ASM

Example Program

```
section .data
        LookupTable:            db "0123456789ABCDEF"   ; Lookup table to convert a number to its hex
equivalent
        TemplateString:         db " 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00"
        StringLength:           equ $ - TemplateString

section .bss
        BufferLength:   equ 16
        Buffer:         resb BufferLength

section .text
        global  _start

EXTERN  printText, printNewline

_start:

ReadIntoBuffer:
        mov     rax, 3
        mov     rbx, 0
        mov     rcx, Buffer
        mov     rdx, BufferLength
        int     0x80

        push    rax                     ; Save the number of read bytes


        xor     rcx, rcx                ; rcx is our 'counter'

ConvertBytes:

        xor     rdx, rdx
        mov     rbx, rcx
        lea     rbx, [rbx * 2 + rbx]
        mov     dl, byte [Buffer + rcx]
        mov     al, dl

        and     al, 0x0F
        mov     al, [LookupTable + rax]
        mov     [TemplateString + rbx + 2], byte al


        shr     dl, 4
        mov     dl, [LookupTable + rdx]
        mov     [TemplateString + rbx + 1], byte dl

        inc     rcx
        cmp     rcx, BufferLength
        jne     ConvertBytes


        ; Print out the template
        mov     rcx, TemplateString
        mov     rdx, StringLength
        call    printText
        call    printNewline

        pop     rax
        cmp     rax, 0
```

```
        jne     ReadIntoBuffer          ; Continue to read if we're not at the end

exit:
        mov     rax, 1
        mov     rbx, 0
        int     0x80
```

Example Library

```
section .data
        Newline:        db 0xA, 0xD     ; Newline characters
        NewlineLength:  equ $ - Newline ; The length of the newline characters

section .text

        GLOBAL  printText, printNewline, convertNumberToString, binaryLog
        GLOBAL  numberToBinaryString

;*****************************************************************************
; Prints a text to the console
;
; Input:        rcx: The address of the ASCII string
;               rdx: The length of the string
;
; Output:       None
;*****************************************************************************
printText:
        push    rax
        push    rbx
        mov     rax, 4
        mov     rbx, 1
        int     0x80
        pop     rbx
        pop     rax
        ret

;*****************************************************************************
; Prints a newline
;
; Input:        None
;
; Output:       None
;*****************************************************************************
printNewline:
        push    rcx
        push    rdx
        mov     rcx, Newline
        mov     rdx, NewlineLength
        call    printText
        pop     rdx
        pop     rcx
        ret

clearNumberString:
        push    rax
        push    rbx
        push    rcx
        xor     rax, rax
        mov     rcx, 15
.clear:
        mov     [rbx + rcx], al
        dec     rcx
        jnz     .clear
```

```
        pop     rcx
        pop     rbx
        pop     rax
ret

;*****************************************************************************
; Converts a number to an ascii string.
;
; Input:        rax: The number that will be converted.
;               rbx: The memory address where the number will be placed
;
; Output:       rcx: The memory address where the string is stored
;*****************************************************************************
convertNumberToString:
        push    rax
        push    rbx
        push    rcx
        push    rdx
        call    clearNumberString
        add     rbx, 15
        mov     ecx, 0xA
.division:
        xor     edx, edx
        idiv    ecx             ; Divide the number by 10
        add     edx, "0"        ; Convert the remainder to an ASCII character
        mov     [rbx], dl       ; Move the character to memory
        dec     ebx             ; Decrement rbx
        cmp     eax, 0          ; Check if we're finished with converting
        jg      .division       ; If not we convert the next digit

        pop     rdx
        pop     rcx
        pop     rbx
        pop     rax
        ret

;*****************************************************************************
; Calculate the binary logarithm of a given number.
;
; Input:        rax: That number we want the logarithm of.
;
; Output:       rbx: The binary logarithm of rax
;*****************************************************************************
binaryLog:
        push    rax
        xor     rbx, rbx        ; This will be the logarithm
.calculate:
        shr     rax, 1          ; Divide rax by 2
        jz      .exit           ; If it's 0 exit the procedure
        inc     rbx             ; Otherwise increase rbx by one
        jmp     .calculate      ; and repeat this
.exit:
        pop     rax
        ret

;*****************************************************************************
; Convert a number to a binary string.
;
; Input:        eax: The number we want to convert.
;               rbx: The address where the string will be stored.
;
; Output:       The string will be written to the address in rbx
;*****************************************************************************
numberToBinaryString:
        push    rax
```

```
        push    rbx
        push    rcx
        push    rdx
        xor     rcx, rcx                ; The iteration counter
.loop:
        push    rax
        and     eax, 0x80000000 ; Get the most significant byte
        mov     edx, eax
        shr     edx, 31
        pop     rax
        add     edx, "0"
        mov     [rbx + rcx], edx

.prepareIteration:
        shl     eax, 1
        inc     rcx
        cmp     rcx, 32
        jne     .loop

        pop     rdx
        pop     rcx
        pop     rbx
        pop     rax
        ret
```

Example Makefile

```
all: hello

clean:
        rm -f *.o > /dev/null

hello:  hello.o
        ld -o hello hello.o ../lib/lib.o

hello.o: hello.asm
        nasm -f elf64 -g -F dwarf hello.asm
```