## 3.6.1

* The line is a comment and won’t be compiled
* The assembly file contains human readable code, whereas the object file contains machine readable code
* The linker combines object files that have dependencies to one another
* Echo $?
* Movl $1 %eax moves the value one into the %eax register, movl 1 %eax moves the content at the address one into %eax
* %eax
* An index is used to access data in an array
* The index is zero because address\_of\_array + index retrieves the first item of the array
* 3643 + 4 \* 13 = 3695
* Eax, ebx, ecx, edx, edi, esi
* Movl moves a whole word, movb only moves a byte
* Flow control are structures that decide the path the program takes, e.g. if-construct
* A conditional jump only jumps to the specified routine if the comparison of two values holds true
* You have to plan the registers to use, the algorithm and the amount of storage needed.
* Address modes:
  + Movl $1 %eax: Immediate mode
  + Movl $0 %ebx: Immediate mode
  + movl $0, %edi: Immediate mode
  + movl data\_items(,%edi,4), %eax: indexed addressing mode
  + movl %eax, %ebx: indirect addressing mode

## 3.6.2

Modify first program

* .section .data
* .section .text
* .globl \_start
* **\_start:**
* movl $1, %eax *# this is the linux kernel command*
* *# number (system call) for exiting*
* *# a program*
* movl $3, %ebx *# this is the status number we will*
* *# return to the operating system.*
* *# Change this around and it will*
* *# return different things to*
* *# echo $?*
* int $0x80 *# this wakes up the kernel to run*
* *# the exit command*

Find minimum

.section .data

**data\_items:** *#These are the data items*

.long 3,67,34,222,45,75,54,34,44,33,22,11,66,0

.section .text

.globl \_start

**\_start:**

movl $0, %edi *# move 0 into the index register*

movl data\_items(,%edi,4), %eax *# load the first byte of data*

movl %eax, %ebx *# since this is the first item, %eax is*

*# the biggest*

**start\_loop:** *# start loop*

cmpl $0, %eax *# check to see if we've hit the end*

je loop\_exit

incl %edi *# load next value*

movl data\_items(,%edi,4), %eax

cmpl %ebx, %eax *# compare values*

jge start\_loop *# jump to loop beginning if the new*

*# one isn't bigger*

movl %eax, %ebx *# move the value as the largest*

jmp start\_loop *# jump to loop beginning*

**loop\_exit:**

movl $1, %eax *#1 is the exit() syscall*

int $0x80

Use 255 as stop

.section .data

**data\_items:** *#These are the data items*

.long 3,67,34,222,45,75,54,34,44,33,22,11,66,255

.section .text

.globl \_start

**\_start:**

movl $0, %edi *# move 0 into the index register*

movl data\_items(,%edi,4), %eax *# load the first byte of data*

movl %eax, %ebx *# since this is the first item, %eax is*

*# the biggest*

**start\_loop:** *# start loop*

cmpl $255, %eax *# check to see if we've hit the end*

je loop\_exit

incl %edi *# load next value*

movl data\_items(,%edi,4), %eax

cmpl %ebx, %eax *# compare values*

jle start\_loop *# jump to loop beginning if the new*

*# one isn't bigger*

movl %eax, %ebx *# move the value as the largest*

jmp start\_loop *# jump to loop beginning*

**loop\_exit:**

movl $1, %eax *#1 is the exit() syscall*

int $0x80

//Use ending address missing

Use length count

.section .data

**data\_items:** *#These are the data items*

.long 3,67,34,222,45,75,54,34,44,33,22,11,66,0

.section .text

.globl \_start

**\_start:**

movl $0, %edi *# move 0 into the index register*

movl data\_items(,%edi,4), %eax *# load the first byte of data*

movl %eax, %ebx *# since this is the first item, %eax is*

*# the biggest*

**start\_loop:** *# start loop*

cmpl $6, %edx *# check to see if we've hit the end*

je loop\_exit

incl %edi *# load next value*

movl data\_items(,%edi,4), %eax

cmpl %ebx, %eax *# compare values*

jge start\_loop *# jump to loop beginning if the new*

*# one isn't bigger*

movl %eax, %ebx *# move the value as the largest*

jmp start\_loop *# jump to loop beginning*

**loop\_exit:**

movl $1, %eax *#1 is the exit() syscall*

int $0x80

* Movl \_start, %eax would move the address of the first instruction into the eax register, movl $\_start, %eax would move the string ‘\_start’ into eax

## 3.6.3

* Error message: Speicherzugriffsfehler, ‚echo $?‘ shows 136. Probably because the system recognises changes in the registers, but without a signal what to do with them, marks them as an error
* Length count is probably best, since sometimes all values are needed (no value free as marker) and the ending address is not always known. Same reasoning if the list was sorted.