NASM Assembly Language Tutorials - asmtutor.com

```
; Hello World Program - asmtutor.com
; Compile with: nasm -f elf helloworld.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld.o -o helloworld
; Run with: ./helloworld

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
```

```
; Hello World Program - asmtutor.com
; Compile with: nasm -f elf helloworld.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld.o -o helloworld
; Run with: ./helloworld

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
```

```
; Hello World Program (Calculating string length)
; Compile with: nasm -f elf helloworld-len.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-len.o -o helloworld-len
; Run with: ./helloworld-len

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 (Zero is an end
        of string delimiter)
    jz      finished        ; jump (if the zero flagged has been set) 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 'loop'

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
```

```
29
30      mov     edx, eax        ; EAX now equals the number of bytes in our string
31      mov     ecx, msg        ; the rest of the code should be familiar now
32      mov     ebx, 1
33      mov     eax, 4
34      int     80h
35
36      mov     ebx, 0
37      mov     eax, 1
38      int     80h
```

```
1  ; Hello World Program (Subroutines)
2  ; Compile with: nasm -f elf helloworld-len.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-len.o -o helloworld-len
4  ; Run with: ./helloworld-len
5
6  SECTION .data
7  msg     db      'Hello, brave new world!', 0AH
8
9  SECTION .text
10 global  _start
11
12 _start:
13
14      mov     eax, msg        ; move the address of our message string into EAX
15      call    strlen          ; call our function to calculate the length of the string
16
17      mov     edx, eax        ; our function leaves the result in EAX
18      mov     ecx, msg        ; this is all the same as before
19      mov     ebx, 1
20      mov     eax, 4
21      int     80h
22
23      mov     ebx, 0
24      mov     eax, 1
25      int     80h
26
27 strlen:                      ; this is our first function declaration
28      push    ebx             ; push the value in EBX onto the stack to preserve it while we use EBX in this
            function
29      mov     ebx, eax        ; move the address in EAX into EBX (Both point to the same segment in memory)
30
31 nextchar:                    ; this is the same as lesson3
32      cmp     byte [eax], 0
33      jz      finished
34      inc     eax
35      jmp     nextchar
36
37 finished:
38      sub     eax, ebx
39      pop     ebx             ; pop the value on the stack back into EBX
40      ret                     ; return to where the function was called
```

```
1  ;-----------------------------------------
2  ; int slen(String message)
3  ; String length calculation function
4  slen:
5      push    ebx
6      mov     ebx, eax
7
8  nextchar:
9      cmp     byte [eax], 0
10     jz      finished
11     inc     eax
12     jmp     nextchar
13
14 finished:
15     sub     eax, ebx
16     pop     ebx
17     ret
18
19
20 ;-----------------------------------------
21 ; void sprint(String message)
22 ; String printing function
23 sprint:
24     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
```

```
; Hello World Program (External file include)
; Compile with: nasm -f elf helloworld-inc.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-inc.o -o helloworld-inc
; Run with: ./helloworld-inc

%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
```

```
;------------------------------------------
; int slen(String message)
; String length calculation function
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
```

```
; Hello World Program (NULL terminating bytes)
; Compile with: nasm -f elf helloworld-inc.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-inc.o -o helloworld-inc
; Run with: ./helloworld-inc

%include        'functions.asm'

SECTION .data
msg1    db      'Hello, brave new world!', 0Ah, 0h          ; NOTE the null terminating byte
msg2    db      'This is how we recycle in NASM.', 0Ah, 0h  ; NOTE the null terminating byte

SECTION .text
global  _start

_start:

    mov     eax, msg1
    call    sprint

    mov     eax, msg2
    call    sprint

    call    quit
```

```
;------------------------------------------
; int slen(String message)
; String length calculation function
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 sprintLF(String message)
; String printing with line feed function
sprintLF:
    call    sprint

    push    eax         ; push eax onto the stack to preserve it while we use the eax register in this function
    mov     eax, 0Ah    ; move 0AH into eax - 0AH is the ascii character for a linefeed
    push    eax         ; push the linefeed onto the stack so we can get the address
    mov     eax, esp    ; move the address of the current stack pointer into eax for sprint
    call    sprint      ; call our sprint function
    pop     eax         ; remove our linefeed character from the stack
    pop     eax         ; restore the original value of eax before our function was called
    ret                 ; return to our program


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

```
; Hello World Program (Print with line feed)
; Compile with: nasm -f elf helloworld-lf.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-lf.o -o helloworld-lf
; Run with: ./helloworld-lf

%include        'functions.asm'

SECTION .data
msg1    db      'Hello, brave new world!', 0h          ; NOTE we have removed the line feed character 0AH
msg2    db      'This is how we recycle in NASM.', 0h  ; NOTE we have removed the line feed character 0AH

SECTION .text
global  _start

_start:

    mov     eax, msg1
    call    sprintLF    ; NOTE we are calling our new print with linefeed function

    mov     eax, msg2
    call    sprintLF    ; NOTE we are calling our new print with linefeed function

    call    quit
```

```
;-----------------------------------------
; int slen(String message)
; String length calculation function
slen:
    push    ebx
    mov     ebx, eax

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

finished:
    sub     eax, ebx
    pop     ebx
    ret
```

```
18
19
20 ;-----------------------------------------
21 ; void sprint(String message)
22 ; String printing function
23 sprint:
24     push    edx
25     push    ecx
26     push    ebx
27     push    eax
28     call    slen
29
30     mov     edx, eax
31     pop     eax
32
33     mov     ecx, eax
34     mov     ebx, 1
35     mov     eax, 4
36     int     80h
37
38     pop     ebx
39     pop     ecx
40     pop     edx
41     ret
42
43
44 ;-----------------------------------------
45 ; void sprintLF(String message)
46 ; String printing with line feed function
47 sprintLF:
48     call    sprint
49
50     push    eax
51     mov     eax, 0AH
52     push    eax
53     mov     eax, esp
54     call    sprint
55     pop     eax
56     pop     eax
57     ret
58
59
60 ;-----------------------------------------
61 ; void exit()
62 ; Exit program and restore resources
63 quit:
64     mov     ebx, 0
65     mov     eax, 1
66     int     80h
67     ret
```

```
1  ; Hello World Program (Passing arguments from the command line)
2  ; Compile with: nasm -f elf helloworld-args.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-args.o -o helloworld-args
4  ; Run with: ./helloworld-args
5
6  %include        'functions.asm'
7
8  SECTION .text
9  global  _start
10
11 _start:
12
13     pop     ecx             ; first value on the stack is the number of arguments
14
15 nextArg:
16     cmp     ecx, 0H         ; check to see if we have any arguments left
17     jz      noMoreArgs      ; if zero flag is set jump to noMoreArgs label (by jumping over the end of the loop
       )
18     pop     eax             ; pop the next argument off the stack
19     call    sprintLF        ; call our print with linefeed function
20     dec     ecx             ; decrease edi (number of arguments left) by 1
21     jmp     nextArg         ; jump to nextArg label
22
23 noMoreArgs:
24     call    quit
```

```
1 ;-----------------------------------------
2 ; int slen(String message)
```

```
3  ; String length calculation function
4  slen:
5      push    ebx
6      mov     ebx, eax
7
8  nextchar:
9      cmp     byte [eax], 0
10     jz      finished
11     inc     eax
12     jmp     nextchar
13
14 finished:
15     sub     eax, ebx
16     pop     ebx
17     ret
18
19
20 ;-----------------------------------------
21 ; void sprint(String message)
22 ; String printing function
23 sprint:
24     push    edx
25     push    ecx
26     push    ebx
27     push    eax
28     call    slen
29
30     mov     edx, eax
31     pop     eax
32
33     mov     ecx, eax
34     mov     ebx, 1
35     mov     eax, 4
36     int     80h
37
38     pop     ebx
39     pop     ecx
40     pop     edx
41     ret
42
43
44 ;-----------------------------------------
45 ; void sprintLF(String message)
46 ; String printing with line feed function
47 sprintLF:
48     call    sprint
49
50     push    eax
51     mov     eax, 0AH
52     push    eax
53     mov     eax, esp
54     call    sprint
55     pop     eax
56     pop     eax
57     ret
58
59
60 ;-----------------------------------------
61 ; void exit()
62 ; Exit program and restore resources
63 quit:
64     mov     ebx, 0
65     mov     eax, 1
66     int     80h
67     ret
```

```
1  ; Hello World Program (Getting input)
2  ; Compile with: nasm -f elf helloworld-input.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-input.o -o helloworld-input
4  ; Run with: ./helloworld-input
5
6  %include        'functions.asm'
7
8  SECTION .data
9  msg1        db      'Please enter your name: ', 0h      ; message string asking user for input
10 msg2        db      'Hello, ', 0h                       ; message string to use after user has entered their
       name
11
12 SECTION .bss
13 sinput:     resb    255                                 ; reserve a 255 byte space in memory for the users
       input string
```

```
14
15 SECTION .text
16 global  _start
17
18 _start:
19
20     mov     eax, msg1
21     call    sprint
22
23     mov     edx, 255        ; number of bytes to read
24     mov     ecx, sinput     ; reserved space to store our input (known as a buffer)
25     mov     ebx, 0          ; write to the STDIN file
26     mov     eax, 3          ; invoke SYS_READ (kernel opcode 3)
27     int     80h
28
29     mov     eax, msg2
30     call    sprint
31
32     mov     eax, sinput     ; move our buffer into eax (Note: input contains a linefeed)
33     call    sprint          ; call our print function
34
35     call    quit
```

```
1  ;------------------------------------------
2  ; int slen(String message)
3  ; String length calculation function
4  slen:
5      push    ebx
6      mov     ebx, eax
7
8  nextchar:
9      cmp     byte [eax], 0
10     jz      finished
11     inc     eax
12     jmp     nextchar
13
14 finished:
15     sub     eax, ebx
16     pop     ebx
17     ret
18
19
20 ;------------------------------------------
21 ; void sprint(String message)
22 ; String printing function
23 sprint:
24     push    edx
25     push    ecx
26     push    ebx
27     push    eax
28     call    slen
29
30     mov     edx, eax
31     pop     eax
32
33     mov     ecx, eax
34     mov     ebx, 1
35     mov     eax, 4
36     int     80h
37
38     pop     ebx
39     pop     ecx
40     pop     edx
41     ret
42
43
44 ;------------------------------------------
45 ; void sprintLF(String message)
46 ; String printing with line feed function
47 sprintLF:
48     call    sprint
49
50     push    eax
51     mov     eax, 0AH
52     push    eax
53     mov     eax, esp
54     call    sprint
55     pop     eax
56     pop     eax
```

```
    ret


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

```
; Hello World Program (Count to 10)
; Compile with: nasm -f elf helloworld-10.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-10.o -o helloworld-10
; Run with: ./helloworld-10

%include        'functions.asm'

SECTION .text
global  _start

_start:

    mov     ecx, 0          ; ecx is initalised to zero.

nextNumber:
    inc     ecx             ; increment ecx

    mov     eax, ecx        ; move the address of our integer into eax
    add     eax, 48         ; add 48 to our number to convert from integer to ascii for printing
    push    eax             ; push eax to the stack
    mov     eax, esp        ; get the address of the character on the stack
    call    sprintLF        ; call our print function

    pop     eax             ; clean up the stack so we don't have unneeded bytes taking up space
    cmp     ecx, 10         ; have we reached 10 yet? compare our counter with decimal 10
    jne     nextNumber      ; jump if not equal and keep counting

    call    quit
```

```
;-----------------------------------------
; void iprint(Integer number)
; Integer printing function (itoa)
iprint:
    push    eax             ; preserve eax on the stack to be restored after function runs
    push    ecx             ; preserve ecx on the stack to be restored after function runs
    push    edx             ; preserve edx on the stack to be restored after function runs
    push    esi             ; preserve esi on the stack to be restored after function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    idiv    esi             ; divide eax by esi
    add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
          instruction
    push    edx             ; push edx (string representation of an intger) onto the stack
    cmp     eax, 0          ; can the integer be divided anymore?
    jnz     divideLoop      ; jump if not zero to the label divideLoop

printLoop:
    dec     ecx             ; count down each byte that we put on the stack
    mov     eax, esp        ; mov the stack pointer into eax for printing
    call    sprint          ; call our string print function
    pop     eax             ; remove last character from the stack to move esp forward
    cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
    jnz     printLoop       ; jump is not zero to the label printLoop

    pop     esi             ; restore esi from the value we pushed onto the stack at the start
    pop     edx             ; restore edx from the value we pushed onto the stack at the start
    pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
    pop     eax             ; restore eax from the value we pushed onto the stack at the start
    ret


;-----------------------------------------
; void iprintLF(Integer number)
```

```
38 ; Integer printing function with linefeed (itoa)
39 iprintLF:
40     call    iprint          ; call our integer printing function
41
42     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
43     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
44     push    eax             ; push the linefeed onto the stack so we can get the address
45     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
46     call    sprint          ; call our sprint function
47     pop     eax             ; remove our linefeed character from the stack
48     pop     eax             ; restore the original value of eax before our function was called
49     ret
50
51
52 ;-----------------------------------------
53 ; int slen(String message)
54 ; String length calculation function
55 slen:
56     push    ebx
57     mov     ebx, eax
58
59 nextchar:
60     cmp     byte [eax], 0
61     jz      finished
62     inc     eax
63     jmp     nextchar
64
65 finished:
66     sub     eax, ebx
67     pop     ebx
68     ret
69
70
71 ;-----------------------------------------
72 ; void sprint(String message)
73 ; String printing function
74 sprint:
75     push    edx
76     push    ecx
77     push    ebx
78     push    eax
79     call    slen
80
81     mov     edx, eax
82     pop     eax
83
84     mov     ecx, eax
85     mov     ebx, 1
86     mov     eax, 4
87     int     80h
88
89     pop     ebx
90     pop     ecx
91     pop     edx
92     ret
93
94
95 ;-----------------------------------------
96 ; void sprintLF(String message)
97 ; String printing with line feed function
98 sprintLF:
99     call    sprint
100
101    push    eax
102    mov     eax, 0AH
103    push    eax
104    mov     eax, esp
105    call    sprint
106    pop     eax
107    pop     eax
108    ret
109
110
111 ;-----------------------------------------
112 ; void exit()
113 ; Exit program and restore resources
114 quit:
115    mov     ebx, 0
116    mov     eax, 1
117    int     80h
```

```
118     ret
```

```
1  ; Hello World Program (Count to 10 itoa)
2  ; Compile with: nasm -f elf helloworld-itoa.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 helloworld-itoa.o -o helloworld-itoa
4  ; Run with: ./helloworld-itoa
5
6  %include        'functions.asm'
7
8  SECTION .text
9  global  _start
10
11 _start:
12
13     mov     ecx, 0
14
15 nextNumber:
16     inc     ecx
17     mov     eax, ecx
18     call    iprintLF        ; NOTE call our new integer printing function (itoa)
19     cmp     ecx, 10
20     jne     nextNumber
21
22     call    quit
```

```
1  ;-----------------------------------------
2  ; void iprint(Integer number)
3  ; Integer printing function (itoa)
4  iprint:
5      push    eax             ; preserve eax on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     ecx, 0          ; counter of how many bytes we need to print in the end
10
11 divideLoop:
12     inc     ecx             ; count each byte to print - number of characters
13     mov     edx, 0          ; empty edx
14     mov     esi, 10         ; mov 10 into esi
15     idiv    esi             ; divide eax by esi
16     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
17     push    edx             ; push edx (string representation of an intger) onto the stack
18     cmp     eax, 0          ; can the integer be divided anymore?
19     jnz     divideLoop      ; jump if not zero to the label divideLoop
20
21 printLoop:
22     dec     ecx             ; count down each byte that we put on the stack
23     mov     eax, esp        ; mov the stack pointer into eax for printing
24     call    sprint          ; call our string print function
25     pop     eax             ; remove last character from the stack to move esp forward
26     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
27     jnz     printLoop       ; jump is not zero to the label printLoop
28
29     pop     esi             ; restore esi from the value we pushed onto the stack at the start
30     pop     edx             ; restore edx from the value we pushed onto the stack at the start
31     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
32     pop     eax             ; restore eax from the value we pushed onto the stack at the start
33     ret
34
35
36 ;-----------------------------------------
37 ; void iprintLF(Integer number)
38 ; Integer printing function with linefeed (itoa)
39 iprintLF:
40     call    iprint          ; call our integer printing function
41
42     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
43     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
44     push    eax             ; push the linefeed onto the stack so we can get the address
45     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
46     call    sprint          ; call our sprint function
47     pop     eax             ; remove our linefeed character from the stack
48     pop     eax             ; restore the original value of eax before our function was called
49     ret
50
51
52 ;-----------------------------------------
```

```
53 ; int slen(String message)
54 ; String length calculation function
55 slen:
56     push    ebx
57     mov     ebx, eax
58
59 nextchar:
60     cmp     byte [eax], 0
61     jz      finished
62     inc     eax
63     jmp     nextchar
64
65 finished:
66     sub     eax, ebx
67     pop     ebx
68     ret
69
70
71 ;------------------------------------------
72 ; void sprint(String message)
73 ; String printing function
74 sprint:
75     push    edx
76     push    ecx
77     push    ebx
78     push    eax
79     call    slen
80
81     mov     edx, eax
82     pop     eax
83
84     mov     ecx, eax
85     mov     ebx, 1
86     mov     eax, 4
87     int     80h
88
89     pop     ebx
90     pop     ecx
91     pop     edx
92     ret
93
94
95 ;------------------------------------------
96 ; void sprintLF(String message)
97 ; String printing with line feed function
98 sprintLF:
99     call    sprint
100
101    push    eax
102    mov     eax, 0AH
103    push    eax
104    mov     eax, esp
105    call    sprint
106    pop     eax
107    pop     eax
108    ret
109
110
111 ;------------------------------------------
112 ; void exit()
113 ; Exit program and restore resources
114 quit:
115    mov     ebx, 0
116    mov     eax, 1
117    int     80h
118    ret
```

```
1  ; Calculator (Addition)
2  ; Compile with: nasm -f elf calculator-addition.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 calculator-addition.o -o calculator-
       addition
4  ; Run with: ./calculator-addition
5
6  %include        'functions.asm'
7
8  SECTION .text
9  global  _start
10
11 _start:
12
13     mov     eax, 90     ; move our first number into eax
```

```
    mov     ebx, 9      ; move our second number into ebx
    add     eax, ebx    ; add ebx to eax
    call    iprintLF    ; call our integer print with linefeed function

    call    quit
```

```
FILENAME=calculator-addition

all: cleanup
    nasm -f elf $(FILENAME).asm
    ld -m elf_i386 $(FILENAME).o -o $(FILENAME)
    ./$(FILENAME)

cleanup:
    rm -f $(FILENAME).o
    rm -f $(FILENAME)
```

```
;-----------------------------------------
; void iprint(Integer number)
; Integer printing function (itoa)
iprint:
    push    eax             ; preserve eax on the stack to be restored after function runs
    push    ecx             ; preserve ecx on the stack to be restored after function runs
    push    edx             ; preserve edx on the stack to be restored after function runs
    push    esi             ; preserve esi on the stack to be restored after function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    idiv    esi             ; divide eax by esi
    add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
          instruction
    push    edx             ; push edx (string representation of an intger) onto the stack
    cmp     eax, 0          ; can the integer be divided anymore?
    jnz     divideLoop      ; jump if not zero to the label divideLoop

printLoop:
    dec     ecx             ; count down each byte that we put on the stack
    mov     eax, esp        ; mov the stack pointer into eax for printing
    call    sprint          ; call our string print function
    pop     eax             ; remove last character from the stack to move esp forward
    cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
    jnz     printLoop       ; jump is not zero to the label printLoop

    pop     esi             ; restore esi from the value we pushed onto the stack at the start
    pop     edx             ; restore edx from the value we pushed onto the stack at the start
    pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
    pop     eax             ; restore eax from the value we pushed onto the stack at the start
    ret


;-----------------------------------------
; void iprintLF(Integer number)
; Integer printing function with linefeed (itoa)
iprintLF:
    call    iprint          ; call our integer printing function

    push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
         function
    mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
    push    eax             ; push the linefeed onto the stack so we can get the address
    mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
    call    sprint          ; call our sprint function
    pop     eax             ; remove our linefeed character from the stack
    pop     eax             ; restore the original value of eax before our function was called
    ret


;-----------------------------------------
; int slen(String message)
; String length calculation function
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 sprintLF(String message)
; String printing with line feed function
sprintLF:
    call    sprint

    push    eax
    mov     eax, 0AH
    push    eax
    mov     eax, esp
    call    sprint
    pop     eax
    pop     eax
    ret


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

```
; Calculator (Subtraction)
; Compile with: nasm -f elf calculator-subtraction.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 calculator-subtraction.o -o calculator-
    subtraction
; Run with: ./calculator-subtraction

%include        'functions.asm'

SECTION .text
global  _start

_start:

    mov     eax, 90     ; move our first number into eax
    mov     ebx, 9      ; move our second number into ebx
    sub     eax, ebx    ; subtract ebx from eax
    call    iprintLF    ; call our integer print with linefeed function

    call    quit
```

```
1  ;-----------------------------------------
2  ; void iprint(Integer number)
3  ; Integer printing function (itoa)
4  iprint:
5      push    eax             ; preserve eax on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     ecx, 0          ; counter of how many bytes we need to print in the end
10
11 divideLoop:
12     inc     ecx             ; count each byte to print - number of characters
13     mov     edx, 0          ; empty edx
14     mov     esi, 10         ; mov 10 into esi
15     idiv    esi             ; divide eax by esi
16     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
17     push    edx             ; push edx (string representation of an intger) onto the stack
18     cmp     eax, 0          ; can the integer be divided anymore?
19     jnz     divideLoop      ; jump if not zero to the label divideLoop
20
21 printLoop:
22     dec     ecx             ; count down each byte that we put on the stack
23     mov     eax, esp        ; mov the stack pointer into eax for printing
24     call    sprint          ; call our string print function
25     pop     eax             ; remove last character from the stack to move esp forward
26     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
27     jnz     printLoop       ; jump is not zero to the label printLoop
28
29     pop     esi             ; restore esi from the value we pushed onto the stack at the start
30     pop     edx             ; restore edx from the value we pushed onto the stack at the start
31     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
32     pop     eax             ; restore eax from the value we pushed onto the stack at the start
33     ret
34
35
36 ;-----------------------------------------
37 ; void iprintLF(Integer number)
38 ; Integer printing function with linefeed (itoa)
39 iprintLF:
40     call    iprint          ; call our integer printing function
41
42     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
43     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
44     push    eax             ; push the linefeed onto the stack so we can get the address
45     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
46     call    sprint          ; call our sprint function
47     pop     eax             ; remove our linefeed character from the stack
48     pop     eax             ; restore the original value of eax before our function was called
49     ret
50
51
52 ;-----------------------------------------
53 ; int slen(String message)
54 ; String length calculation function
55 slen:
56     push    ebx
57     mov     ebx, eax
58
59 nextchar:
60     cmp     byte [eax], 0
61     jz      finished
62     inc     eax
63     jmp     nextchar
64
65 finished:
66     sub     eax, ebx
67     pop     ebx
68     ret
69
70
71 ;-----------------------------------------
72 ; void sprint(String message)
73 ; String printing function
74 sprint:
75     push    edx
76     push    ecx
77     push    ebx
78     push    eax
79     call    slen
```

```
80
81      mov     edx, eax
82      pop     eax
83
84      mov     ecx, eax
85      mov     ebx, 1
86      mov     eax, 4
87      int     80h
88
89      pop     ebx
90      pop     ecx
91      pop     edx
92      ret
93
94
95  ;-----------------------------------------
96  ; void sprintLF(String message)
97  ; String printing with line feed function
98  sprintLF:
99      call    sprint
100
101     push    eax
102     mov     eax, 0AH
103     push    eax
104     mov     eax, esp
105     call    sprint
106     pop     eax
107     pop     eax
108     ret
109
110
111 ;-----------------------------------------
112 ; void exit()
113 ; Exit program and restore resources
114 quit:
115     mov     ebx, 0
116     mov     eax, 1
117     int     80h
118     ret
```

```
1  ; Calculator (Multiplication)
2  ; Compile with: nasm -f elf calculator-multiplication.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 calculator-multiplication.o -o calculator-
       multiplication
4  ; Run with: ./calculator-multiplication
5
6  %include        'functions.asm'
7
8  SECTION .text
9  global  _start
10
11 _start:
12
13      mov     eax, 90     ; move our first number into eax
14      mov     ebx, 9      ; move our second number into ebx
15      mul     ebx         ; multiply eax by ebx
16      call    iprintLF    ; call our integer print with linefeed function
17
18      call    quit
```

```
1  ;-----------------------------------------
2  ; void iprint(Integer number)
3  ; Integer printing function (itoa)
4  iprint:
5      push    eax             ; preserve eax on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     ecx, 0          ; counter of how many bytes we need to print in the end
10
11 divideLoop:
12     inc     ecx             ; count each byte to print - number of characters
13     mov     edx, 0          ; empty edx
14     mov     esi, 10         ; mov 10 into esi
15     idiv    esi             ; divide eax by esi
16     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
17     push    edx             ; push edx (string representation of an intger) onto the stack
18     cmp     eax, 0          ; can the integer be divided anymore?
```

```
19      jnz     divideLoop      ; jump if not zero to the label divideLoop
20
21  printLoop:
22      dec     ecx             ; count down each byte that we put on the stack
23      mov     eax, esp        ; mov the stack pointer into eax for printing
24      call    sprint          ; call our string print function
25      pop     eax             ; remove last character from the stack to move esp forward
26      cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
27      jnz     printLoop       ; jump is not zero to the label printLoop
28
29      pop     esi             ; restore esi from the value we pushed onto the stack at the start
30      pop     edx             ; restore edx from the value we pushed onto the stack at the start
31      pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
32      pop     eax             ; restore eax from the value we pushed onto the stack at the start
33      ret
34
35
36  ;-----------------------------------------
37  ; void iprintLF(Integer number)
38  ; Integer printing function with linefeed (itoa)
39  iprintLF:
40      call    iprint          ; call our integer printing function
41
42      push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
            function
43      mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
44      push    eax             ; push the linefeed onto the stack so we can get the address
45      mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
46      call    sprint          ; call our sprint function
47      pop     eax             ; remove our linefeed character from the stack
48      pop     eax             ; restore the original value of eax before our function was called
49      ret
50
51
52  ;-----------------------------------------
53  ; int slen(String message)
54  ; String length calculation function
55  slen:
56      push    ebx
57      mov     ebx, eax
58
59  nextchar:
60      cmp     byte [eax], 0
61      jz      finished
62      inc     eax
63      jmp     nextchar
64
65  finished:
66      sub     eax, ebx
67      pop     ebx
68      ret
69
70
71  ;-----------------------------------------
72  ; void sprint(String message)
73  ; String printing function
74  sprint:
75      push    edx
76      push    ecx
77      push    ebx
78      push    eax
79      call    slen
80
81      mov     edx, eax
82      pop     eax
83
84      mov     ecx, eax
85      mov     ebx, 1
86      mov     eax, 4
87      int     80h
88
89      pop     ebx
90      pop     ecx
91      pop     edx
92      ret
93
94
95  ;-----------------------------------------
96  ; void sprintLF(String message)
97  ; String printing with line feed function
98  sprintLF:
```

```
99      call    sprint
100
101     push    eax
102     mov     eax, 0AH
103     push    eax
104     mov     eax, esp
105     call    sprint
106     pop     eax
107     pop     eax
108     ret
109
110
111 ;----------------------------------------
112 ; void exit()
113 ; Exit program and restore resources
114 quit:
115     mov     ebx, 0
116     mov     eax, 1
117     int     80h
118     ret
```

```
1  ; Calculator (Division)
2  ; Compile with: nasm -f elf calculator-division.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 calculator-division.o -o calculator-
       division
4  ; Run with: ./calculator-division
5
6  %include        'functions.asm'
7
8  SECTION .data
9  msg1        db      ' remainder '      ; a message string to correctly output result
10
11 SECTION .text
12 global  _start
13
14 _start:
15
16      mov     eax, 90     ; move our first number into eax
17      mov     ebx, 9      ; move our second number into ebx
18      div     ebx         ; divide eax by ebx
19      call    iprint      ; call our integer print function on the quotient
20      mov     eax, msg1   ; move our message string into eax
21      call    sprint      ; call our string print function
22      mov     eax, edx    ; move our remainder into eax
23      call    iprintLF    ; call our integer printing with linefeed function
24
25      call    quit
```

```
1  ;----------------------------------------
2  ; int atoi(Integer number)
3  ; Ascii to integer function (atoi)
4  atoi:
5       push    ebx             ; preserve ebx on the stack to be restored after function runs
6       push    ecx             ; preserve ecx on the stack to be restored after function runs
7       push    edx             ; preserve edx on the stack to be restored after function runs
8       push    esi             ; preserve esi on the stack to be restored after function runs
9       mov     esi, eax        ; move pointer in eax into esi (our number to convert)
10      mov     eax, 0          ; initialise eax with decimal value 0
11      mov     ecx, 0          ; initialise ecx with decimal value 0
12
13 .multiplyLoop:
14      xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
15      mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
16      cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
17      jl      .finished       ; jump if less than to label finished
18      cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
19      jg      .finished       ; jump if greater than to label finished
20      cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
            character)
21      je      .finished       ; jump if equal to label finished
22      cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
23      jz      .finished       ; jump if zero to label finished
24
25      sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
26      add     eax, ebx        ; add ebx to our interger value in eax
27      mov     ebx, 10         ; move decimal value 10 into ebx
28      mul     ebx             ; multiply eax by ebx to get place value
29      inc     ecx             ; increment ecx (our counter register)
30      jmp     .multiplyLoop   ; continue multiply loop
```

```
31
32 .finished:
33     mov     ebx, 10         ; move decimal value 10 into ebx
34     div     ebx             ; divide eax by value in ebx (in this case 10)
35     pop     esi             ; restore esi from the value we pushed onto the stack at the start
36     pop     edx             ; restore edx from the value we pushed onto the stack at the start
37     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38     pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39     ret
40
41
42 ;-----------------------------------------
43 ; void iprint(Integer number)
44 ; Integer printing function (itoa)
45 iprint:
46     push    eax             ; preserve eax on the stack to be restored after function runs
47     push    ecx             ; preserve ecx on the stack to be restored after function runs
48     push    edx             ; preserve edx on the stack to be restored after function runs
49     push    esi             ; preserve esi on the stack to be restored after function runs
50     mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52 .divideLoop:
53     inc     ecx             ; count each byte to print - number of characters
54     mov     edx, 0          ; empty edx
55     mov     esi, 10         ; mov 10 into esi
56     idiv    esi             ; divide eax by esi
57     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
58     push    edx             ; push edx (string representation of an intger) onto the stack
59     cmp     eax, 0          ; can the integer be divided anymore?
60     jnz     .divideLoop      ; jump if not zero to the label divideLoop
61
62 .printLoop:
63     dec     ecx             ; count down each byte that we put on the stack
64     mov     eax, esp        ; mov the stack pointer into eax for printing
65     call    sprint          ; call our string print function
66     pop     eax             ; remove last character from the stack to move esp forward
67     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68     jnz     .printLoop       ; jump is not zero to the label printLoop
69
70     pop     esi             ; restore esi from the value we pushed onto the stack at the start
71     pop     edx             ; restore edx from the value we pushed onto the stack at the start
72     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73     pop     eax             ; restore eax from the value we pushed onto the stack at the start
74     ret
75
76
77 ;-----------------------------------------
78 ; void iprintLF(Integer number)
79 ; Integer printing function with linefeed (itoa)
80 iprintLF:
81     call    iprint          ; call our integer printing function
82
83     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
84     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85     push    eax             ; push the linefeed onto the stack so we can get the address
86     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87     call    sprint          ; call our sprint function
88     pop     eax             ; remove our linefeed character from the stack
89     pop     eax             ; restore the original value of eax before our function was called
90     ret
91
92
93 ;-----------------------------------------
94 ; int slen(String message)
95 ; String length calculation function
96 slen:
97     push    ebx
98     mov     ebx, eax
99
100 .nextchar:
101    cmp     byte [eax], 0
102    jz      .finished
103    inc     eax
104    jmp     .nextchar
105
106 .finished:
107    sub     eax, ebx
108    pop     ebx
109    ret
```

```
110
111
112 ;-----------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
136 ;-----------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;-----------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
1  ; Calculator (ATOI)
2  ; Compile with: nasm -f elf calculator-atoi.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 calculator-atoi.o -o calculator-atoi
4  ; Run with: ./calculator-atoi 20 1000 317
5
6  %include        'functions.asm'
7
8  SECTION .text
9  global  _start
10
11 _start:
12
13     pop     ecx             ; first value on the stack is the number of arguments
14     mov     edx, 0          ; initialise our data register to store additions
15
16 nextArg:
17     cmp     ecx, 0h         ; check to see if we have any arguments left
18     jz      noMoreArgs      ; if zero flag is set jump to noMoreArgs label (jumping over the end of the loop)
19     pop     eax             ; pop the next argument off the stack
20     call    atoi            ; convert our ascii string to decimal integer
21     add     edx, eax        ; perform our addition logic
22     dec     ecx             ; decrease ecx (number of arguments left) by 1
23     jmp     nextArg         ; jump to nextArg label
24
25 noMoreArgs:
26     mov     eax, edx        ; move our data result into eax for printing
27     call    iprintLF        ; call our integer printing with linefeed function
28     call    quit            ; call our quit function
```

```
1  ;-----------------------------------------
2  ; int atoi(Integer number)
3  ; Ascii to integer function (atoi)
4  atoi:
5      push    ebx             ; preserve ebx on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     esi, eax        ; move pointer in eax into esi (our number to convert)
10     mov     eax, 0          ; initialise eax with decimal value 0
11     mov     ecx, 0          ; initialise ecx with decimal value 0
12
13 .multiplyLoop:
14     xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
15     mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
16     cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
17     jl      .finished       ; jump if less than to label finished
18     cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
19     jg      .finished       ; jump if greater than to label finished
20     cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
           character)
21     je      .finished       ; jump if equal to label finished
22     cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
23     jz      .finished       ; jump if zero to label finished
24
25     sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
26     add     eax, ebx        ; add ebx to our interger value in eax
27     mov     ebx, 10         ; move decimal value 10 into ebx
28     mul     ebx             ; multiply eax by ebx to get place value
29     inc     ecx             ; increment ecx (our counter register)
30     jmp     .multiplyLoop   ; continue multiply loop
31
32 .finished:
33     mov     ebx, 10         ; move decimal value 10 into ebx
34     div     ebx             ; divide eax by value in ebx (in this case 10)
35     pop     esi             ; restore esi from the value we pushed onto the stack at the start
36     pop     edx             ; restore edx from the value we pushed onto the stack at the start
37     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38     pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39     ret
40
41
42 ;-----------------------------------------
43 ; void iprint(Integer number)
44 ; Integer printing function (itoa)
45 iprint:
46     push    eax             ; preserve eax on the stack to be restored after function runs
47     push    ecx             ; preserve ecx on the stack to be restored after function runs
48     push    edx             ; preserve edx on the stack to be restored after function runs
49     push    esi             ; preserve esi on the stack to be restored after function runs
50     mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52 .divideLoop:
53     inc     ecx             ; count each byte to print - number of characters
54     mov     edx, 0          ; empty edx
55     mov     esi, 10         ; mov 10 into esi
56     idiv    esi             ; divide eax by esi
57     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
58     push    edx             ; push edx (string representation of an intger) onto the stack
59     cmp     eax, 0          ; can the integer be divided anymore?
60     jnz     .divideLoop      ; jump if not zero to the label divideLoop
61
62 .printLoop:
63     dec     ecx             ; count down each byte that we put on the stack
64     mov     eax, esp        ; mov the stack pointer into eax for printing
65     call    sprint          ; call our string print function
66     pop     eax             ; remove last character from the stack to move esp forward
67     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68     jnz     .printLoop       ; jump is not zero to the label printLoop
69
70     pop     esi             ; restore esi from the value we pushed onto the stack at the start
71     pop     edx             ; restore edx from the value we pushed onto the stack at the start
72     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73     pop     eax             ; restore eax from the value we pushed onto the stack at the start
74     ret
75
76
77 ;-----------------------------------------
78 ; void iprintLF(Integer number)
79 ; Integer printing function with linefeed (itoa)
```

```
80  iprintLF:
81      call    iprint          ; call our integer printing function
82
83      push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
            function
84      mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85      push    eax             ; push the linefeed onto the stack so we can get the address
86      mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87      call    sprint          ; call our sprint function
88      pop     eax             ; remove our linefeed character from the stack
89      pop     eax             ; restore the original value of eax before our function was called
90      ret
91
92
93  ;------------------------------------------
94  ; int slen(String message)
95  ; String length calculation function
96  slen:
97      push    ebx
98      mov     ebx, eax
99
100 .nextchar:
101     cmp     byte [eax], 0
102     jz      .finished
103     inc     eax
104     jmp     .nextchar
105
106 .finished:
107     sub     eax, ebx
108     pop     ebx
109     ret
110
111
112 ;------------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
136 ;------------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;------------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
; Namespace
; Compile with: nasm -f elf namespace.asm
; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 namespace.o -o namespace
; Run with: ./namespace

%include        'functions.asm'

SECTION .data
msg1        db      'Jumping to finished label.', 0h        ; a message string
msg2        db      'Inside subroutine number: ', 0h        ; a message string
msg3        db      'Inside subroutine "finished".', 0h     ; a message string

SECTION .text
global  _start

_start:

subrountineOne:
    mov     eax, msg1       ; move the address of msg1 into eax
    call    sprintLF        ; call our string printing with linefeed function
    jmp     .finished       ; jump to the local label under the subrountineOne scope

.finished:
    mov     eax, msg2       ; move the address of msg2 into eax
    call    sprint          ; call our string printing function
    mov     eax, 1          ; move the value one into eax (for subroutine number one)
    call    iprintLF        ; call our integer printing function with linefeed function

subrountineTwo:
    mov     eax, msg1       ; move the address of msg1 into eax
    call    sprintLF        ; call our string print with linefeed function
    jmp     .finished       ; jump to the local label under the subrountineTwo scope

.finished:
    mov     eax, msg2       ; move the address of msg2 into eax
    call    sprint          ; call our string printing function
    mov     eax, 2          ; move the value two into eax (for subroutine number two)
    call    iprintLF        ; call our integer printing function with linefeed function

    mov     eax, msg1       ; move the address of msg1 into eax
    call    sprintLF        ; call our string printing with linefeed function
    jmp     finished        ; jump to the global label finished

finished:
    mov     eax, msg3       ; move the address of msg3 into eax
    call    sprintLF        ; call our string printing with linefeed function
    call    quit            ; call our quit function
```

```
;-----------------------------------------
; int atoi(Integer number)
; Ascii to integer function (atoi)
atoi:
    push    ebx             ; preserve ebx on the stack to be restored after function runs
    push    ecx             ; preserve ecx on the stack to be restored after function runs
    push    edx             ; preserve edx on the stack to be restored after function runs
    push    esi             ; preserve esi on the stack to be restored after function runs
    mov     esi, eax        ; move pointer in eax into esi (our number to convert)
    mov     eax, 0          ; initialise eax with decimal value 0
    mov     ecx, 0          ; initialise ecx with decimal value 0

.multiplyLoop:
    xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
    mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
    cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
    jl      .finished       ; jump if less than to label finished
    cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
    jg      .finished       ; jump if greater than to label finished
    cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
        character)
    je      .finished       ; jump if equal to label finished
    cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
    jz      .finished       ; jump if zero to label finished

    sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
    add     eax, ebx        ; add ebx to our interger value in eax
    mov     ebx, 10         ; move decimal value 10 into ebx
    mul     ebx             ; multiply eax by ebx to get place value
    inc     ecx             ; increment ecx (our counter register)
    jmp     .multiplyLoop   ; continue multiply loop
```

```
31
32 .finished:
33     mov     ebx, 10         ; move decimal value 10 into ebx
34     div     ebx             ; divide eax by value in ebx (in this case 10)
35     pop     esi             ; restore esi from the value we pushed onto the stack at the start
36     pop     edx             ; restore edx from the value we pushed onto the stack at the start
37     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38     pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39     ret
40
41
42 ;-----------------------------------------
43 ; void iprint(Integer number)
44 ; Integer printing function (itoa)
45 iprint:
46     push    eax             ; preserve eax on the stack to be restored after function runs
47     push    ecx             ; preserve ecx on the stack to be restored after function runs
48     push    edx             ; preserve edx on the stack to be restored after function runs
49     push    esi             ; preserve esi on the stack to be restored after function runs
50     mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52 .divideLoop:
53     inc     ecx             ; count each byte to print - number of characters
54     mov     edx, 0          ; empty edx
55     mov     esi, 10         ; mov 10 into esi
56     idiv    esi             ; divide eax by esi
57     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
58     push    edx             ; push edx (string representation of an intger) onto the stack
59     cmp     eax, 0          ; can the integer be divided anymore?
60     jnz     .divideLoop      ; jump if not zero to the label divideLoop
61
62 .printLoop:
63     dec     ecx             ; count down each byte that we put on the stack
64     mov     eax, esp        ; mov the stack pointer into eax for printing
65     call    sprint          ; call our string print function
66     pop     eax             ; remove last character from the stack to move esp forward
67     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68     jnz     .printLoop       ; jump is not zero to the label printLoop
69
70     pop     esi             ; restore esi from the value we pushed onto the stack at the start
71     pop     edx             ; restore edx from the value we pushed onto the stack at the start
72     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73     pop     eax             ; restore eax from the value we pushed onto the stack at the start
74     ret
75
76
77 ;-----------------------------------------
78 ; void iprintLF(Integer number)
79 ; Integer printing function with linefeed (itoa)
80 iprintLF:
81     call    iprint          ; call our integer printing function
82
83     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
84     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85     push    eax             ; push the linefeed onto the stack so we can get the address
86     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87     call    sprint          ; call our sprint function
88     pop     eax             ; remove our linefeed character from the stack
89     pop     eax             ; restore the original value of eax before our function was called
90     ret
91
92
93 ;-----------------------------------------
94 ; int slen(String message)
95 ; String length calculation function
96 slen:
97     push    ebx
98     mov     ebx, eax
99
100 .nextchar:
101    cmp     byte [eax], 0
102    jz      .finished
103    inc     eax
104    jmp     .nextchar
105
106 .finished:
107    sub     eax, ebx
108    pop     ebx
109    ret
```

```
110
111
112 ;-----------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
136 ;-----------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;-----------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
1  ; Fizzbuzz
2  ; Compile with: nasm -f elf fizzbuzz.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 fizzbuzz.o -o fizzbuzz
4  ; Run with: ./fizzbuzz
5
6  %include        'functions.asm'
7
8  SECTION .data
9  fizz        db      'Fizz', 0h      ; a message string
10 buzz        db      'Buzz', 0h      ; a message string
11
12 SECTION .text
13 global  _start
14
15 _start:
16
17     mov     esi, 0          ; initialise our checkFizz boolean variable
18     mov     edi, 0          ; initialise our checkBuzz boolean variable
19     mov     ecx, 0          ; initialise our counter variable
20
21 nextNumber:
22     inc     ecx             ; increment our counter variable
23
24 .checkFizz
25     mov     edx, 0          ; clear the edx register - this will hold our remainder after division
26     mov     eax, ecx        ; move the value of our counter into eax for division
27     mov     ebx, 3          ; move our number to divide by into ebx (in this case the value is 3)
28     div     ebx             ; divide eax by ebx
29     mov     edi, edx        ; move our remainder into edi (our checkFizz boolean variable)
30     cmp     edi, 0          ; compare if the remainder is zero (meaning the counter divides by 3)
```

```
31      jne     .checkBuzz      ; if the remainder is not equal to zero jump to local label checkBuzz
32      mov     eax, fizz       ; else move the address of our fizz string into eax for printing
33      call    sprint          ; call our string printing function
34
35  .checkBuzz:
36      mov     edx, 0          ; clear the edx register - this will hold our remainder after division
37      mov     eax, ecx        ; move the value of our counter into eax for division
38      mov     ebx, 5          ; move our number to divide by into ebx (in this case the value is 5)
39      div     ebx             ; divide eax by ebx
40      mov     esi, edx        ; move our remainder into edi (our checkBuzz boolean variable)
41      cmp     esi, 0          ; compare if the remainder is zero (meaning the counter divides by 5)
42      jne     .checkInt       ; if the remainder is not equal to zero jump to local label checkInt
43      mov     eax, buzz       ; else move the address of our buzz string into eax for printing
44      call    sprint          ; call our string printing function
45
46  .checkInt:
47      cmp     edi, 0          ; edi contains the remainder after the division in checkFizz
48      je     .continue        ; if equal (counter divides by 3) skip printing the integer
49      cmp     esi, 0          ; esi contains the remainder after the division in checkBuzz
50      je     .continue        ; if equal (counter divides by 5) skip printing the integer
51      mov     eax, ecx        ; else move the value in ecx (our counter) into eax for printing
52      call    iprint          ; call our integer printing function
53
54  .continue:
55      mov     eax, 0Ah        ; move an ascii linefeed character into eax
56      push    eax             ; push the address of eax onto the stack for printing
57      mov     eax, esp        ; get the stack pointer (address on the stack of our linefeed char)
58      call    sprint          ; call our string printing function to print a line feed
59      pop     eax             ; pop the stack so we don't waste resources
60      cmp     ecx, 100        ; compare if our counter is equal to 100
61      jne     nextNumber      ; if not equal jump to the start of the loop
62
63      call    quit            ; else call our quit function
```

```
1   ;-----------------------------------------
2   ; int atoi(Integer number)
3   ; Ascii to integer function (atoi)
4   atoi:
5       push    ebx             ; preserve ebx on the stack to be restored after function runs
6       push    ecx             ; preserve ecx on the stack to be restored after function runs
7       push    edx             ; preserve edx on the stack to be restored after function runs
8       push    esi             ; preserve esi on the stack to be restored after function runs
9       mov     esi, eax        ; move pointer in eax into esi (our number to convert)
10      mov     eax, 0          ; initialise eax with decimal value 0
11      mov     ecx, 0          ; initialise ecx with decimal value 0
12
13  .multiplyLoop:
14      xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
15      mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
16      cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
17      jl      .finished       ; jump if less than to label finished
18      cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
19      jg      .finished       ; jump if greater than to label finished
20      cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
            character)
21      je      .finished       ; jump if equal to label finished
22      cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
23      jz      .finished       ; jump if zero to label finished
24
25      sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
26      add     eax, ebx        ; add ebx to our interger value in eax
27      mov     ebx, 10         ; move decimal value 10 into ebx
28      mul     ebx             ; multiply eax by ebx to get place value
29      inc     ecx             ; increment ecx (our counter register)
30      jmp     .multiplyLoop   ; continue multiply loop
31
32  .finished:
33      mov     ebx, 10         ; move decimal value 10 into ebx
34      div     ebx             ; divide eax by value in ebx (in this case 10)
35      pop     esi             ; restore esi from the value we pushed onto the stack at the start
36      pop     edx             ; restore edx from the value we pushed onto the stack at the start
37      pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38      pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39      ret
40
41
42  ;-----------------------------------------
43  ; void iprint(Integer number)
44  ; Integer printing function (itoa)
```

```
45 iprint:
46     push    eax             ; preserve eax on the stack to be restored after function runs
47     push    ecx             ; preserve ecx on the stack to be restored after function runs
48     push    edx             ; preserve edx on the stack to be restored after function runs
49     push    esi             ; preserve esi on the stack to be restored after function runs
50     mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52 .divideLoop:
53     inc     ecx             ; count each byte to print - number of characters
54     mov     edx, 0          ; empty edx
55     mov     esi, 10         ; mov 10 into esi
56     idiv    esi             ; divide eax by esi
57     add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
           instruction
58     push    edx             ; push edx (string representation of an intger) onto the stack
59     cmp     eax, 0          ; can the integer be divided anymore?
60     jnz     .divideLoop      ; jump if not zero to the label divideLoop
61
62 .printLoop:
63     dec     ecx             ; count down each byte that we put on the stack
64     mov     eax, esp        ; mov the stack pointer into eax for printing
65     call    sprint          ; call our string print function
66     pop     eax             ; remove last character from the stack to move esp forward
67     cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68     jnz     .printLoop       ; jump is not zero to the label printLoop
69
70     pop     esi             ; restore esi from the value we pushed onto the stack at the start
71     pop     edx             ; restore edx from the value we pushed onto the stack at the start
72     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73     pop     eax             ; restore eax from the value we pushed onto the stack at the start
74     ret
75
76
77 ;-----------------------------------------
78 ; void iprintLF(Integer number)
79 ; Integer printing function with linefeed (itoa)
80 iprintLF:
81     call    iprint          ; call our integer printing function
82
83     push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
           function
84     mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85     push    eax             ; push the linefeed onto the stack so we can get the address
86     mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87     call    sprint          ; call our sprint function
88     pop     eax             ; remove our linefeed character from the stack
89     pop     eax             ; restore the original value of eax before our function was called
90     ret
91
92
93 ;-----------------------------------------
94 ; int slen(String message)
95 ; String length calculation function
96 slen:
97     push    ebx
98     mov     ebx, eax
99
100 .nextchar:
101     cmp     byte [eax], 0
102     jz      .finished
103     inc     eax
104     jmp     .nextchar
105
106 .finished:
107     sub     eax, ebx
108     pop     ebx
109     ret
110
111
112 ;-----------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
```

```
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
136 ;-----------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;-----------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
1  ; Execute
2  ; Compile with: nasm -f elf execute.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 execute.o -o execute
4  ; Run with: ./execute
5
6  %include        'functions.asm'
7
8  SECTION .data
9  command         db      '/bin/echo', 0h     ; command to execute
10 arg1            db      'Hello World!', 0h
11 arguments       dd      command
12                 dd      arg1                ; arguments to pass to commandline (in this case just one)
13                 dd      0h                  ; end the struct
14 environment     dd      0h                  ; arguments to pass as environment variables (inthis case none) end
       the struct
15
16 SECTION .text
17 global  _start
18
19 _start:
20
21     mov     edx, environment    ; address of environment variables
22     mov     ecx, arguments      ; address of the arguments to pass to the commandline
23     mov     ebx, command        ; address of the file to execute
24     mov     eax, 11             ; invoke SYS_EXECVE (kernel opcode 11)
25     int     80h
26
27     call    quit                ; call our quit function
```

```
1  ;-----------------------------------------
2  ; int atoi(Integer number)
3  ; Ascii to integer function (atoi)
4  atoi:
5      push    ebx             ; preserve ebx on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     esi, eax        ; move pointer in eax into esi (our number to convert)
10     mov     eax, 0          ; initialise eax with decimal value 0
11     mov     ecx, 0          ; initialise ecx with decimal value 0
12
13 .multiplyLoop:
```

```
14      xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
15      mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
16      cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
17      jl      .finished       ; jump if less than to label finished
18      cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
19      jg      .finished       ; jump if greater than to label finished
20      cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
            character)
21      je      .finished       ; jump if equal to label finished
22      cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
23      jz      .finished       ; jump if zero to label finished
24
25      sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
26      add     eax, ebx        ; add ebx to our interger value in eax
27      mov     ebx, 10         ; move decimal value 10 into ebx
28      mul     ebx             ; multiply eax by ebx to get place value
29      inc     ecx             ; increment ecx (our counter register)
30      jmp     .multiplyLoop   ; continue multiply loop
31
32  .finished:
33      mov     ebx, 10         ; move decimal value 10 into ebx
34      div     ebx             ; divide eax by value in ebx (in this case 10)
35      pop     esi             ; restore esi from the value we pushed onto the stack at the start
36      pop     edx             ; restore edx from the value we pushed onto the stack at the start
37      pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38      pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39      ret
40
41
42  ;-----------------------------------------
43  ; void iprint(Integer number)
44  ; Integer printing function (itoa)
45  iprint:
46      push    eax             ; preserve eax on the stack to be restored after function runs
47      push    ecx             ; preserve ecx on the stack to be restored after function runs
48      push    edx             ; preserve edx on the stack to be restored after function runs
49      push    esi             ; preserve esi on the stack to be restored after function runs
50      mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52  .divideLoop:
53      inc     ecx             ; count each byte to print - number of characters
54      mov     edx, 0          ; empty edx
55      mov     esi, 10         ; mov 10 into esi
56      idiv    esi             ; divide eax by esi
57      add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
            instruction
58      push    edx             ; push edx (string representation of an intger) onto the stack
59      cmp     eax, 0          ; can the integer be divided anymore?
60      jnz     .divideLoop     ; jump if not zero to the label divideLoop
61
62  .printLoop:
63      dec     ecx             ; count down each byte that we put on the stack
64      mov     eax, esp        ; mov the stack pointer into eax for printing
65      call    sprint          ; call our string print function
66      pop     eax             ; remove last character from the stack to move esp forward
67      cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68      jnz     .printLoop      ; jump is not zero to the label printLoop
69
70      pop     esi             ; restore esi from the value we pushed onto the stack at the start
71      pop     edx             ; restore edx from the value we pushed onto the stack at the start
72      pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73      pop     eax             ; restore eax from the value we pushed onto the stack at the start
74      ret
75
76
77  ;-----------------------------------------
78  ; void iprintLF(Integer number)
79  ; Integer printing function with linefeed (itoa)
80  iprintLF:
81      call    iprint          ; call our integer printing function
82
83      push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
            function
84      mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85      push    eax             ; push the linefeed onto the stack so we can get the address
86      mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87      call    sprint          ; call our sprint function
88      pop     eax             ; remove our linefeed character from the stack
89      pop     eax             ; restore the original value of eax before our function was called
90      ret
91
```

```
92
93  ;-----------------------------------------
94  ; int slen(String message)
95  ; String length calculation function
96  slen:
97      push    ebx
98      mov     ebx, eax
99
100 .nextchar:
101     cmp     byte [eax], 0
102     jz      .finished
103     inc     eax
104     jmp     .nextchar
105
106 .finished:
107     sub     eax, ebx
108     pop     ebx
109     ret
110
111
112 ;-----------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
136 ;-----------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;-----------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
1  ; Fork
2  ; Compile with: nasm -f elf fork.asm
3  ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 fork.o -o fork
4  ; Run with: ./fork
5
6  %include        'functions.asm'
7
8  SECTION .data
9  childMsg        db      'This is the child process', 0h     ; a message string
10 parentMsg       db      'This is the parent process', 0h    ; a message string
11
12 SECTION .text
```

```
13 global  _start
14
15 _start:
16
17     mov     eax, 2              ; invoke SYS_FORK (kernel opcode 2)
18     int     80h
19
20     cmp     eax, 0              ; if eax is zero we are in the child process
21     jz      child               ; jump if eax is zero to child label
22
23 parent:
24     mov     eax, parentMsg      ; inside our parent process move parentMsg into eax
25     call    sprintLF            ; call our string printing with linefeed function
26
27     call    quit                ; quit the parent process
28
29 child:
30     mov     eax, childMsg       ; inside our child process move childMsg into eax
31     call    sprintLF            ; call our string printing with linefeed function
32
33     call    quit                ; quit the child process
```

```
1  ;-----------------------------------------
2  ; int atoi(Integer number)
3  ; Ascii to integer function (atoi)
4  atoi:
5      push    ebx             ; preserve ebx on the stack to be restored after function runs
6      push    ecx             ; preserve ecx on the stack to be restored after function runs
7      push    edx             ; preserve edx on the stack to be restored after function runs
8      push    esi             ; preserve esi on the stack to be restored after function runs
9      mov     esi, eax        ; move pointer in eax into esi (our number to convert)
10     mov     eax, 0          ; initialise eax with decimal value 0
11     mov     ecx, 0          ; initialise ecx with decimal value 0
12
13 .multiplyLoop:
14     xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
15     mov     bl, [esi+ecx]   ; move a single byte into ebx register's lower half
16     cmp     bl, 48          ; compare ebx register's lower half value against ascii value 48 (char value 0)
17     jl      .finished       ; jump if less than to label finished
18     cmp     bl, 57          ; compare ebx register's lower half value against ascii value 57 (char value 9)
19     jg      .finished       ; jump if greater than to label finished
20     cmp     bl, 10          ; compare ebx register's lower half value against ascii value 10 (linefeed
           character)
21     je      .finished       ; jump if equal to label finished
22     cmp     bl, 0           ; compare ebx register's lower half value against decimal value 0 (end of string)
23     jz      .finished       ; jump if zero to label finished
24
25     sub     bl, 48          ; convert ebx register's lower half to decimal representation of ascii value
26     add     eax, ebx        ; add ebx to our interger value in eax
27     mov     ebx, 10         ; move decimal value 10 into ebx
28     mul     ebx             ; multiply eax by ebx to get place value
29     inc     ecx             ; increment ecx (our counter register)
30     jmp     .multiplyLoop   ; continue multiply loop
31
32 .finished:
33     mov     ebx, 10         ; move decimal value 10 into ebx
34     div     ebx             ; divide eax by value in ebx (in this case 10)
35     pop     esi             ; restore esi from the value we pushed onto the stack at the start
36     pop     edx             ; restore edx from the value we pushed onto the stack at the start
37     pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
38     pop     ebx             ; restore ebx from the value we pushed onto the stack at the start
39     ret
40
41
42 ;-----------------------------------------
43 ; void iprint(Integer number)
44 ; Integer printing function (itoa)
45 iprint:
46     push    eax             ; preserve eax on the stack to be restored after function runs
47     push    ecx             ; preserve ecx on the stack to be restored after function runs
48     push    edx             ; preserve edx on the stack to be restored after function runs
49     push    esi             ; preserve esi on the stack to be restored after function runs
50     mov     ecx, 0          ; counter of how many bytes we need to print in the end
51
52 .divideLoop:
53     inc     ecx             ; count each byte to print - number of characters
54     mov     edx, 0          ; empty edx
55     mov     esi, 10         ; mov 10 into esi
56     idiv    esi             ; divide eax by esi
```

```
57      add     edx, 48         ; convert edx to it's ascii representation - edx holds the remainder after a divide
            instruction
58      push    edx             ; push edx (string representation of an intger) onto the stack
59      cmp     eax, 0          ; can the integer be divided anymore?
60      jnz     .divideLoop     ; jump if not zero to the label divideLoop
61
62  .printLoop:
63      dec     ecx             ; count down each byte that we put on the stack
64      mov     eax, esp        ; mov the stack pointer into eax for printing
65      call    sprint          ; call our string print function
66      pop     eax             ; remove last character from the stack to move esp forward
67      cmp     ecx, 0          ; have we printed all bytes we pushed onto the stack?
68      jnz     .printLoop      ; jump is not zero to the label printLoop
69
70      pop     esi             ; restore esi from the value we pushed onto the stack at the start
71      pop     edx             ; restore edx from the value we pushed onto the stack at the start
72      pop     ecx             ; restore ecx from the value we pushed onto the stack at the start
73      pop     eax             ; restore eax from the value we pushed onto the stack at the start
74      ret
75
76
77  ;-----------------------------------------
78  ; void iprintLF(Integer number)
79  ; Integer printing function with linefeed (itoa)
80  iprintLF:
81      call    iprint          ; call our integer printing function
82
83      push    eax             ; push eax onto the stack to preserve it while we use the eax register in this
            function
84      mov     eax, 0Ah        ; move 0Ah into eax - 0Ah is the ascii character for a linefeed
85      push    eax             ; push the linefeed onto the stack so we can get the address
86      mov     eax, esp        ; move the address of the current stack pointer into eax for sprint
87      call    sprint          ; call our sprint function
88      pop     eax             ; remove our linefeed character from the stack
89      pop     eax             ; restore the original value of eax before our function was called
90      ret
91
92
93  ;-----------------------------------------
94  ; int slen(String message)
95  ; String length calculation function
96  slen:
97      push    ebx
98      mov     ebx, eax
99
100 .nextchar:
101     cmp     byte [eax], 0
102     jz      .finished
103     inc     eax
104     jmp     .nextchar
105
106 .finished:
107     sub     eax, ebx
108     pop     ebx
109     ret
110
111
112 ;-----------------------------------------
113 ; void sprint(String message)
114 ; String printing function
115 sprint:
116     push    edx
117     push    ecx
118     push    ebx
119     push    eax
120     call    slen
121
122     mov     edx, eax
123     pop     eax
124
125     mov     ecx, eax
126     mov     ebx, 1
127     mov     eax, 4
128     int     80h
129
130     pop     ebx
131     pop     ecx
132     pop     edx
133     ret
134
135
```

```
136 ;-----------------------------------------
137 ; void sprintLF(String message)
138 ; String printing with line feed function
139 sprintLF:
140     call    sprint
141
142     push    eax
143     mov     eax, 0AH
144     push    eax
145     mov     eax, esp
146     call    sprint
147     pop     eax
148     pop     eax
149     ret
150
151
152 ;-----------------------------------------
153 ; void exit()
154 ; Exit program and restore resources
155 quit:
156     mov     ebx, 0
157     mov     eax, 1
158     int     80h
159     ret
```

```
 1 ; Time
 2 ; Compile with: nasm -f elf time.asm
 3 ; Link with (64 bit systems require elf_i386 option): ld -m elf_i386 time.o -o time
 4 ; Run with: ./time
 5
 6 %include        'functions.asm'
 7
 8 SECTION .data
 9 msg        db      'Seconds since Jan 01 1970: ', 0h     ; a message string
10
11 SECTION .text
12 global  _start
13
14 _start:
15
16     mov     eax, msg        ; move our message string into eax for printing
17     call    sprint          ; call our string printing function
18
19     mov     eax, 13         ; invoke SYS_TIME (kernel opcode 13)
20     int     80h             ; call the kernel
21
22     call    iprintLF        ; call our integer printing function with linefeed
23     call    quit            ; call our quit function
```