3 . Hausübung

Gruppe: - Ulf Gebhardt (rbg: hu56nifa)
- Michael Scholz (rbg: mi48azih)

Aufgabe 1: Unterprogramm zum Sortieren:

a)

```
.data
       # zu sortierende List einkommentieren
       # list: .long 10,9,8,7,6,5,4,3,2,1
         list: .long 16,5,18,12,11,66,62,1,9,33

       # Länge der Liste:
       length: .long 10

intout:
.string "Sortierte Liste: %d,%d,%d,%d,%d,%d,%d,%d,%d,%d\n"

.text

.globl main

main:


       leal list, %esi      # Laden der Adresse mach %esi
       leal length, %edi    # Laden der Adresse nach %edi


       call .bubble_sort    # Aufruf von Unterprogramm


       jmp .ende



.bubble_sort:

       # Basepointer muss nicht gesichert werden. Das übernimmt der call-Befehl!


       movl (%edi), %ecx    # lese Wert von length


       dec %ecx             # length - 1
       xor %edi, %edi       # %edi = 0
       movl %esi, %edi

       ###########################################################################

       .schleife_aussen:



              pushl %ecx    # Speicher den Wert von %ecx auf Stack

              ###################################################

              .schleife_innen:
```

```
                        movl 0(%edi), %eax          # %eax = Wert an Adresse %edi (j)
                        movl 4(%edi), %ebx          # %ebx = Wert an Adresse %edi + 4 (j + 1)
                        cmp %eax, %ebx              # vergleiche nun j und j+1 und springe zu
                                                    # den passenden Labels
                        jb .true

                        jmp .false

                        .true:
                               # Registertausch:

                               movl %ebx, 0(%edi) # zurück ins Datenfeld schreiben
                               movl %eax, 4(%edi) # zurück ins Datenfeld schreiben

                        .false:

                               # Weiter im Code

                        addl $4, %edi        # Adresse in %edi um 4 erhöhen, um nun das
                                             # nächste Paar im Array zu vergleichen


                loop .schleife_innen

                ############################################################


        popl %ecx     # ursprünglichen Wert von %ecx vom Stack holen

        movl %esi, %edi

        loop .schleife_aussen
        ###############################################################################



ret           # Unterprogramm verlassen

.ende:



#### Ausgabe ####

movl length, %esi           # Kopiere Arraygröße nach %esi(= k)

.printElement:

        cmpl $0 , %esi      # Vergleiche %esi = 0 ?
        je .callPrintf      # Wenn ja ,springe zu callPrintf
        dec %esi            # Veringere %esi um 1

        movl list(,%esi,4) , %ecx  # Kopiere a[k] nach %ecx
        pushl %ecx                 # pushl auf Stack
        jmp .printElement

.callPrintf:
pushl $intout
call  printf

# Exit
movl $1, %eax
int $0x80
```

b)

Es steht nur des Basepointer, welcher von dem call-Befehl geschrieben wird, auf dem Stack. Die Variablen werden über die Register ans Unterprogramm übergeben und befinden sich deshalb nicht auf dem Stack

Aufgabe 2: Ägyptisches Potenzieren:

a)

Exponent	Basis	relevant ?
7	5	ja
3	25 mod 77 = 25	ja
1	625 mod 77 = 9	ja

Ergebnis: 5 · 25 · 9 mod 77 = 1125 mod 77 = 47

b)

```
.data

b: .long 5
e: .long 7
m: .long 77


intout:
.string "Ergebnis: %d\n"

.text

.globl main

main:

       movl b, %eax
       movl e, %ebx
       movl m, %ecx


       call .agy_pot

       jmp .end

#####################################
# %eax = b
# %ebx = e
# %ecx = m

       .agy_pot:

              cmp $0, %ebx               # exponent = 0?
              je .agy_pot_ancor          # jump to ancor

              movl %ebx, %edx            # copy e to %edx
              and $0b000001, %edx        # mask last bit of %edx
              cmp $0, %edx               # %edx uneven?
              je .agy_pot_irrelevant     # even = irrelevant
```

```
        jmp .agy_pot_relevant           # uneven = relevant

        .agy_pot_relevant:
                pushl %eax              # save basis
                pushl %eax              # save basis again

                movl %ebx, %eax         # %eax = exponent
                movl $2, %ebx           # %ebx = 2
                cdq
                idivl %ebx              # exponent / 2
                movl %eax, %ebx         # %ebx = exponent / 2

                popl %eax               # restore basis
                imull %eax, %eax        # basis^2

                cdq                     # Basis^2 MOD m
                idivl %ecx
                movl %edx, %eax

                call .agy_pot           # call agy_pot for the rest

                popl %edx               # get relevant basis

                imull %edx, %eax        # relvant basis * result from agy_pot

                cdq
                idivl %ecx              # (relvant basis * result from agy_pot)
                                        # mod m
                movl %edx, %eax

                ret

        .agy_pot_irrelevant:

                pushl %eax              # save basis

                movl %ebx, %eax         # %eax = exponent
                movl $2, %ebx           # %ebx = 2
                cdq
                idivl %ebx              # exponent / 2
                movl %eax, %ebx         # %ebx = exponent / 2

                popl %eax               # restore basis
                imull %eax, %eax        # basis^2

                cdq                     # Basis^2 MOD m
                idivl %ecx
                movl %edx, %eax

                call .agy_pot           # call agy_pot for the rest
                ret

        .agy_pot_ancor:
                movl $1, %eax           # result = 1
                ret


##AUSGABE#####################################################################
.end:

# Wert im %eax ausgeben
pushl %eax
pushl $intout
call printf


# Exit
movl $1, %eax
int $0x80
```