Rekenaarstelsels 245 - Prakties 5
Computer Systems 245 - Practical 5

2015-08-28

Hierdie prakties **sal tel** tot jou punt vir RS245. Probeer om die vrae so goed as moontlik in die beskikbare tyd te antwoord.

Gebruik jou antwoord van vraag 2 van prakties 4 om vraag 1 van hierdie prakties te toets.

1 Gaan toetssom na

Vir 'n studentenommer om geldig te wees moet die toetssom deelbaar wees deur 11. Skryf die `_validate_checksum` saamsteltaalfunksie om te toets of die toetssom wat jy in Vraag 2 van Prakties 4 bereken het geldig is of nie. Stuur 'n integer 1 terug na C indien dit geldig is, of 'n integer 0 as dit ongeldig is (1=waar/geldig, 0=vals/ongeldig). Die C-lêer wat aan jou verskaf is, is opgestel om die LED op die demobord aan te skakel indien die toetssom geldig is.

Gewoonlik sal 'n mens die modulus operator (%) in C gebruik om te bepaal of 'n getal deelbaar is deur 'n ander een, of nie. Die RL78-C2 verwerkerkern wat ons gebruik het nie 'n modulus instruksie nie. Die verwerker kan ook nie deel nie. Om hierdie toets te doen moet ons dus in 'n lus 11 vanaf die toetssom aftrek totdat die antwoord gelyk is aan 11, of kleiner aan 11. Gelyk aan 11 beteken die toetssom is deelbaar deur 11, en iets kleiner as 11 beteken dit is nie deelbaar deur 11 nie.

Sien jy dat 'n mens 'n kennis van die saamsteltaalfunksies van 'n verwerker moet hê, al skryf jy kode daarvoor in C? Indien ons hierdie funksie in C geskryf het sou ons die %-operator gebruik het. Die C-samesteller sou waarskynlik gegaan het en dit vertaal het na 'n soortgelyke lus as wat ons nou implementeer. Die berekening wat jy in C sou implementeer is moontlik effektiewer doenbaar deur nie 'n modulus te gebruik nie. Sonder 'n kennis van wat die verwerker kan, en nie kan doen nie, sal 'n mens dus nie uiters effektiewe kode kan skryf nie. In hierdie vraag het ons ongelukkig nie 'n alternatief anders as om die lus te implementeer nie.

Inhandinging: Handig in voor **23:55 op 17 September** op `learn.sun.ac.za` (na die vakansie).

This practical **will count** towards your mark for CS245. Try to answer the questions the best you can in the available time.

Use your answer from question 2 of practical 4 to test question 1 of this practical.

1 Validate checksum

For a student number to be valid, the checksum must be divisible by 11. Write the `_validate_checksum` assembly function to validate the checksum you have calculated in Question 2 of Practical 4. Return an integer 1 to C if the checksum is valid, or an integer 0 if it is not valid (1=true/valid, 0=false/invalid). The C-file that was given to you is set up to switch the LED on the demo board on if the checksum is valid.

Normally one would use the modulus operator (%) in C to check if one number is divisible by another, or not. The RL78-C2 CPU core that we use does not have a modulus instruction. It also can't divide. To do this test we will thus need to subtract 11 in loop from our checksum until the answer is equal to 11 or smaller that 11. Equal to 11 means the checksum is divisible by 11, whereas an answer smaller than 11 means it is not divisible by 11.

Do you now see that a knowledge of the low level assembly capabilities of a processor is needed, even when writing code in C? Say we were writing this function in C, then we would've use the %-operator. The C compiler likely would have compiled it to a similar loop as the one we are writing now. The calculation you might be doing in C is likely doable more effectively without using a modulus. Thus without knowing the abilities of the processor one can not write extremely effective code. In this question we unfortunately do not have a choice other that implementing a loop.

Hand in time: Hand in before **23:55 on 17 September** on `learn.sun.ac.za` (after the holidays).

2 Borrelsortering

As jy nog nie weet wat borrelsortering is nie, gaan na die volgende twee webblaaie en vind uit wat dit is, hoe dit werk en hoe om dit te implementeer:

2 Bubble Sorting

If you do not know bubble sorting yet, go to the following two web pages to see what it is, how it works, and how to implement it:

`http://en.wikipedia.org/wiki/Bubble_sort#Step-by-step_example`
`http://www.go4expert.com/articles/bubble-sort-algorithm-absolute-beginners-t27883`

In die Prakties5.c-lêer sal jy die volgende lyn kode sien:

In the Prakties5.c file you will see the following line of code:

```
bubble_sort(studentNr);
```

Dit roep die funksie `_bubble_sort` in die bubblesort.asm-lêer. Voltooi die funksie om die string, waarvan die eerste karakter se adres op die stapel oorgestuur is, te sorteer. Dit maak nie saak of jou funksie dit van groot na klein, of klein na groot sorteer nie, aangesien dit slegs van een vergelyk stelling afhang. Jy kan later 'n tweede parameter oorstuur vanaf C wat die rigting bepaal. Onthou dat die string 'n nul karakter aan die einde het, wat nie saam gesorteer moet word nie.

It calls the function `_bubble_sort` in the bubblesort.asm-file. Complete the function to sort the string of which the first character's address has been sent over on the stack. It does not matter if your function sorts ascending or descending, as it depends only on one compare. You can later send a second parameter from C to tell the direction. Remember that the string has a zero character at the end, which should not be sorted.

Inhandinging: Handig in voor **23:55 op 17 September** op `learn.sun.ac.za` (na die vakansie).

Hand in time: Hand in before **23:55 on 17 September** on `learn.sun.ac.za` (after the holidays).