Prof. Dr. Christoph Scholl
Dr. Tim Welschehold
Alexander Konrad
Niklas Wetzel

Freiburg, 25.11.2022

Betriebssysteme
Musterlösung zu Übungsblatt 6

Aufgabe 1 (6 Punkte)

Betrachten Sie folgendes Pico-C Programm mit Deklarations- und Anweisungsteil.

```
void main()
{
   //Deklarationsteil
   int x;
   int y;
   const int z = 2;

   //Anweisungsteil
   y = 3;
   x = 15;

   while(x >= z * y)
   {
      x = x - 3;
   }
}
```

Erstellen Sie die Einträge der Symboltabelle für den Deklarationsteil (nehmen Sie an, dass *bds*= 128) und übersetzen sie obiges Programm in eine Befehlsfolge der erweiterten ReTI. Werten Sie die Ausdrücke und Anweisungsfolgen aus, wie Sie es in der Vorlesung gelernt haben. Versehen Sie Ihr Programm mit Kommentaren, die erklären, was Sie mit den entsprechenden Programmzeilen erzielen wollen. Nehmen Sie weiter an, dass *eds*= 256 (dezimal). Gehen Sie dabei wie in Kapitel 3 der Vorlesung der Einfachheit halber davon aus, dass an die ReTI keine anderen Speicher als der Hauptspeicher angeschlossen sind und die Adressen des Hauptspeichers bei 0 beginnen. DS ist mit 0^{10} vorbelegt. Wenn mehrere Speicher angeschlossen wären und das Memory Mapping wäre wie in Kapitel 2.3 angegeben, dann müssten die Binärdarstellungen von *bds* und *eds* natürlich mit "10 . . ." beginnen.

Lösung:

Symboltabelle:
st(x) = (var,int,128)

st(y) = (var,int,129)
st(z) = (const,int,'2')

```
/** Zuweisungen (Konstanten auf rechter Seite), Adressen ergeben sich aus Symboltabelle **/
LOADI SP eds          //erste freie Stack-Zelle initial am Ende des Datensegments
SUBI SP 1             // Zuweisung y=3 beginnt.
LOADI ACC 3           //Konstante 3
STOREIN SP ACC 1
LOADIN SP ACC 1
ADDI SP 1
STORE ACC 129         //Adresse von y. Zuweisung y endet.
SUBI SP 1             //Zuweisung x = 15 beginnt.
LOADI ACC 15          //Konstante 15
STOREIN SP ACC 1
LOADIN SP ACC 1
ADDI SP 1
STORE ACC 128         // Adresse von x Zuweisung x endet.
/** Stack ist wieder leer, Variablen in Hauptspeicher belegt **/
/** while-Schleife **/
/** x **/
SUBI SP 1             // Variablenbezeichner x, Adresse 128
LOAD ACC 128
STOREIN SP ACC 1      // x auf Stack
/** z * y **/
SUBI SP 1             // Variablenbezeichner z, Konstante
LOADI ACC 2
STOREIN SP ACC 1      // z auf Stack
SUBI SP 1             // Variablenbezeichner y, Adresse 129
LOAD ACC 129
STOREIN SP ACC 1      // y auf Stack
LOADIN SP ACC 2       // Wert von z in ACC laden
LOADIN SP IN2 1       // Wert von y in IN2 laden
MUL ACC IN2           // z * y in ACC laden
STOREIN SP ACC 2      // Ergebnis in zweitoberste Stack-Zelle
ADDI SP 1 ;
/** x >= z * y **/
LOADIN SP ACC 2       // Wert von x in ACC laden
LOADIN SP IN2 1       // Wert von z * y in IN2 laden
SUB ACC IN2           // x - z * y in ACC laden
JUMP>= 3              // Ergebnis 1, wenn x - z * y >= 0 wahr
LOADI ACC 0           // Ergebnis 0, wenn wenn x - z * y >= 0 falsch
JUMP 2
LOADI ACC 1           // Ergebnis 1, wenn x - z * y >= 0 wahr
STOREIN SP ACC 2      // Ergebnis in zweitoberste Stack-Zelle
ADDI SP 1 ;
LOADIN SP ACC 1       // Wert von x - z * y >= 0 in ACC laden
ADDI SP 1
JUMP= 16              // Inhalt überspringen, wenn x - z * y >= 0 üunerfllt
/** x = x -3; **/
/** x - 3 **/
SUBI SP 1
LOAD ACC 128          // Adresse von x ist 128
STOREIN SP ACC 1
SUBI SP 1
LOADI ACC 3           // Konstante 3
STOREIN SP ACC 1
```

```
LOADIN SP ACC 2       // Wert von x in ACC laden
LOADIN SP IN2 1       // 3 in IN2 laden
SUB ACC IN2           // x - 3 in ACC laden
STOREIN SP ACC 2      // Ergebnis in zweitoberste Stack-Zelle
ADDI SP 1 ;
LOADIN SP ACC 1       // Wert von x - 3 in ACC laden
ADDI SP 1
STORE ACC 128         // an Adresse 128 von x speichern
JUMP -40;             //Sprung zum Anfang der Schleife
JUMP 0                //HALT-Befehl
```

Keine Punkte für Symboltabelle (gab es letztes Blatt schon). [1.5P] für Zuweisungsteil, [3P] für korrekte while-Schleifen Behandlung, [1.5P] für x = x - 3 Teil (Anweisungsfolge innerhalb der Schleife).

Aufgabe 2 (6 Punkte)

In der Vorlesung wurde Pico-C unter anderem um Felder (Arrays) erweitert.

Betrachten Sie den allgemeinen Fall eines Feldes das mit $t\ ab[s_1][s_2]\dots[s_n]$ deklariert wurde. Der Zugriff auf ein Element erfolgt durch $ab[e_1]\dots[e_n]$ mit arithmetischen Ausdrücken $e_1,\dots,e_n$. Schreiben Sie ein Programm mit Befehlen der erweiterten RETI, das die e_i zum jeweiligen Wert $val(e_i)$ auswertet und die Adresse $a + \sum_{i=1}^{n} l_i \cdot val(e_i)$ des indizierten Elements berechnet und in IN1 abspeichert. Hierbei sei $l_i = \prod_{j=i+1}^{n} s_j$ wie in der Vorlesung. Die l_i ergeben sich also aus dem Programmcode und können somit zur Übersetzungszeit als Konstanten angenommen werden. Sie dürfen annehmen, dass der Typ des Feldes t in eine Speicherzelle passt, also $size(t) = 1$ gilt. Sie dürfen außerdem $\text{code}^{aa}(e_i)$ (so wie dieser Befehl in der Vorlesung definiert wurde) als abkürzende Schreibweise für die Auswertung der e_i benutzen. **Gehen Sie für Ihr Programm von den konkreten Werten $n = 3, s_1 = 4, s_2 = 2, s_3 = 3$ aus.** Versehen Sie Ihr Programm mit Kommentaren, die erklären, was Sie mit den entsprechenden Programmzeilen erzielen wollen.

Lösung:

```
code^aa (e_1)
code^aa (e_2)
code^aa (e_3)
/** Die Auswertung der Ausdruecke liegt nun auf den oberen 3 Stackzellen **/
/** Die l_i liegen zur Übersetzungszeit als Konstanten vor: l_1 = 6, l_2 = 3, l_3 = 1**/
LOADI IN1 a   // IN1 auf a setzen (hier wird die Adresse "gesammelt")
LOADIN SP ACC 1 // ACC enthaelt val(e_3)
ADDI SP 1     // Stack um eins üverkrzen.
ADD IN1 ACC     // Auf die bestehende Summe draufaddieren (l_3=1, also ist hier keine Multiplikation öntig)
LOADIN SP ACC 1 // ACC enthaelt val(e_2)
ADDI SP 1     // Stack um eins üverkrzen.
MULTI ACC 3   // Mult. mit dem korrespondierenden Faktor l_2=3
ADD IN1 ACC     // Auf die bestehende Summe draufaddieren
LOADIN SP ACC 1 // ACC enthaelt val(e_1)
ADDI SP 1     // Stack um eins üverkrzen.
MULTI ACC 6   // Mult. mit dem korrespondierenden Faktor l_1=6
ADD IN1 ACC     // Jetzt enthaelt IN1 die Adresse. Ende
```

Bepunktung:

- +[1] für korrekte Nutzung von $code^{aa}(e_i)$
- +[1] für a im Ergebnis enthalten
- +[1] für korrektes Reduzieren des Stacks nachdem vom Stack geladen wurde
- +[1] für Multiplikation mit l_i an der richtigen Stelle
- +[1] für korrekte Konstanten für l_i genutzt
- +[1] für korrektes Aufaddieren der Adresse nach und nach auf IN1

Aufgabe 3 (3+6+1 Punkte)

Betrachten Sie folgendes Pico-C-Programm.

```
#include <stdlib.h> //Stellt Bibliotheksfunktionen malloc() und free() bereit
```

```
void main()
{
   struct point
   {
      int x;
      int y;
   };

   // Annahme Symboltabelleneintrag st(p1) = (var, struct point*, 8)
   struct point *p1;
   // Annahme Symboltabelleneintrag st(p3) = (var, struct point*, 9)
   struct point *p3;
   // Annahme Symboltabelleneintrag st(a) = (var, int*, 10)
   int* a;

   //Annahme Symboltabelleneintrag:
   //st(p2) = (struct, x -> (int,0), y -> (int,1), 15)
   struct point p2;

   a = &(p2.x);
   p2.x = 7;
   p2.y = 4;

   /*** MARKE 1 ***/

   //Annahme: reserviert zusammenhaengenden Bereich auf dem Heap ab Adresse 33
   p1 = (struct point *) malloc(sizeof(struct point));

   (*p1).y = *a;

   p3 = p1;
   p1 = &p2;

   /*** MARKE 2 ***/

   if((*p1).y > 5)
   {
      *a = 42;
   }
   else
   {
      *a = 1;
   };

   /*** MARKE 3 ***/

   free(p3);
};
```

a ist ein Pointer auf einen Integer

p1->y enthält danach den Wert 7

p1 ist bereits ein Pointer, p2 muss erst zu einem umgewandelt werden

weil p1 jetzt die Adresse von p2 enthält

free nimmt einen Pointer auf den Heap

weil p3 jetzt die Adresse von p1 hält

a) Zählen Sie alle Speicheradressen auf, die beim Ablauf dieses Programms gelesen bzw. geschrieben werden (beachten Sie die Kommentare im Programmcode). Begründen Sie Ihre Antwort.

b) Erstellen sie einen Speicherabzug mit den relevanten Speicheradressen nach Erreichen der Marken 1, 2 und 3, indem Sie den Inhalt der Speicherzellen aus a) benennen. Undefinierte

Speicherzellen können Sie entsprechend kennzeichnen. Begründen Sie Ihre Antwort.

c) Ist die letzte Zeile `free(p3);` zulässig? Falls ja, welcher Speicherbereich wird hier freigegeben? Falls nein, weshalb?

Hinweis: Nehmen Sie wie in der Vorlesung an, dass Code für die RETI erzeugt wird, die 4-Byte-Worte adressiert und nicht einzelne Bytes. Ein `int` passe in eine Speicherzelle.

Lösung:

a) 8, 9, 10, 15, 16, 34 (33 wird weder gelesen noch geschrieben, von daher nicht nötig hier aufzuzählen. Würde es aber auch nicht als Fehler zählen, wenn die 33 trotzdem mitaufgezählt wird.)
[0.5P] pro richtig aufgezählte Adresse, max. [3P].

b) **Marke** 1
8: undefiniert
9: undefiniert
10: 15
15: 7
16: 4
33: undefiniert
34: undefiniert

Marke 2
8: 15
9: 33
10: 15
15: 7
16: 4
33: undefiniert
34: 7

Marke 3
8: 15
9: 33
10: 15
15: 1
16: 4
33: undefiniert
34: 7

[2P] pro richtige Marke. [-0.5P] pro Fehler bei einer Marke. Bei undefinierten muss natürlich nicht exakt undefiniert stehen, aber etwas sinngemäßes oder einfach leer gelassen ginge auch.

c) Ja, da p3 auf den Speicherbereich zeigt, der mit malloc() reserviert wurde, also Speicheradressen 33 und 34 werden wieder freigegeben.
[0.5P] für richtige Antwort: Ja. [0.5P] für Adressen 33 und 34, die freigegeben werden.

Abgabe: als PDF im Übungsportal bis 02.12.2022 um 12:00