# Rekenaarstelsels 245 - Prakties 3
# Computer Systems 245 - Practical 3

2015-08-14

## 1 Keer 'n sin om

Skryf 'n saamsteltaal program wat 'n string ontvang, dit omdraai en terugstuur. Die string sal 'n 0 greep aan die einde hê (asciiz string).

Jou funksie sal die adres van die eerste karakter in die string by posisie $[sp + 4]$ op die stapel ontvang. Jy kan hierdie adres gebruik om die string in die geheue te vind.

'n Goeie benadering sal wees om eers deur die string te lees om die lengte daarvan te bepaal. Gebruik dan twee registers as tellers: een tel op vanaf 0, en een tel af vanaf die lengte. Beide die `B` en `C` registers kan gebruik word om relatief tot die `HL` register geheue te adresseer. Inkrementeer/dekrementeer die teller-registers, en ruil die data in die geheue om, totdat die teller-registers gelyk of kleiner aan mekaar is.

```
mov  a, [hl+b]
mov  a, [hl+c]
```

Omdat die string as karakters gestoor word, word slegs 'n enkele greep per karakter gebruik. Jy kan dus die byte-manipulasie instruksies gebruik, eerder as die woord-manipulasie instruksies.

Probeer hierdie program klaar te hê teen **vanmiddag 17:00**. Handig slegs jou .asm-lêer in, met jou en jou groepmaat se name bo aan as kommentaarstellings.

## 1 Reverse a sentence

Write an assembly program that receives a string, reverses it, and returns it. The string will be terminated by a 0 byte (asciiz string).

Your function will receive the address of the first character in the string at position $[sp + 4]$ on the stack. You can use this address to find the string in memory.

A good approach would be to first read through the string to determine its length. Then, use two registers as counters: one to count up from 0, and one to count down from the length. Both the `B` and `C` registers can be used to address memory relative to the `HL` register. Increment/decrement the counter registers, and swap the data in memory, until the counter registers are equal or smaller to each other.

Since the string is stored as characters, only a single bytes is used per character. You should therefore use the byte manipulation instructions, rather than the word manipulation instructions.

Try to have this program done by **today at 17:00**. Hand in only you .asm-file with your name and your team member's name at the top as comments.

## 2 Sakrekenaar

Skryf 'n saamsteltaalfunksie wat basiese sakrekenaarfunksies implementeer. Jou funksie sal 3 parameters ontvang: 'n karakter wat die bewerkingsteken aandui, en twee heelgetalle waarop die bewerking uitgevoer moet word.

Die funksie moet die volgende instruksies kan interpreteer:

| Character | Hex | Operand/Bewerking |
|---|---|---|
| '+' | 0x2B | $num1 + num2$ |
| '-' | 0x2D | $num1 - num2$ |
| '*' | 0x2A | $num1 \times num2$ |

## 2 Calculator

Write an assembly function that implements basic calculator functions. Your program will receive three parameters: a character that indicates the operator, and two integers on which the operation must be performed.

The calculator needs to interpret the following instructions:

Om te vermenigvuldig kan jy die `mulu` instruksie gebruik, wat `a` en `x` maal, en die resultaat in `ax` plaas.

Handing slegs jou .asm-lêer in voor **Donderdag 20 Augustus teen 23:55**. Onthou julle groeplede se name bo aan die lêer.

To multiply, you can use the `mulu` instruction, which multiplies `a` with `x`, and places the result in `ax`.

Hand in only your .asm-file by **Thursday 20 August at 23:55**. Remember your team members' names at the top of the file.

```
extern void calculate(char operand, int num1, int num2);

int main(void)
{
  int answer = 0;
  answer = calculate('+',5,16);
  answer = calculate('-',42,13);
  answer = calculate('*',2,7);
}
```

# 3 Wenke

## 3.1 Vergelyking en aftakking

Die `cmp`-instruksie word gebruik om twee waardes met mekaar te vergelyk. Wanneer die waardes vergelyk word, word hulle van mekaar afgetrek, maar die resultaat word net gebruik om die vlaggies te stel. Die vlaggies word dan deur een van die aftak*(branch)*-instruksies gebruik om te bepaal of die program moet aftak of nie. Dit kan gebruik word om vloei van die program te beheer, net soos *if*, *for* en *while* stellings. Die kode hieronder sal die waarde in `ax` verdubbel as dit groter as 5 is.

# 3 Hints

## 3.1 Comparing and Branching

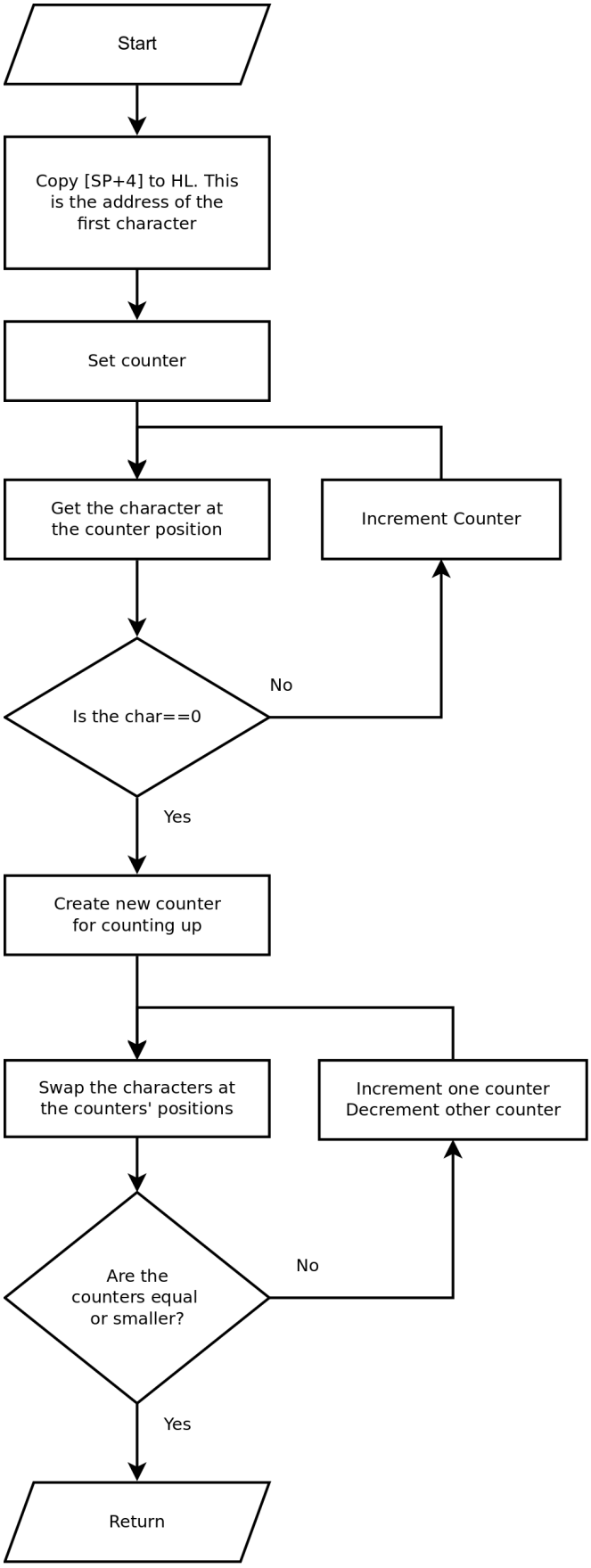
The `cmp` instruction is used to compare two values with each other. When the values are compared, they are subtracted, but only the flags are set. These flags are then used by one of the branch instructions to determine if the program should branch or not. This can be used to control program flow, just like if, for and while statements. The following piece of code will double the value of `ax` if it is larger than 5.

```
  cmpw ax,#5
  bh $higher
  br $lower

higher:
  addw ax,ax

lower:
  ret
```

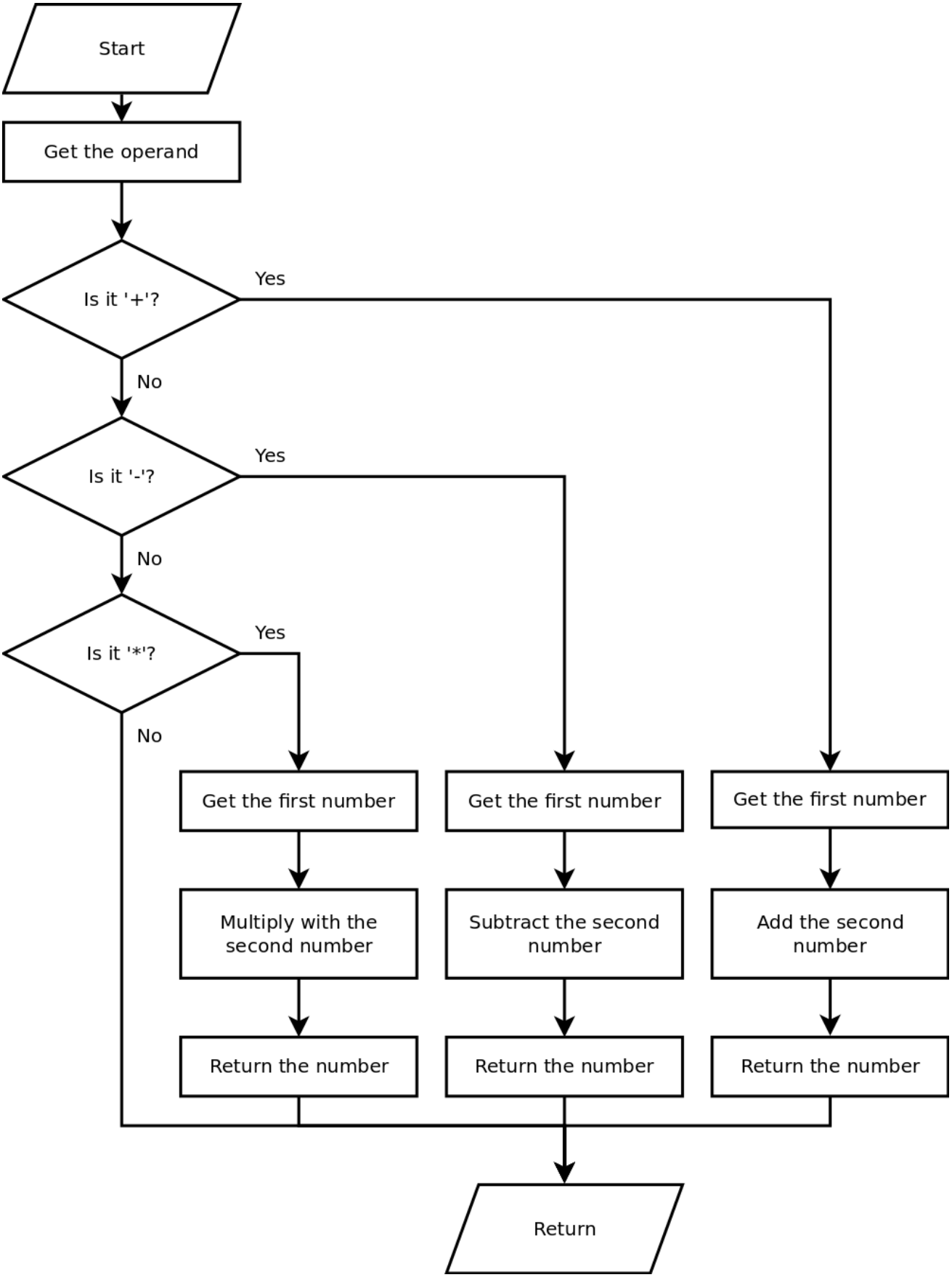
Vergelyk en aftak kan ook gebruik word om 'n lus te form, soos hierdie voorbeeld wat tot 10 tel.

Compare and branch can also be used to create loops, such as this example which counts to 10.

```
  mov a,#0

loop_start:
  inc a
  cmp a,#10
  bz $loop_start
```




Vloeidiagram van die string-omkeer algoritme
Flow diagram for the string reverse algorithm




Vloeidiagram vir die sakrekenaar
Flow diagram for the calculator