```
                                 ; convert hex to ascii using offset values
                                 ; Jared Dyreson
                                 ; CPSC-240-09 TR @ 11:30 - 13:20

                                 ; some light -> https://stackoverflow.com/questions/36336045/print-register-value-to-console
                                 ; I also included a bash script that I was using to compile the ASM files, thought it might be useful for the class
                                 global _start

                                 SECTION .text

                                 _start:
00000000 BF01000000                mov rdi, 1
00000005 8A0425[11000000]          mov al, byte[bval]
0000000C 488D1C25[00000000]        lea rbx, [xtable]
00000014 D7                        xlat
00000015 880425[11000000]          mov byte[bval], al
0000001C 48BE-                     mov rsi, bval
0000001E [1100000000000000]
00000026 4831C0                    xor rax, rax
00000029 B801000000                mov rax, 1
0000002E BA02000000                mov rdx, 2 ; if you pass in length, it prints the content of the a.out bin for some odd, inexplicable reason
00000033 0F05                      syscall
00000035 B83C000000                mov rax, 60
0000003A BF00000000                mov rdi, 0
0000003F 0F05                      syscall

                                 SECTION .data

                                 length: equ $bval
00000000 303132333435363738-     xtable: db ’0123456789ABCDEF’, 10
00000009 394142434445460A
00000011 0A0A                    bval: db 10, 10
```

Character Translation Methods

One task that assembly language programs handle best is character translation. There are many computer applications for this. The rapidly expanding field of data encryption is one, where data files must be securely stored and transmitted between computers while their contents are kept secret. Another application is data communications, where keyboard and screen codes must be translated in order to emulate various terminals. Often, we need to translate characters from one encoding system to another—from ASCII to EBCDIC, for example. A critical factor in each of these applications is speed, which just happens to be a feature of assembly language.

The XLAT Instruction

The XLAT instruction adds AL to EBX and uses the resulting offset to point to an entry in an 8-bit *translate table*. This table contains values that are substituted for the original value in AL. The byte in the table entry pointed to by EBX + AL is moved to AL. The syntax is

```
XLAT  [tablename]
```

Tablename is optional because the table is assumed to be pointed to by EBX (or BX, in Real-address mode). Therefore, be sure to load BX with the offset of the translate table before invoking XLAT. The flags are not affected by this instruction. The table can have a maximum of 256 entries, the same range of values possible in the 8-bit AL register.

Example Let's store the characters representing all 16 hexadecimal digits in a table:

```
table BYTE '0123456789ABCDEF'
```

The table contains the ASCII code of each hexadecimal digit. If we place 0Ah in AL with the thought of converting it to ASCII, we can set EBX to the table offset and invoke XLAT. The instruction adds EBX and AL, generating an effective address that points to the eleventh entry in the table. It then moves the contents of this table entry to A:

```
    mov al,0Ah                  ; index value
    mov ebx,OFFSET table        ; point to table
    xlat                        ; AL = 41h, or 'A'
```