Aufgabenblatt 1 Zahlensysteme

1 Zahlendarstellung

Laden Sie die Zahlen #0xFFFFFFF5 , 4294967285, -11, -0xB, ~10 sowie <strg>+<c> 0b11111111111111111111111111110101 <strg>+<v> in die Register R0, R1,R2, R3, R4 sowie R5. Schauen Sie sich bitte die geladenen Zahlen im Debugger - einerseits im Disassemblerfenster als auch im Registerfenster als Binär-, Dezimal- sowie Hexadezimalzahlen an. Was ist mit den Zahlen und Befehlen passiert? Warum?

Skript:
4.11 Zahlen Notieren

2 Addition von Zahlen

a) Addieren Sie zwei von 0 unterschiedliche, verschiedene positive Werte zusammen. Als Ergebnis dieser Addition sollte im Zielregister der Wert 0 angezeigt werden (x+y = 0).

b) Unter den aus der Aufgabe 1 gewonnenen Erkenntnissen führen Sie die gleiche Operation aus, nur dieses mal notieren Sie bitte die beiden Werte mit so wenig **Zeichen/Stellen** wie möglich. Lassen Sie den Präprozessor die Umrechnungsarbeit erledigen.

c) Modifizieren Sie die Lösung aus der Aufgabe 2a so, dass sie nur noch eine Zahl verwenden. (z + z = 0 wobei z != 0)

Skript:
4.1 Zahlen auf dem Zahlenkreis

3 Flags und bedingte Ausführung

Gegeben Sei ein Programm zu binarisierung von Daten (siehe Struktogramm). Liegt der Datenwert über eine bestimmte Schwelle, so soll dieser in dem Wert 1 umgewandelt werden. Ist der Wert gleich oder kleiner dieser Schwelle, so wird dieser zu 0 geändert. Implementieren Sie dieses Programm in der ersten Version mit bedingten Sprüngen. Bei dem zweiten Lösungsansatz verwenden Sie bitte statt Sprungbefehlen die bedingte Ausführung von Befehlen. Testen sie die beiden Versionen mit mehreren geeigneten Werten für x und y (copy & paste).

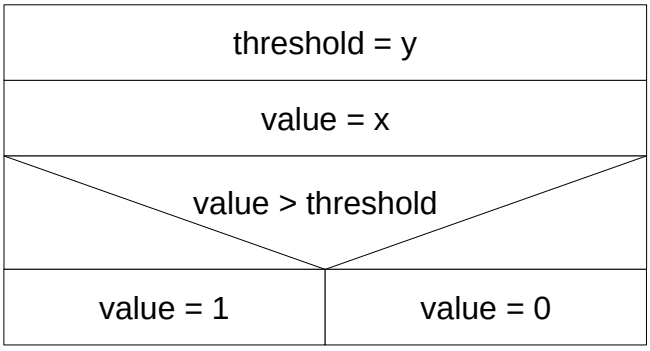

Abbildung 1: Datenbinarisierung

Tipps:
- Die Auswertung kann entweder mit der Vergleichsanweisung ***cmp*** oder mit der Subtraktion ***sub*** realisiert werden.

Skript:
1.6 Bedingte Befehle
2.8.1 Unbedingter Sprung
2.8.2 Bedingte Sprungbefehle

4 Maskenoperationen

Definieren Sie mitder .equ Anweisung Ein-Bit Masken in der „Bitschiftschreibweise“ für folgende Produkte:

Bread, Butter, Cheese,	in Nibble	0
Orange, Banana, Kiwi	in Nibble	1
Water, Milk, Fanta	in Byte	1

Beachte - Masken werden mit Großbuchstaben definiert und mit dem Suffix ***_bm*** (Bitmaske), zur einfacher Unterscheidung von Variablen im Programmcode, markiert.

Erstellen Sie für alle drei Kategorien je eine separate Gesamtmaske (z.b. DRINKS_bm).
Definieren Sie eine Maske für das Tagesfrühstück „Breakfast“ bestehend aus Brot mit Butter und Käse, Glas Milch, sowie einem Kiwi.

Servieren Sie folgende Menüs:
- R0 bekommt das Tagesfrühstück
- R1 bekommt außer dem Tagesfrühstück noch alle Früchte.

- R2 ist laktoseintolerant, seien Sie also so nett und ersetzten Sie Milch durch Fanta.
- R3 ist theoretisch auf Diät, bekommt nur Früchte, dafür aber eine doppelte Portion + Milch. Den zweiten Tablett „Packen“ sie in dem ***High Half Word*** (siehe Tipp).
- R4 isst nur Vegan - entfernen Sie bitte Milch und Käse.

Sind alle glücklich ?

Um diese Aufgabe zu lösen benutzen Sie bitte Masken mit Binäroperatoren Und (&), Oder (|), Nicht(~) sowie Schiebeoperatoren. Bitte definieren sie die Masken, so weit wie möglich, rekursiv. Lassen Sie die Maskenoperationen auch so weit möglich durch den Präprozessor durchführen.

Tipp: Packen bedeutet das zwei oder mehrere Variablen/Konstanten in einer Serialisiert werden (hier vier 8 Bit Variablen in 32 Bit gepackt):
var_a = 0x4f
var_b = 0xaa
var_c = 0x67
var_d = 0x88

var_packed = (var_d«24) | (var_c«16) | (var_b«8) | (var_a) somit ***var_packed = 0x8867aa4f***

Diese Technik wird hauptsächlich bei Parallelisierung der Daten verwendet (SIMD, Grafikbeschleuniger, Audio) um z.b. gleichzeitig mehrere Datenpunkte mit geringeren Genauigkeit (als die des Prozessors) zu berechnen.

Skript:
Vorsicht - 4.9.5 Mathematische Operationen vs. Logische Operationen

4.7 Zahlen Schieben
4.9 Binäre Operationen
4.10 Bitmasken definieren
4.10.1 Masken drehen/negieren
4.10.2 Masken verketten
4.10.3 Bits aus Masken herausnehmen
4.10.4 Masken an anderen Stellen einsetzten