Load-store-maskin

Ola Ågren

5 september 1999

Abstrakt


Det finns 32 register i maskinen. I alla instruktioner som jobbar med data måste minst en av operanderna vara ett register och resultatet hamnar även det i ett register. Tre av registren har speciella användningsområden och bör användas med försiktighet;

1. Register 0 (noll) har alltid värden noll och kan inte ändras.
2. Register 1 (ett, "zero") är reserverat av assemblatorn.
3. Register 31 (även känt som "sp") används som stackpekare och pekar alltid på nuvarande stacktop i den nedåtväxande stacken.
Innehåll

1 Syntaxdefinition 3

2 Symboliska adresser 3

3 Register 3
  3.1 $ ................................. 3
  3.2 sp .............................. 3
  3.3 zero ........................... 3

4 Variabeldefinitioner 4
  4.1 data ............................ 4

5 Kommentarer 4

6 Instruktionsuppsättning 4
  6.1 add .............................. 4
  6.2 and ............................. 4
  6.3 band ............................ 4
  6.4 bnot ............................. 4
  6.5 bor .............................. 5
  6.6 bxor ............................ 5
  6.7 call ............................. 5
  6.8 dec ............................. 5
  6.9 div .............................. 5
  6.10 end ............................ 6
  6.11 inc ............................. 6
  6.12 jeq ............................. 6
  6.13 jge ............................. 6
  6.14 jgt ............................. 6
  6.15 jle ............................. 7
  6.16 jlt ............................. 7
  6.17 jmp ............................ 7
  6.18 jne ............................. 7
  6.19 load ............................ 7
  6.20 mod ............................. 8
  6.21 mov ............................. 8
  6.22 mul ............................. 8
  6.23 mv .............................. 8
  6.24 not ............................. 8
  6.25 or .............................. 8
  6.26 ret ............................. 9
  6.27 stop ............................ 9
  6.28 store ........................... 9
  6.29 sub ............................. 9
  6.30 xor ............................. 9
1 Syntaxdefinition

- Något som är skrivet i courier är ett nyckelord som måste vara med.
- Något som är skrivet i small caps är ett ord eller uttryck som måste ges
  men där det rätta värden bestäms av programmeraren, t ex variabelnamn.
  Dessa får inte vara samma som ett nyckelord.
- Något som är skrivet i italic är något som eventuellt kan förekomma.
- Något som är skrivet i normal stil i sematikbeskrivningarna är en uppda-
  tering av ett internt (ev. osynligt) register.
- \{a|b\} betyder antingen a eller b.
- NUMMER är vilket nummer som helst.
- VARIABEL är vilket variabel som helst.
- Om en stack (t ex returnstack) står på vänster sida av ett \leftarrow så innebär
  det att ett värde läggs på stacken, om den står till höger så innebär det
  att värdet hämtas från toppen av stacken och tas bort.

2 Symboliska adresser

Syntax:
ETIKETT
Semantik:
ETIKETT \leftarrow minnesposition

Sätter att LABEL pekar ut var i minnet ett visst stycke kod börjar. Dessa får
skrivas var som helst utom inne i instruktioner, t ex så går det alldeles utmärkt
att skriva snurra ret.

3 Register

3.1 $

Syntax:
$NUMMER

Siffran anger vilket register man vill använda.

3.2 sp

Syntax:
sp

Anger att man vill använda register 31.

3.3 zero

Syntax:
zero

Anger att man vill använda register 0.
4 Variabeldefinitioner

4.1 data

Syntax:
data nummer NAMN

Om ett nummer är givet kommer variabeln att ha detta värde vid början av
exekveringen av programmet, annars är värdena inte definierat.

5 Kommentarer

Syntax:
; text

All text på raden efter ett semikolon tas som en kommentar.

6 Instruktionsuppsättning

6.1 add

Syntax:
add REGISTER,REGISTER,{REGISTER|NUMMER|VARIABEL}

Semantik:
register ← register + {register|NUMMER|VARIABEL}

Målregistret får summan av det andra registret och det sista värdena.

6.2 and

Syntax:
and REGISTER,REGISTER,{REGISTER|NUMMER|VARIABEL}

Semantik:
register ← register \ {register|NUMMER|VARIABEL}

Målregistret får värdena hos den logiska konjunktionen av det andra registret
och det sista värdena.

6.3 band

Syntax:
band REGISTER,REGISTER,{REGISTER|NUMMER|VARIABEL}

Semantik:
register ← register .\ {register|NUMMER|VARIABEL}

Målregistret får värdena hos den bitvisa logiska konjunktionen av det andra
registret och det sista värdena.

6.4 bnot

Syntax:
bnot REGISTER,REGISTER

Semantik:
register ← .\ register

Den bitvisa logiska negationen av det andra registret läggs i det första.
6.5  bor

Syntax:
bor REGISTER,REGISTER,\{REGISTER|NUMMER|VARIABEL\}

Semantik:
register ← register \lor \{register|NUMMER|VARIABEL\}

Målregistret får värdet hos den bitvisa logiska disjunktionen av det andra registret och det sista värdet.

6.6  bxor

Syntax:
bxor REGISTER,REGISTER,\{REGISTER|NUMMER|VARIABEL\}

Semantik:
register ← register \xor \{register|NUMMER|VARIABEL\}

Målregistret får värdet hos den bitvisa logiska exklusiva disjunktionen av det andra registret och det sista värdet.

6.7  call

Syntax:
call ETIKETT

Semantik:
returstack ← pc
pc ← ETIKETT

Lägger addressen till nuvarande instruktion på returstacken och hoppar till den instruktion som ETIKETT pekar ut.

6.8  dec

Syntax:
dec REGISTER

Semantik:
register ← register - 1

Målregistret minskas med ett.

6.9  div

Syntax:
div REGISTER,REGISTER,\{REGISTER|NUMMER|VARIABEL\}

Semantik:
register ← register / \{register|NUMMER|VARIABEL\}

Målregistret får kvoten av det andra registret och det sista värdet.
6.10 end

Syntax:
end {\text{\texttt{ETIKETT}}}

Semantik:
$\text{pc} \leftarrow \text{ETIKETT}$

Slutet på källkodsfilen. Om \text{ETIKETT} är given skall processorn börja exekvera på denna instruktion, annars startar processorn med den absolut första instruktionen.

6.11 inc

Syntax:
inc register

Semantik:
register $\leftarrow$ register + 1

Målregistret ökas med ett.

6.12 jeq

Syntax:
jeq register, register, \text{ETIKETT}

Semantik:
\{\text{pc} \leftarrow \text{ETIKETT}\}

Om värdena i registren är lika hoppar programmet till den symboliska adressen \text{ETIKETT}, annars händer ingenting.

6.13 jge

Syntax:
jge register, register, \text{ETIKETT}

Semantik:
\{\text{pc} \leftarrow \text{ETIKETT}\}

Om värdena i det första registret är större än eller lika med värdet i det andra registret så hoppar programmet till \text{ETIKETT}, annars händer ingenting.

6.14 jgt

Syntax:
jgt register, register, \text{ETIKETT}

Semantik:
\{\text{pc} \leftarrow \text{ETIKETT}\}

Om värdena i det första registret är större än värdet i det andra registret så hoppar programmet till \text{ETIKETT}, annars händer ingenting.
6.15  jle

Syntax:
\texttt{jle \texttt{REGISTER,REGISTER,ETIKETT}}

Semantik:
\{\texttt{pc} \leftarrow \texttt{ETIKETT}\}

Om värdet i det första registret är mindre än eller lika med värdet i det andra registret så hoppar programmet till ETIKETT, annars händer ingenting.

6.16  jlt

Syntax:
\texttt{jlt \texttt{REGISTER,REGISTER,ETIKETT}}

Semantik:
\{\texttt{pc} \leftarrow \texttt{ETIKETT}\}

Om värdet i det första registret är mindre än värdet i det andra registret så hoppar programmet till ETIKETT, annars händer ingenting.

6.17  jmp

Syntax:
\texttt{jmp \texttt{ETIKETT}}

Semantik:
\texttt{pc} \leftarrow \texttt{ETIKETT}

Exekveringen fortsätter med instruktionen vid ETIKETT.

6.18  jne

Syntax:
\texttt{jne \texttt{REGISTER,REGISTER,ETIKETT}}

Semantik:
\{\texttt{pc} \leftarrow \texttt{ETIKETT}\}

Om vårdena i registren skiljer sig åt hoppar programmet till ETIKETT, annars händer ingenting.

6.19  load

Syntax:
\texttt{load \texttt{REGISTER},\{\texttt{REGISTER,NUMMER|NUMMER|VARIABEL}\}}

Semantik:
\texttt{register} \leftarrow \{\texttt{mem[register + NUM|NUM|VAR]}\}

Målregistret får antingen värdet som har angetts som ett omedelbart nummer, innehållet i variabeln eller innehållet i minnespositionen som pekas ut av registret + nummret.
6.20 mod

Syntax:
mod REGISTER,REGISTER,\{REGISTER\|NUMMER\|VARIABEL\}

Semantik:
register ← register \ \{register\|NUMMER\|VARIABEL\}
Målregistret får resten efter division av det andra registret och det sista värdet.

6.21 mov

Se mv

6.22 mul

Syntax:
mul REGISTER,REGISTER,\{REGISTER\|NUMMER\|VARIABEL\}

Semantik:
register ← register * \{register\|NUMMER\|VARIABEL\}
Målregistret får produkten av det andra registret och det sista värdet.

6.23 mv

Syntax:
\{mv\|mov\} REGISTER,REGISTER

Semantik:
register ← register
Värdet av det andra registret kopieras in i det första.

6.24 not

Syntax:
not REGISTER,REGISTER

Semantik:
register ← \neg register
Den logiska negationen av det andra registret läggs i det första.

6.25 or

Syntax:
or REGISTER,REGISTER,\{REGISTER\|NUMMER\|VARIABEL\}

Semantik:
register ← register \lor \{register\|NUMMER\|VARIABEL\}
Målregistret får värdet hos den logiska disjunktionen av det andra registret och det sista värdet.
6.26 ret

Syntax:
ret

Semantik:
pc ← returstack
    Exekveringen hoppar tillbaka från en subrutin.

6.27 stop

Syntax:
stop

Semantik:

    Exekveringen avslutas.

6.28 store

Syntax:
store REGISTER,{REGISTER,NUMMER|VARIABEL}

Semantik:
{mem[register + NUMMER]|VARIABEL} ← register
    Registret sparas i variabeln/minnespositionen som pekas ut av registret + nummret.

6.29 sub

Syntax:
sub REGISTER,REGISTER,{REGISTER|NUMMER|VARIABEL}

Semantik:
register ← register - {register|NUMMER|VARIABEL}
    Målregistret får differensen av det andra registret och det sista värden.

6.30 xor

Syntax:
xor REGISTER,REGISTER,{REGISTER|NUMMER|VARIABEL}

Semantik:
register ← register ⊕ {register|NUMMER|VARIABEL}
    Målregistret får värden hos den logiska exklusiva disjunktionen av det andra registret och det sista värden.