ASM LABs

sys_write()

```
SECTION .data
msg     db      'Hello World!', 0Ah; assign msg variable with your message string

SECTION .text
global  _start

_start:

    mov     edx, 13     ; number of bytes to write - one for each letter
                        ;plus 0Ah (line feed character)
    mov     ecx, msg    ; move the memory address of our message string into ecx
    mov     ebx, 1      ; write to the STDOUT file
    mov     eax, 4      ; invoke SYS_WRITE (kernel opcode 4)
    int     80h
```

```
~$ nasm -f elf helloworld.asm
~$ ld -m elf_i386 helloworld.o -o helloworld
~$ ./helloworld

Hello World!
Segmentation fault
```

sys_exit()

```
SECTION .data
msg     db      'Hello World!', 0Ah

SECTION .text
global  _start

_start:

    mov     edx, 13
    mov     ecx, msg
    mov     ebx, 1
    mov     eax, 4
    int     80h

    mov     ebx, 0      ; return 0 status on exit - 'No Errors'
    mov     eax, 1      ; invoke SYS_EXIT (kernel opcode 1)
    int     80h
```

Calculate string length

```
SECTION .data
msg     db      'Hello, brave new world!', 0Ah ; we can modify this now without having to update
                                               ; anywhere else in the program

SECTION .text
global  _start

_start:

    mov     ebx, msg        ; move the address of our message string into EBX
    mov     eax, ebx        ; move the address in EBX into EAX as well (Both now point to the same segment in memory)

nextchar:
    cmp     byte [eax], 0   ; compare the byte pointed to by EAX at this address against zero
    jz      finished        ; jump to the point in the code labeled 'finished'
    inc     eax             ; increment the address in EAX by one byte (if the zero flagged has NOT been set)
    jmp     nextchar        ; jump to the point in the code labeled 'nextchar'

finished:
    sub     eax, ebx         ; subtract the address in EBX from the address in EAX
                             ; remember both registers started pointing to the same address (see line 15)
                             ; but EAX has been incremented one byte for each character in the message string
                             ; when you subtract one memory address from another of the same type
                             ; the result is number of segments between them-in this case the number of bytes

    mov     edx, eax       ; EAX now equals the number of bytes in our string
    mov     ecx, msg        ; the rest of the code should be familiar now
    mov     ebx, 1
    mov     eax, 4
    int     80h

    mov     ebx, 0
    mov     eax, 1
    int     80h
```

strlen subroutine

```
SECTION .data
msg     db      'Hello, brave new world!', 0Ah

SECTION .text
global  _start

_start:

    mov     eax, msg        ; move the address of our message string into EAX
    call    strlen          ; call our function to calculate the length of the string

    mov     edx, eax        ; our function leaves the result in EAX
    mov     ecx, msg        ; this is all the same as before
    mov     ebx, 1
    mov     eax, 4
    int     80h

    mov     ebx, 0
    mov     eax, 1
    int     80h

strlen:                     ; this is our first function declaration
    push    ebx             ; push the value in EBX onto the stack to preserve it while we use EBX in this
function
    mov     ebx, eax        ; move the address in EAX into EBX (Both point to the same segment in memory)

nextchar:                   ; this is the same as lesson3
    cmp     byte [eax], 0
    jz      finished
    inc     eax
    jmp     nextchar

finished:
    sub     eax, ebx
    pop     ebx             ; pop the value on the stack back into EBX
    ret                     ; return to where the function was called
```

External include files

functions.asm

```
slen:
    push    ebx
    mov     ebx, eax

nextchar:
    cmp     byte [eax], 0
    jz      finished
    inc     eax
    jmp     nextchar

finished:
    sub     eax, ebx
    pop     ebx
    ret

; void sprint(String message)
; String printing function
sprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    slen
```

```
    mov     edx, eax
    pop     eax

    mov     ecx, eax
    mov     ebx, 1
    mov     eax, 4
    int     80h

    pop     ebx
    pop     ecx
    pop     edx
    ret


; void exit()
; Exit program and restore resources
quit:
    mov     ebx, 0
    mov     eax, 1
    int     80h
    ret
```

External include files (cont.)

```
%include        'functions.asm'                         ; include our external file

SECTION .data
msg1    db      'Hello, brave new world!', 0Ah           ; our first message string
msg2    db      'This is how we recycle in NASM.', 0Ah   ; our second message string

SECTION .text
global  _start

_start:

    mov     eax, msg1       ; move the address of our first message string into EAX
    call    sprint          ; call our string printing function

    mov     eax, msg2       ; move the address of our second message string into EAX
    call    sprint          ; call our string printing function

    call    quit            ; call our quit function
```

Passing arguments

- When run a program, any passed arguments are loaded onto the stack in reverse order.
- The last two stack items for a NASM compiled program are always the name of the program and the number of passed arguments.

```
%include        'functions.asm'

SECTION .text
global  _start

_start:

    pop     ecx             ; first value on the stack is the number of arguments

nextArg:
    cmp     ecx, 0h         ; check to see if we have any arguments left
    jz      noMoreArgs      ; if zero flag is set jump to noMoreArgs label
    pop     eax             ; pop the next argument off the stack
    call    sprintLF        ; call our print with linefeed function
    dec     ecx             ; decrease ecx (number of arguments left) by 1
    jmp     nextArg         ; jump to nextArg label

noMoreArgs:
    call    quit
```

sys_read()

```
%include        'functions.asm'

SECTION .data
msg1        db      'Please enter your name: ', 0h    ; message string asking user for input
msg2        db      'Hello, ', 0h   ; message string to use after user has entered their name

SECTION .bss
sinput:     resb    255    ; reserve a 255 byte space in memory for the users input string

SECTION .text
global  _start

_start:

    mov     eax, msg1
    call    sprint

    mov     edx, 255        ; number of bytes to read
    mov     ecx, sinput     ; reserved space to store our input (known as a buffer)
    mov     ebx, 0          ; write to the STDIN file
    mov     eax, 3          ; invoke SYS_READ (kernel opcode 3)
    int     80h

    mov     eax, msg2
    call    sprint

    mov     eax, sinput     ; move our buffer into eax (Note: input contains a linefeed)
    call    sprint          ; call our print function

    call    quit
```

Exercises

1. Write a program that print to display numbers from 0 to 9

2. Write a program that print to display numbers from 0 to 10

Execute Command

The EXEC family of functions replace the currently running process with a new process, that executes the command specified when calling it.

Naming convention

The naming convention used for this family of functions is **exec** (execute) followed by one or more of the following letters.

•E - An array of pointers to environment variables is explicitly passed to the new process image.

•L - Command-line arguments are passed individually to the function.

•P - Uses the PATH environment variable to find the file named in the path argument to be executed.

•V - Command-line arguments are passed to the function as an array of pointers.

sys_execve

```
%include        'functions.asm'

SECTION .data
command         db      '/bin/echo', 0h      ; command to execute
arg1            db      'Hello World!', 0h
arguments       dd      command
                dd      arg1                 ; arguments to pass to commandline (in this case just one)
                dd      0h                   ; end the struct
environment     dd      0h     ; arguments to pass as environment variables (inthis case none) end the struct

SECTION .text
global  _start

_start:

    mov     edx, environment     ; address of environment variables
    mov     ecx, arguments       ; address of the arguments to pass to the commandline
    mov     ebx, command         ; address of the file to execute
    mov     eax, 11              ; invoke SYS_EXECVE (kernel opcode 11)
    int     80h

    call    quit                 ; call our quit function
```

sys_fork

- Use sys_fork to create a new process that duplicates the current process

```
%include        'functions.asm'

SECTION .data
childMsg        db      'This is the child process', 0h     ; a message string
parentMsg       db      'This is the parent process', 0h    ; a message string

SECTION .text
global  _start

_start:

    mov     eax, 2              ; invoke SYS_FORK (kernel opcode 2)
    int     80h

    cmp     eax, 0              ; if eax is zero we are in the child process
    jz      child               ; jump if eax is zero to child label

parent:
    mov     eax, parentMsg      ; inside our parent process move parentMsg into eax
    call    sprintLF            ; call our string printing with linefeed function

    call    quit                ; quit the parent process

child:
    mov     eax, childMsg       ; inside our child process move childMsg into eax
    call    sprintLF            ; call our string printing with linefeed function

    call    quit                ; quit the child process
```

Exercises

- Write code example that call other common syscall, such as:

  creat, sync, open, close, wait, kill