Einführung in den Compilerbau

Prof. Dr.-Ing. Andreas Koch
David Volz, Tim Noack, Jan Braun

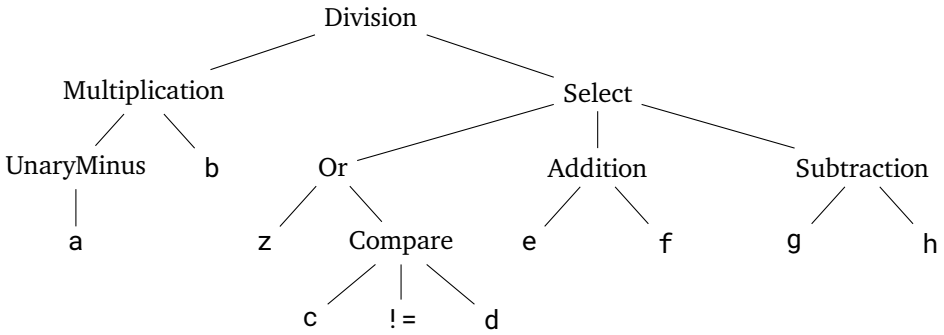
TECHNISCHE UNIVERSITÄT DARMSTADT

Theorie
Wintersemester 23/24
Übungsblatt 3

Abgabe bis Sonntag, 14.01.2024, 18:00 Uhr (MEZ)

Aufgabe 3.1: Ausdrucksauswertung (15 Punkte)

Gegeben sei der folgende Ausdruck:



Hinweis: Der Select-Knoten hat folgende Kindknoten (von links nach rechts): „Condition“, „True“, „False“.

3.1a) Typische Stackmaschine (6 Punkte)

Erzeugen Sie für den Ausdruck eine Instruktionsfolge für eine typische Stackmaschine (vgl. Folie 33, Foliensatz „Laufzeitorganisation“). Nach der Verarbeitung der letzten Instruktion soll das Ergebnis der Ausdrucksauswertung als oberstes Element auf dem Stack liegen. Verwenden Sie folgende Instruktionen:

```
LOAD v    Hole den Wert der Variablen 'v' und lege ihn auf den Stack.

OR        Ersetze die zwei obersten Werte auf dem Stack durch ihr bitweises Oder:
          value(ST') ← value(ST-1) | value(ST)

CMP_NE    Ersetze die zwei obersten Werte auf dem Stack durch das Ergebnis des
          Ungleich-Vergleichs:
          value(ST') ← 1 wenn value(ST-1) != value(ST), 0 sonst

ADD       Ersetze die zwei obersten Werte auf dem Stack durch ihre Summe:
          value(ST') ← value(ST-1) + value(ST)

SUB       Ersetze die zwei obersten Werte auf dem Stack durch ihre Differenz:
          value(ST') ← value(ST-1) - value(ST)

MUL       Ersetze die zwei obersten Werte auf dem Stack durch ihr Produkt:
          value(ST') ← value(ST-1) * value(ST)
```

```
DIV     Ersetze die zwei obersten Werte auf dem Stack durch ihren Quotienten:
        value(ST') ← value(ST-1) / value(ST)

NEG     Ersetze den obersten Wert auf dem Stack durch seine Negation:
        value(ST') ← -1 * value(ST)

SEL     Ersetze die obersten drei Werte auf dem Stack durch den dritten Wert,
        falls der erste 0 ist, ansonsten durch den zweiten Wert:
        value(ST') ← value(ST-2) ? value(ST-1) : value(ST)
```

3.1b) Ershov-Zahlen (3 Punkte)

Informieren Sie sich über die sog. „Ershov-Zahlen“[1] und geben Sie diese für alle Knoten im oben gegebenen Ausdrucksbaum an.

3.1c) Registermaschine (6 Punkte)

Erzeugen Sie für den Ausdruck eine Instruktionsfolge für eine imaginäre Registermaschine. Die Maschine habe 8 Register `r1, r2, ..., r8` und unterstütze die folgenden Instruktionen:

```
rx = ld v           // Hole den Wert der Variablen 'v' und schreibe ihn in das Register 'rx'.

rx = or ra, rb      // Schreibe das Ergebnis des bitweisen Oders 'ra'|'rb' nach 'rx'.

rx = cmp_ne ra, rb  // Schreibe das Ergebnis des Kleiner-Vergleichs 'ra'!='rb' nach 'rx'.

rx = add ra, rb     // Schreibe das Ergebnis der Addition 'ra'+'rb' nach 'rx'.

rx = sub ra, rb     // Schreibe das Ergebnis der Subtraktion 'ra'-'rb' nach 'rx'.

rx = mul ra, rb     // Schreibe das Ergebnis der Multiplikation 'ra'*'rb' nach 'rx'.

rx = div ra, rb     // Schreibe das Ergebnis der Division 'ra'/'rb' nach 'rx'.

rx = neg ra         // Schreibe das Ergebnis der Negation -1*'ra' nach 'rx'.

rx = sel ra, rb, rc // Schreibe das Ergebnis der Selektion 'ra'?'rb':'rc' nach 'rx'.

// Die Quell- und das Zielregister können identisch sein.
```

Verwenden Sie in Ihrer Instruktionsfolge **so wenige Register wie möglich**. Nach der Verarbeitung der letzten Instruktion soll das Ergebnis der Ausdrucksauswertung in `r1` stehen.

[1] z.B. https://de.wikipedia.org/wiki/Ershov-Zahl

Aufgabe 3.2: MTAM-Routinenprotokoll (12 Punkte)

In den folgenden Teilaufgaben sind MAVL-Programme und ihre Übersetzung in Assemblercode für die MTAM gegeben. Sie sollen zu jedem dieser Programme den Stackaufbau **wortweise** zu einem gegebenen Zeitpunkt ermitteln[2]. Geben Sie für jedes Stackelement dessen Inhalt **und** seine Bedeutung (z.B. „Speicherort von Variable v“, „2. Argument für Funktion foo“, usw.) an. Handelt es sich beim Inhalt um eine Adresse, so verwenden Sie die in der TAM-Assemblerdarstellung übliche Offsetform (z.B. „`42[SB]`“, „`11[CB]`“). Der Stack soll von oben nach unten wachsen, d.h. die erste Zeile in Ihrer Lösung entspricht der Adresse `0[SB]`, die zweite Zeile hat die Adresse `1[SB]` usw.

3.2a) MTAM-Routinenprotokoll (4 Punkte)

Betrachten Sie folgendes MAVL-Programm und die zugehörige Übersetzung in Assemblercode für die MTAM. Geben Sie den Stackaufbau zum Zeitpunkt **vor** Ausführung der `CALL` instruction in `19[CB]` an.

```
function int calculate(int a, int b, int c) {
  return a * b + c;
}

function void main() {
  val int x = 3;
  val int y = 7;
  var int z;
  z = 42 - 13;
  val int n = calculate(x, y, z);
}
```

```
00: CALL          8[CB]     ; main
01: HALT

calculate:
02: LOAD   (1)    -3[LB]    ; a
03: LOAD   (1)    -2[LB]    ; b
04: mulI
05: LOAD   (1)    -1[LB]    ; c
06: addI
07: RETURN (1)    3

main:
08: LOADL         3
09: LOADL         7
10: PUSH          1         ; z
11: LOADL         42
12: LOADL         13
13: subI
14: LOADA         4[LB]     ; z
15: STOREI (1)
16: LOAD   (1)    2[LB]     ; x
17: LOAD   (1)    3[LB]     ; y
18: LOAD   (1)    4[LB]     ; z
19: CALL          2[CB]     ; calculate
20: RETURN (0)    0
```

[2] Orientieren Sie sich hierbei an der MTAM Spezifikation

3.2b) MTAM-Routinenprotokoll (8 Punkte)

Betrachten Sie folgendes MAVL-Programm und die zugehörige Übersetzung in Assemblercode für die MTAM. Geben Sie den Stackaufbau zum Zeitpunkt **vor** Ausführung der `RETURN` instruction in `06[CB]` an.

```
function matrix<int>[2][2] bar(matrix<int>[2][2] mat)
{
  return ~mat;
}

function matrix<int>[2][2] foo(int s)
{
  return bar([[s, 0], [0, s]]);
}

function void main()
{
  var matrix<int>[2][2] x;
  x = foo(5);
}
```

```
00: CALL          13[CB]    ; main
01: HALT

bar:
02: LOAD   (4)    -4[LB]    ; mat
03: LOADL         2
04: LOADL         2
05: matTranspose
06: RETURN (4)    4

foo:
07: LOAD   (1)    -1[LB]    ; s
08: LOADL         0
09: LOADL         0
10: LOAD   (1)    -1[LB]    ; s
11: CALL          2[CB]     ; bar
12: RETURN (4)    1

main:
13: PUSH          4         ; x
14: LOADL         5
15: CALL          7[CB]     ; foo
16: LOADA         2[LB]     ; x
17: STOREI (4)
18: RETURN (0)    0
```

Aufgabe 3.3: Speicherlayout (5 Punkte)

3.3a) Speicherlayout (2 Punkte)

Geben Sie für **alle** Zuweisungen in der folgenden MAVL-Funktion die Adresse des Zuweisungsziels als Offset zum Beginn des Stackframes (z.B. `42[LB]`) an.

```
function void main() {
  var vector<int>[128] a;
  var matrix<float>[7][11] b;
  a[69] = 54;
  b[5][8] = 27.5;
}
```

3.3b) Speicherlayout (3 Punkte)

Geben Sie für **alle** Zuweisungen in der folgenden MAVL-Funktion die Adresse des Zuweisungsziels als Offset zum Beginn des Stackframes (z.B. `42[LB]`) an.

```
record rec {
  val matrix<int>[3][3] m;
  var bool b;
  var int i;
}

function void main() {
  var vector<int>[16] v;
  var int a;
  var rec r;
  r = @rec[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], true, 54];
  a = 80;
  var int b;
  r@b = false;
}
```

Aufgabe 3.4: MAVL ↔ MTAM (8 Punkte)

Gegeben sei folgendes MAVL-Programm und die zugehörige Übersetzung in Assemblercode für die MTAM:

```
1   function int mac(int a, int b, int c){
2     // missing (1)
3     z = a * b + c;
4     return z;
5   }
6
7   function bool bar(float s, float t){
8     // missing (2)
9     val bool n = s < t;
10    return m | n;
11  }
12
13  function bool foo(){
14    val float pi = 3.14;
15    val float e = 2.718;
16    return bar(e, pi^2.0);
17  }
18
19  function void main(){
20    var int d;
21    var int e;
22    // missing (3)
23    d = foo() ? 42 : e;
24  }
```

```
00: CALL          28[CB]     ; main
01: HALT
mac:
02: PUSH          1          ; z
03: LOAD   (1)    -3[LB]     ; a
04: LOAD   (1)    -2[LB]     ; b
05: mulI
06: LOAD   (1)    -1[LB]     ; c
07: addI
08: LOADA         2[LB]      ; z
09: STOREI (1)
10: LOAD   (1)    2[LB]      ; z
11: ; missing (4)
bar:
12: LOADL         1
13: LOAD   (1)    -2[LB]     ; s
14: LOAD   (1)    -1[LB]     ; t
15: ; missing (5)
16: LOAD   (1)    2[LB]      ; m
17: LOAD   (1)    3[LB]      ; n
18: or
19: RETURN (1)    2
foo:
20: LOADL         107852333
21: LOADL         107675333
22: LOAD   (1)    2[LB]      ; e
23: LOAD   (1)    3[LB]      ; pi
24: LOADL         107374182
25: powFloat
26: ; missing (6)
27: RETURN (1)    0
main:
28: PUSH          1          ; d
29: PUSH          1          ; e
30: LOADL         5
31: LOADL         13
32: LOADL         27
33: CALL          2[CB]      ; mac
34: LOADA         3[LB]      ; e
35: STOREI (1)
36: CALL          20[CB]     ; foo
37: JUMPIF (0)    40[CB]
38: LOADL         42
39: ; missing (7)
40: LOAD   (1)    3[LB]      ; e
41: LOADA         2[LB]      ; d
42: STOREI (1)
43: RETURN (0)    0
```

Sowohl im MAVL-Programm als auch im Assemblercode fehlen einige Zeilen (gekennzeichnet mit `missing`). Ihre Aufgabe ist es die sieben fehlenden Zeilen mithilfe der dazu passenden Übersetzung zu rekonstruieren. Es dürfen keine zusätzlichen Zeilen eingefügt werden! Nutzen Sie in Ihrer Lösung bitte die Nummer hinter `missing`, um deutlich zu machen auf welche Zeile Sie sich beziehen.

Hinweis: Im Assemblercode wird `true` als `1` und `false` als `0` dargestellt.