  /* A program to compute the {greatest common} of two numbers,
3     i.e., the biggest number by which the two original
4     numbers can be divided without a remainder. */
5
6  const v1 = 1071; v2 = 462;
7
8  var res: integer;
9
10 function GCD (m: integer; n: integer): integer;
11 begin
12     if n = 0 then
13         GCD := m
14     else
15         GCD := GCD(n, m mod n)
16 end; { GCD }
17
18 begin
19     res := GCD(v1,v2);
20     write('G', 'C', 'D', '(', v1, ', ', v2, ')', '=', res, eol);
21 end.
```

Figur 4.15: Et litt større Pascal2016-program `gcd.pas`

```
1  # Code file created by Pascal2016 compiler 2016-07-29 11:47:15
2  .globl main
3  main:
4      call    prog$gcd_1          # Start program
5      movl    $0,%eax             # Set status 0 and
6      ret                     # terminate the program
7  func$gcd_2:
8      enter   $32,$2              # Start of gcd
9                                   # Start if-statement
10     movl    -8(%ebp),%edx
11     movl    12(%edx),%eax        # n
12     pushl   %eax
13     movl    $0,%eax             # 0
14     popl    %ecx
15     cmpl    %eax,%ecx
16     movl    $0,%eax
17     sete    %al                 # Test =
18     cmpl    $0,%eax
19     je      .L0003
20     movl    -8(%ebp),%edx
21     movl    8(%edx),%eax         # m
22     movl    -8(%ebp),%edx
23     movl    %eax,-32(%edx)       # gcd :=
24     jmp     .L0004
```

```

25 .L0003:
26     movl    -8(%ebp),%edx
27     movl    8(%edx),%eax           # m
28     pushl   %eax
29     movl    -8(%ebp),%edx
30     movl    12(%edx),%eax         # n
31     movl    %eax,%ecx
32     popl    %eax
33     cdq
34     idivl   %ecx
35     movl    %edx,%eax             # mod
36     pushl   %eax                 # Push param #2
37     movl    -8(%ebp),%edx
38     movl    12(%edx),%eax         # n
39     pushl   %eax                 # Push param #1
40     call    func$gcd_2
41     addl    $8,%esp              # Pop parameters
42     movl    -8(%ebp),%edx
43     movl    %eax,-32(%edx)        # gcd :=
44 .L0004:
45     # End if-statement
46     movl    -32(%ebp),%eax        # Fetch return value
47     leave
48     ret
49 prog$gcd_1:
50     enter   $36,$1               # Start of gcd
51     movl    $462,%eax            # 462
52     pushl   %eax                 # Push param #2
53     movl    $1071,%eax           # 1071
54     pushl   %eax                 # Push param #1
55     call    func$gcd_2
56     addl    $8,%esp              # Pop parameters
57     movl    -4(%ebp),%edx
58     movl    %eax,-36(%edx)        # res :=
59     movl    $71,%eax             # 'G'
60     pushl   %eax                 # Push next param.
61     call    write_char
62     addl    $4,%esp              # Pop param.
63     movl    $67,%eax             # 'C'
64     pushl   %eax                 # Push next param.
65     call    write_char
66     addl    $4,%esp              # Pop param.
67     movl    $68,%eax             # 'D'
68     pushl   %eax                 # Push next param.
69     call    write_char
70     addl    $4,%esp              # Pop param.
71     movl    $40,%eax             # '('
72     pushl   %eax                 # Push next param.
73     call    write_char
74     addl    $4,%esp              # Pop param.
75     movl    $1071,%eax           # 1071
76     pushl   %eax                 # Push next param.
77     call    write_int
78     addl    $4,%esp              # Pop param.
79     movl    $44,%eax             # ','
80     pushl   %eax                 # Push next param.
81     call    write_char
82     addl    $4,%esp              # Pop param.
83     movl    $462,%eax            # 462
84     pushl   %eax                 # Push next param.
85     call    write_int
86     addl    $4,%esp              # Pop param.
87     movl    $41,%eax             # ')'
88     pushl   %eax                 # Push next param.
89     call    write_char
90     addl    $4,%esp              # Pop param.
91     movl    $61,%eax             # '='
92     pushl   %eax                 # Push next param.
93     call    write_char
94     addl    $4,%esp              # Pop param.
95     movl    -4(%ebp),%edx
96     movl    -36(%edx),%eax        # res
97     pushl   %eax                 # Push next param.
98     call    write_int
99     addl    $4,%esp              # Pop param.
100    movl    $10,%eax              # 10
101    pushl   %eax                 # Push next param.
102    call    write_char
103    addl    $4,%esp              # Pop param.
104    leave
105    ret

```

I tabell 4.5 er vist hvilken kode som må genereres for å hente en verdi $\langle n \rangle$, en enkel variabel $\langle v^{(b)o} \rangle$ (der b er blokknivået og o er variabelens «offset»), et arrayelement $\langle a^{(b)o} \rangle[\langle e \rangle]$ eller et uttrykk i parenteser $\langle e \rangle$ inn i register %EAX.

$\langle n \rangle$	\Rightarrow	<code>movl \$$\langle n \rangle$,%eax</code>
$\langle v^{(b)o} \rangle$	\Rightarrow	<code>movl $-4b(\%ebp)$,%edx movl $o(\%edx)$,%eax</code>
$\langle a^{(b)o} \rangle[\langle e \rangle]$	\Rightarrow	<code>\langleBeregn $\langle e \rangle$ med svar i %EAX subl \$low,%eax (Dropp om low=0) movl $-4b(\%ebp)$,%edx leal $o(\%edx)$,%edx movl $(\%edx,\%eax,4)$,%eax</code>
$\langle e \rangle$	\Rightarrow	<code>\langleBeregn $\langle e \rangle$ med svar i %EAX</code>

Tabell 4.5: Kode for å hente en verdi inn i %EAX

<code>+ $\langle e \rangle$</code>	\Rightarrow	<code>\langleBeregn $\langle e \rangle$ med svar i %EAX</code>
<code>- $\langle e \rangle$</code>	\Rightarrow	<code>\langleBeregn $\langle e \rangle$ med svar i %EAX negl %eax</code>
<code>not $\langle e \rangle$</code>	\Rightarrow	<code>\langleBeregn $\langle e \rangle$ med svar i %EAX xorl \$1,%eax</code>

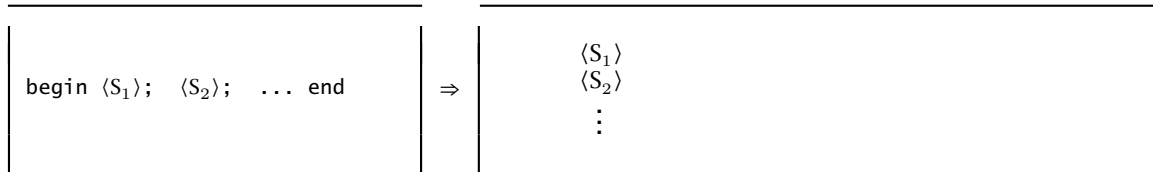
Tabell 4.6: Kode generert av unære operatører i uttrykk

$\langle e_1 \rangle + \langle e_2 \rangle$	\Rightarrow	<code>\langleBeregn $\langle e_1 \rangle$ med svar i %EAX pushl %eax \langleBeregn $\langle e_2 \rangle$ med svar i %EAX movl %eax,%ecx popl %eax addl %ecx,%eax</code>
$\langle e_1 \rangle \text{ div } \langle e_2 \rangle$	\Rightarrow	<code>\langleBeregn $\langle e_1 \rangle$ med svar i %EAX pushl %eax \langleBeregn $\langle e_2 \rangle$ med svar i %EAX movl %eax,%ecx popl %eax cdq idivl %ecx</code>
$\langle e_1 \rangle = \langle e_2 \rangle$	\Rightarrow	<code>\langleBeregn $\langle e_1 \rangle$ med svar i %EAX pushl %eax \langleBeregn $\langle e_2 \rangle$ med svar i %EAX popl %ecx cmpl %eax,%ecx movl \$0,%eax sete %al</code>

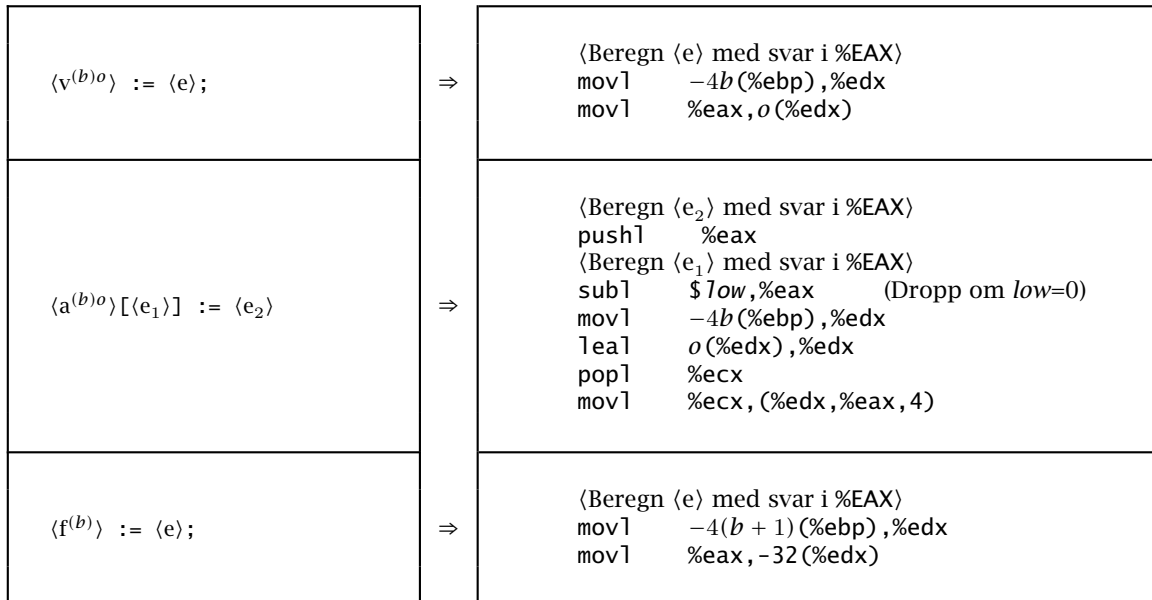
Tabell 4.7: Kode generert av binære operatører i uttrykk



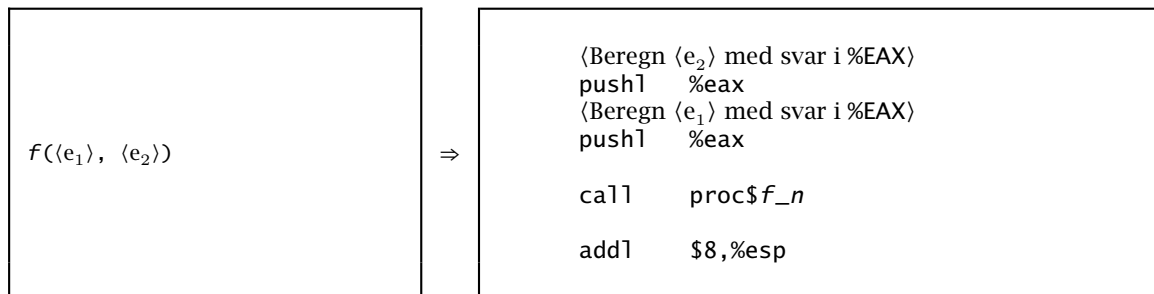
Tabell 4.8: Kode generert av tom setning



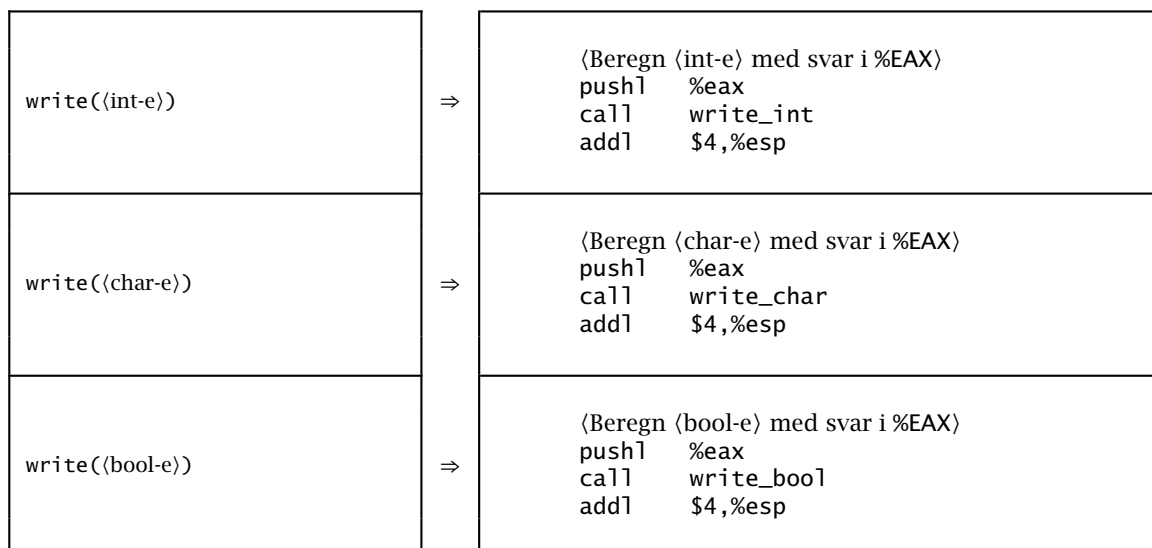
Tabell 4.9: Kode generert av sammensatt setning



Tabell 4.10: Kode generert av tilordning



Tabell 4.11: Kode generert av prosedyrekall



Tabell 4.12: Kode generert av kall på write

if <e> then <S>	⇒	<Beregn <e> med svar i %EAX> cmp1 \$0,%eax je <lab> <S> <lab>:
if <e> then <S ₁ > else <S ₂ >	⇒	<Beregn <e> med svar i %EAX> cmp1 \$0,%eax je <lab ₁ > <S ₁ > jmp <lab ₂ > <lab ₁ >: <S ₂ > <lab ₂ >:

Tabell 4.13: Kode generert av if-setning

while <e> do <S>	⇒	<lab ₁ >: <Beregn <e> med svar i %EAX> cmp1 \$0,%eax je <lab ₂ > <S> jmp <lab ₁ > <lab ₂ >:
------------------	---	--

Tabell 4.14: Kode generert av while-setning

function <f> (...): <type>; <D> begin <S> end	⇒	func\$<f>_n: enter \$(32+ant byte i <D>),\$<blokknivå> <S> movl -32(%ebp),%eax leave ret
procedure <p> (...); <D> begin <S> end	⇒	proc\$<p>_n: enter \$(32+ant byte i <D>),\$<blokknivå> <S> leave ret

Tabell 4.15: Kode generert av funksjons- og prosedyredeklarasjon

program xx; <D> begin <S> end.	⇒	.globl main main: call prog\$xx_n movl \$0,%eax ret prog\$xx_n: enter \$(32+ant byte i <D>),\$1 <S> leave ret
--	---	--

Tabell 4.16: Kode generert av hovedprogrammet