Betriebssysteme

Übungsblatt 6

Micha Erkel Felix Ruh

Aufgabe 1

a) Symboltabelle:
st(x) = (var, int, 128)
st(y) = (var, int, 129)
st(z) = (const, int, 2)

b) Der Code:

SUBI SP 2	Stelle den SP auf Zelle 130.
LOADI ACC 3	Lade 3 in den ACC.
STOREIN ACC SP 1	Speiche die 3 für y in 129 ab.
LOADI ACC 15	Lade 15 in den ACC.
STOREIN ACC SP 2	Speichere die 15 für x in 128 ab.
SUBI SP 1	Erhöhe den Stack um 1, auf 131.
LOADI IN2 z	Lade z in den ACC, dabei gilt z = 2
LOADIN ACC SP 2	Lade y aus dem Speicher in den ACC.
MUL IN2 ACC	Multipliziere y im ACC mit z = 2.
STOREI IN2 SP 1	Speicher das Ergebnis in Zelle 130.
LOADIN ACC SP 1	Lade Ergebnis der Multiplikation in den ACC
LOADIN IN1 SP 3	Lade x aus dem Speicher in IN1.
SUB ACC IN1	Subtrahiere x vom vorherigen Ergebnis
JUMP> 5	Falls x kleiner war, wird Ergebnis positiv -> Bedingung nicht erfüllt, überspringe Schleife
LOADIN ACC SP3	Lade x aus dem Speicher in den ACC.
SUBI ACC 3	Subtrahiere x mit 3
STOREIN ACC SP 3	Speicher den neuen Wert in x ab.
JUMP -7	Springe zur Schleifenbedingung
JUMP 0	Beende das Programm.

Aufgabe 2

In der Beschreibung der Aufgabe - weitgehend auch in der Vorlesung - wurden einige Dinge nicht klar gestellt, welche wir für durchaus notwendig erachten. Beispielsweise ist nirgends erklärt, woher der Zugriff in ab[e1]...[en] kommt oder wie/ wo dieser gespeichert ist. Wie soll auf die einzelnen e1 zugegriffen werden, wo gespeichert werden, welche Form haben diese Einträge überhaupt?
Wir haben uns dazu entschlossen, diese und alle weiteren Unzulänglichkeiten zu ignorieren und das beste gemacht aus dem was wir hatten. Alles in allem aber eine interessante Aufgabenstellung!

Hab euer Programm mit dem RETI-Interpreter lauffähtig gemahct. Ihr könnt eure Korrektur hier finden: https://github.com/matthejue/Abgaben_Blatt_3/blob/main/Blatt6/walnuss.reti. Führt eure Korrektur mit dem Befehl picoc_compiler -D 50 walnuss.reti -S -P 5 aus.

4.5/6

$l_1 = s_2 * s_3 = 6$	NEBENRECHNUNG
$l_2 = s_3 = 3$	NEBENRECHNUNG
$l_3 = 1$	NEBENRECHNUNG
$code^{aa}(e_1)$	berechne e_1
$code^{aa}(e_2)$	berechne e_1
$code^{aa}(e_3)$	berechne e_1
LOAD ACC SP 3	lade e_1
MULI ACC 6	multipliziere e_1 mit l_1
MOVE ACC IN1	Speicher das aktuelle Zwischenergebniss in IN1
LOAD ACC SP 2	lade e_2
MULI ACC 3	multipliziere e_2 mit l_2
ADDI IN1 ACC	Addiere das obige Produkt auf das aktuelle Zwischenergebniss
LOAD ACC SP 1	lade e_3
MULI ACC 1	multipliziere e_3 mit l_3
ADDI IN1 ACC	Addiere das obige Produkt auf das aktuelle Zwischenergebniss
ADDI IN1 a	Addiere a auf das aktuelle Zwischenergebniss
JUMP 0	Beendet das Programm.

Aufgabe 3

a) Alle, im Programm verwendeten, Speicherzellen:

10	da mit dem Befehl a = &(p2.x); der Speicherzelle 10 ein Inhalt zugewiesen wird.
15	da mit dem Befehl a = &(p2.x); der Inhalt der Speicherzelle 15 eingelesen wird.
16	da mit dem Befehl p2.y = 4; der Speicherzelle 16 ein Inhalt zugewiesen wird.
8	da mit dem Befehl p1 = (struct point *) malloc(sizeof(struct point)); der Speicherzelle 8 ein Inhalt zugewiesen wird.
34	da mit dem Befehl (*p1).y = *a; der Speicherzelle 34 ein Inhalt zugewiesen wird.
9	da mit dem Befehl p3 = p1; der Speicherzelle 9 ein Inhalt zugewiesen wird.

b) Der Speicherabzug:

Marke 1:

...	
8	undefined
9	undefined
10	15 (adresse von p2.x)
15	7
16	4
...	

Marke 2:

...	
8	15 (adresse von p2.x)
9	33
10	15 (adresse von p2.x)
15	7
16	4
33	undefined
34	7
...	

Marke 3:

...	
8	15 (adresse von p2.x)
9	33
10	15 (adresse von p2.x)
15	1
16	4
33	undefined
34	7
...	

c) Der Befehl ist so zulässig. Dabei wird p3 gelöscht, also Zelle 9 wieder freigegeben.