

Script generated by TTT

Title: Petter: Compiler Construction (02.07.2020)
 - 51: Arithmetic Expressions

Date: Wed Jul 01 15:08:40 CEST 2020

Duration: 23:04 min

Pages: 11

Using variables stored in registers; loading constants:

instruction	semantics	intuition
loadc $R_i c$	$R_i = c$	load constant
move $R_i R_j$	$R_i = R_j$	copy R_j to R_i

12/49

Translation of Simple Expressions

Using variables stored in registers; loading constants:

instruction	semantics	intuition
loadc $R_i c$	$R_i = c$	load constant
move $R_i R_j$	$R_i = R_j$	copy R_j to R_i

We define the following translation schema (with $\rho[x = a]$):

$$\begin{aligned} \text{code}_R^i c \rho &= \text{loadc } R_i c \\ \text{code}_R^i x \rho &= \text{move } R_i R_a \\ \text{code}_R^i x = e \rho &= \boxed{\text{code}_R^i e \rho} \\ &\quad \boxed{\text{move } R_a R_i} \end{aligned}$$

Translation of Expressions

Let $\text{op} = \{\text{add}, \text{sub}, \text{div}, \text{mul}, \text{mod}, \text{le}, \text{gr}, \text{eq}, \text{leq}, \text{geq}, \text{and}, \text{or}\}$. The **R-CMa** provides an instruction for each operator op.

$\text{op } R_i R_j R_k$

where R_i is the target register, R_j the first and R_k the second argument.

Correspondingly, we generate code as follows:

$$\text{code}_R^i e_1 \text{op } e_2 \rho = \boxed{\text{code}_R^i e_1 \rho} \\ \quad \boxed{\text{code}_R^{i+1} e_2 \rho} \\ \quad \boxed{\text{op } R_i R_i R_{i+1}}$$

12/49

13/49

Translation of Expressions

Let $\text{op} = \{\text{add}, \text{sub}, \text{div}, \text{mul}, \text{mod}, \text{le}, \text{gr}, \text{eq}, \text{leq}, \text{geq}, \text{and}, \text{or}\}$. The R-CMa provides an instruction for each operator op.

$\text{op } R_i R_j R_k$

where R_i is the target register, R_j the first and R_k the second argument.

Correspondingly, we generate code as follows:

$$\begin{aligned} \text{code}_R^i e_1 \text{ op } e_2 \rho &= \text{code}_R^i e_1 \rho \\ &\quad \text{code}_R^{i+1} e_2 \rho \\ &\quad \text{op } R_i R_i R_{i+1} \end{aligned}$$

Example: Translate $3 * 4$ with $i = 4$:

$$\begin{aligned} \text{code}_R^4 3 * 4 \rho &= \boxed{\text{code}_R^4 3 \rho} \\ &\quad \boxed{\text{code}_R^4 4 \rho} \\ &\quad \boxed{\text{mul } R_4 R_4 R_5} \end{aligned}$$

13/49

Translation of Expressions

Let $\text{op} = \{\text{add}, \text{sub}, \text{div}, \text{mul}, \text{mod}, \text{le}, \text{gr}, \text{eq}, \text{leq}, \text{geq}, \text{and}, \text{or}\}$. The R-CMa provides an instruction for each operator op.

$\text{op } R_i R_j R_k$

where R_i is the target register, R_j the first and R_k the second argument.

Correspondingly, we generate code as follows:

$$\begin{aligned} \text{code}_R^i e_1 \text{ op } e_2 \rho &= \text{code}_R^i e_1 \rho \\ &\quad \text{code}_R^{i+1} e_2 \rho \\ &\quad \text{op } R_i R_i R_{i+1} \end{aligned}$$

Example: Translate $3 * 4$ with $i = 4$:

$$\begin{aligned} \text{code}_R^4 3 * 4 \rho &= \text{loadc } R_4 3 \\ &\quad \text{loadc } R_5 4 \\ &\quad \boxed{\text{mul } R_4 R_4 R_5} \end{aligned}$$

13/49

Managing Temporary Registers

Observe that temporary registers are re-used: translate $3 * 4 + 3 * 4$ with $t = 4$:

$$\begin{aligned} \text{code}_R^4 3 * 4 + 3 * 4 \rho &= \text{code}_R^4 3 * 4 \rho \\ &\quad \text{code}_R^4 3 * 4 \rho \\ &\quad \boxed{\text{add } R_4 R_4 R_5} \end{aligned}$$

where

$$\begin{aligned} \text{code}_R^4 3 * 4 \rho &= \boxed{\text{loadc } R_i 3} \\ &\quad \boxed{\text{loadc } R_{i+1} 4} \\ &\quad \boxed{\text{mul } R_i R_i R_{i+1}} \end{aligned}$$

we obtain

$$\text{code}_R^4 3 * 4 + 3 * 4 \rho =$$

14/49

Managing Temporary Registers

Observe that temporary registers are re-used: translate $3 * 4 + 3 * 4$ with $t = 4$:

$$\begin{aligned} \text{code}_R^4 3 * 4 + 3 * 4 \rho &= \boxed{\text{code}_R^4 3 * 4 \rho} \\ &\quad \boxed{\text{code}_R^5 3 * 4 \rho} \\ &\quad \boxed{\text{add } R_4 R_4 R_5} \end{aligned}$$

where

$$\begin{aligned} \text{code}_R^4 3 * 4 \rho &= \text{loadc } R_i 3 \\ &\quad \text{loadc } R_{i+1} 4 \\ &\quad \boxed{\text{mul } R_i R_i R_{i+1}} \end{aligned}$$

we obtain

$$\begin{aligned} \text{code}_R^4 3 * 4 + 3 * 4 \rho &= \text{loadc } R_4 3 \\ &\quad \text{loadc } R_5 4 \\ &\quad \boxed{\text{mul } R_4 R_4 R_5} \\ &\quad \boxed{\text{loadc } R_5 3} \\ &\quad \text{loadc } R_6 4 \\ &\quad \boxed{\text{mul } R_5 R_5 R_6} \\ &\quad \boxed{\text{add } R_4 R_4 R_5} \end{aligned}$$

14/49

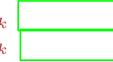
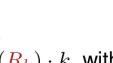
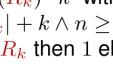
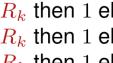
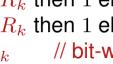
Semantics of Operators

The operators have the following semantics:

<code>add $R_i R_j R_k$</code>	$R_i = R_j + R_k$
<code>sub $R_i R_j R_k$</code>	$R_i = R_j - R_k$
<code>div $R_i R_j R_k$</code>	$R_i = R_j / R_k$
<code>mul $R_i R_j R_k$</code>	$R_i = R_j * R_k$
<code>mod $R_i R_j R_k$</code>	$R_i = \text{signum}(R_k) \cdot k \text{ with } R_j = n \cdot R_k + k \wedge n \geq 0, 0 \leq k < R_k $
<code>le $R_i R_j R_k$</code>	$R_i = \text{if } R_j < R_k \text{ then } 1 \text{ else } 0$
<code>gr $R_i R_j R_k$</code>	$R_i = \text{if } R_j > R_k \text{ then } 1 \text{ else } 0$
<code>eq $R_i R_j R_k$</code>	$R_i = \text{if } R_j = R_k \text{ then } 1 \text{ else } 0$
<code>leq $R_i R_j R_k$</code>	$R_i = \text{if } R_j \leq R_k \text{ then } 1 \text{ else } 0$
<code>geq $R_i R_j R_k$</code>	$R_i = \text{if } R_j \geq R_k \text{ then } 1 \text{ else } 0$
<code>and $R_i R_j R_k$</code>	$R_i = R_j \& R_k \quad // \text{bit-wise and}$
<code>or $R_i R_j R_k$</code>	$R_i = R_j R_k \quad // \text{bit-wise or}$

Semantics of Operators

The operators have the following semantics:

<code>add $R_i R_j R_k$</code>	$R_i = R_j + R_k$	
<code>sub $R_i R_j R_k$</code>	$R_i = R_j - R_k$	
<code>div $R_i R_j R_k$</code>	$R_i = R_j / R_k$	
<code>mul $R_i R_j R_k$</code>	$R_i = R_j * R_k$	
<code>mod $R_i R_j R_k$</code>	$R_i = \text{signum}(R_k) \cdot k \text{ with } R_j = n \cdot R_k + k \wedge n \geq 0, 0 \leq k < R_k $	
<code>le $R_i R_j R_k$</code>	$R_i = \text{if } R_j < R_k \text{ then } 1 \text{ else } 0$	
<code>gr $R_i R_j R_k$</code>	$R_i = \text{if } R_j > R_k \text{ then } 1 \text{ else } 0$	
<code>eq $R_i R_j R_k$</code>	$R_i = \text{if } R_j = R_k \text{ then } 1 \text{ else } 0$	
<code>leq $R_i R_j R_k$</code>	$R_i = \text{if } R_j \leq R_k \text{ then } 1 \text{ else } 0$	
<code>geq $R_i R_j R_k$</code>	$R_i = \text{if } R_j \geq R_k \text{ then } 1 \text{ else } 0$	
<code>and $R_i R_j R_k$</code>	$R_i = R_j \& R_k \quad // \text{bit-wise and}$	
<code>or $R_i R_j R_k$</code>	$R_i = R_j R_k \quad // \text{bit-wise or}$	

Note: all registers and memory cells contain operands in Z

15/49

15/49

Translation of Unary Operators

Unary operators $\text{op} = \{\text{neg}, \text{not}\}$ take only two registers:

$$\text{code}_R^i \text{ op } e \rho = \text{code}_R^i e \rho \\ \text{op } R_i R_i$$

Note: We use the same register.



15/49

15/49

Translation of Unary Operators

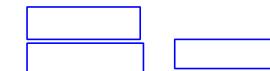
Unary operators $\text{op} = \{\text{neg}, \text{not}\}$ take only two registers:

$$\text{code}_R^i \text{ op } e \rho = \text{code}_R^i e \rho \\ \text{op } R_i R_i$$

Note: We use the same register.

Example: Translate -4 into R_5 :

$$\text{code}_R^5 -4 \rho = \text{loadc } R_5 4 \\ \text{neg } R_5 R_5$$



16/49

16/49