System Programming Lab
COE – 410

By – Divjot Singh
262/CO/12
COE – 1

Index

Assembler Used: NASM for OS X 10.11 (UNIX based)

Build Instructions:

```
nasm –f macho filename.asm
ld –o programName –e startLabel filename.o
```

Execution Instruction:

```
./programName
```

Experiment 1

Code

```
; Assembly program that reads a character from the keyboard and writes the
character on the screen.
global start ; making start visible externally

; uninitialized data goes to bss section
section .bss
input resb 2

; initialized data goes to data section
section .data
msg db 'Enter a character and press enter after it: '
msgLen equ $ - msg

; code section
section .text
start:
  call _prompt        ; print welcome msg

  ; read (int fileDescriptor, void* buff, size_t count)
  push dword 1        ; 1 character to be read
  push dword input    ; input buffer
  push dword 0        ; file descriptor number for STDIN in 0
  sub esp, 4          ; unix requriement of extra space in stack

  mov eax, 3          ; system call number for read is 3
  int 0x80            ; invoke system call

  add esp, 16         ; clearing stack (4 bytes x 3 args + 4 extra space)

  ; appending input with carriage return
  mov dword [input + 1], 0xA

  ; write (int fileDescriptor, void* buff, size_t count)
  push dword 2        ; 2 character to be printed
  push dword input    ; write the input
  push dword 1        ; file descriptor number for STDOUT is 1
  sub esp, 4          ; unix requirement of extra space in stack

  mov eax, 4          ; system call number for write is 4
  int 0x80            ; invoke system call

  add esp, 16         ; clearing stack (4 bytes x 3 args + 4 extra space)
```

```
; exit (int statusCode)
push dword 0            ; exit status code
sub esp, 4              ; unix requirement of extra space in stack

mov eax, 1              ; system call number for exit is 1
int 0x80                ; invoke system call

; Prints welcome messsage of the program
_prompt:
 ; write (int fileDescriptor, void* buff, size_t count)
 push dword msgLen ; length of message
 push dword msg     ; address of message
 push dword 1       ; STDOUT
 sub esp, 4         ; unix requirement of extra space in stack

 mov eax, 4         ; system call number for write is 4
 int 0x80           ; invoke system call

 add esp, 16        ; clearing stack (4 bytes x 3 args + 4 extra space)
 ret
```

Output

```
>nasm -f macho inputFromKeyboard.asm && ld -o key -e start inputFromKeyboard.o && ./key
Enter a character and press enter after it: f
f
```

Macros.asm

Since printing, reading and exiting system calls are readily used, I've created a macros.asm file to use them in a quicker way.

Code

```
; Usage: print <stringPtrName> <length>
%macro print 2
  ; write (int fileDescriptor, void* buff, size_t count)
  push dword %2       ; length of input
  push dword %1       ; pointer to the input
  push dword 1        ; file descriptor number for STDOUT is 1
  sub esp, 4          ; unix requirement of extra space in stack
  mov eax, 4          ; system call number for write is 4
  int 0x80            ; system call
  add esp, 16         ; clearing stack (4 bytes x 3 args + 4 extra space)
%endmacro

; Usage: read <stringPtrName> <length>
%macro read 2
  ; read (int fileDescriptor, void* buff, size_t count)
  push dword %2       ; size of input buffer
  push dword %1       ; pointer to input buffer
  push dword 0        ; file descriptor number for STDIN in 0
  sub esp, 4          ; unix requriement of extra space in stack
  mov eax, 3          ; system call number for read is 3
  int 0x80            ; system call
%endmacro

; Usage:  exit <statusCode>
%macro exit 1
  ; exit (int statusCode)
  push dword %1    ; exit status code
  sub esp, 4       ; unix requirement of extra space in stack
  mov eax, 1       ; system call number for exit is 1
  int 0x80         ; system call
%endmacro
```

Experiment 2

Code

```
%include "../macros.asm" ; contains print, read and exit macros

; Assembly program that reverses given string
global start ; making start visible externally

section .text
start:
  ; Reading string
  print mEnterString, mEnterStringLength
  read input, 100
  mov dword [inputLength], eax

  ; Reversing
  mov ecx, 0
  mov edx, dword [inputLength]
  sub edx, 2                    ; ignoring 0xA from string and fixing 0 indexing
  mov eax, edx                  ; copy of last index

  ; Traversing input from behind and saving it to reversedInput in forward
directon
  loop: mov bl, byte [input + edx]          ; copy last of input to bl
      mov byte [reversedInput + ecx], bl  ; copy bl to first of reversedInput
      inc ecx                   ; increment first of reversedInput
      dec edx                   ; decrement last of input
      cmp ecx, eax              ; compare first of reversedInput to length of input - 1
      jle loop                  ; if not equal, continue

  ; Print result
  print mReversedString, mReversedStringLength
  print reversedInput, dword [inputLength]
  print mNewLine, mNewLineLength

  exit 0
```

```
section .bss
  input              resb 100
  reversedInput resb 100
  inputLength    resd 1

section .data
  mEnterString db 'Enter a string: '
  mEnterStringLength equ $ - mEnterString

  mReversedString db 'Reversed string: '
  mReversedStringLength equ $ - mReversedString

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine
```

Output

```
[>nasm -f macho reverse.asm && ld -o reverse -e start reverse.o && ./reverse
 Enter a string: esrever ni epyt nac I
 Reversed string: I can type in reverse
```

Experiment 3

Code

```
%include "../macros.asm"
; Assembly program that compares 2 given strings
global start ; making start visible externally

section .text
start:
  ; reading first string
  print outStr1, outStr1L
  read str1, 100d
  mov dword [str1L], eax

  ; reading second string
  print outStr2, outStr2L
  read str2, 100d
  mov dword [str2L], eax

  ; printing the strings
  print outS1, outS1L
  print str1, dword [str1L]
  print outS2, outS2L
  print str2, dword [str2L]

  ; comparing length
  mov eax, dword [str1L]
  cmp dword [str2L], eax
  jne printLNE

  ; comparing each character
  mov edx, -1
  _loop: inc edx
        cmp edx, dword [str1L]
        jg printE
        mov ebx, dword [str1 + edx]
        cmp ebx, dword [str2 + edx]
        je _loop
        jne printNE

  end: exit 0
```

```
  ; print that the two strings are not equal in length
  printLNE:
    print outLNE, outLNEL
    jmp end

  ; print that the two strings are not equal
  printNE:
    print outNE, outNEL
    jmp end

  ; print that the two strings are equal
  printE:
    print outE, outEL
    jmp end

section .bss
  str1 resb 100d
  str2 resb 100d
  str1L resd 1
  str2L resd 1

section .data
  outS1 db 'String 1 : '
  outS1L equ $ - outS1

  outS2 db 'String 2 : '
  outS2L equ $ - outS2

  outLNE db 'Length of given strings not equal', 0xA
  outLNEL equ $ - outLNE

  outNE db 'Given strings are not equal', 0xA
  outNEL equ $ - outNE

  outE db 'Given strings are equal', 0xA
  outEL equ $ - outE

  outStr1 db 'Enter String 1: '
  outStr1L equ $ - outStr1

  outStr2 db 'Enter String 2: '
  outStr2L equ $ - outStr2
```

Output

```
[>nasm -f macho compare.asm && ld -o compare -e start compare.o && ./compare
 Enter String 1: hello
 Enter String 2: bye
 String 1 : hello
 String 2 : bye
 Length of given strings not equal
 ~/work/code/CollegePrograms/SystemProgramming/compare 04:27:49 AM
[>./compare
 Enter String 1: hello
 Enter String 2: cello
 String 1 : hello
 String 2 : cello
 Given strings are not equal
 ~/work/code/CollegePrograms/SystemProgramming/compare 04:27:58 AM
[>./compare
 Enter String 1: hello
 Enter String 2: hello
 String 1 : hello
 String 2 : hello
 Given strings are equal
```

Experiment 4

Code

```
%include "../macros.asm" ; macros.asm contains print, read and exit macros

; Assembly program that tells whether given string is a palindrome or not
global start ; making start visible externally

; code section
section .text
start:
  ; reading string
  print mEnterString, mEnterStringLength
  read input, 100d
  mov dword [inputLength], eax

  ; printing the string
  print mYouEntered, mYouEnteredLength
  print input, dword [inputLength]

  mov eax, -1                    ; -1 index
  mov ebx, dword [inputLength]
  sub ebx, 1                     ; n index

  loop: inc eax                  ; increment start index
      dec ebx                    ; decrement end index
      cmp eax, ebx               ; compare them
      jge printP                 ; if greater than or equal, string is palindrome
      mov dl, byte [input + eax]     ; else, save input[eax] to dl
      mov dh, byte [input + ebx]     ;      save input[ebx] to dh
      cmp dl, dh                 ;      compare dl and dh
      jne printNP                ;      if not equal, string isn't palindrome
      je loop                    ;      else, continue

  end: exit 0

  ; print that the string is not a palindrome
  printNP:
    print mIsNotPalindrome, mIsNotPalindromeLength
    jmp end

  ; print that the string is a palindrome
  printP:
    print mIsPalindrome, mIsPalindromeLength
    jmp end
```

```
section .bss
  input resb 100d
  inputLength resd 1

section .data
  newLine db 0xA
  mYouEntered db 'You Entered : '
  mYouEnteredLength equ $ - mYouEntered

  mIsNotPalindrome db 'Given string is not a palindrome', 0xA
  mIsNotPalindromeLength equ $ - mIsNotPalindrome

  mIsPalindrome db 'Given string is a palindrome', 0xA
  mIsPalindromeLength equ $ - mIsPalindrome

  mEnterString db 'Enter a String: '
  mEnterStringLength equ $ - mEnterString
```

Output

```
>./palindrome
Enter a String: jersey
You Entered : jersey
Given string is not a palindrome
~/work/code/CollegePrograms/SystemProgramming/palindrome 04:32:43 AM
>./palindrome
Enter a String: nitin
You Entered : nitin
Given string is a palindrome
```

Experiment 5

Code

```
%include "../macros.asm"

; Program to print sum of first N natural numbers
global start

section .text
start:
  ; Taking input
  print mEnterN, mEnterNLength
  read input, 100
  dec eax               ; ignoring 0xA from length
  mov dword [inputLength], eax

  call dec2AX

  ; Computing sum
  mov ebx, eax          ; n in BX
  inc ebx               ; n + 1 in BX
  mul ebx               ; n*(n + 1) in DX AX
  shr eax, 1            ; n*(n + 1)/2 in AX

  call AX2dec

  ; Printing output
  print mSum, mSumLength
  print input, [inputLength]
  print mNewLine, mNewLineLength

  exit 0

; Converts ascii decimal number string [input] -> [input + inputLength] to hex
number in EAX
dec2AX:
  push ebx                  ; saving BX before overwriting
  push ecx                  ; saving CX before overwriting
  push edx                  ; saving DX before overwriting

  mov eax, 0                ; stores final number
  mov ebx, 0
  mov ecx, 0                ; our index variable
  mov edx, dword [inputLength]     ; DX =
  dec edx                   ;     length - 1
```

```
 .loop: mul byte [ten]          ; AX = AX * 10d
      mov bl, byte [input + ecx] ; bl = ascii digit
      sub bl, 0x30            ; bl = digit
      add eax, ebx             ; AX = AX + digit
      inc ecx                ; set index pointer to next digit
      cmp ecx, edx             ; if index pointer points to last digit
      jle dec2AX.loop          ; loop if CX <= DX

 pop edx                     ; restoring DX after using it
 pop ecx                     ; restoring CX after using it
 pop ebx                     ; restoring BX after using it
 ret

; Converts EAX hex number to ascii decimal number string in [input] -> [input +
inputLength]
AX2dec:
 push ebx                    ; saving BX before overwriting
 push ecx                    ; saving CX before overwriting

 mov ebx, 0x0                ; count of digits pushed to stack
 mov ecx, 0x0                ; holds the digit before pushing to stack

 ; converting to ascii
 .loop: div byte [ten]         ; AL = AX/10, AH = AX%10
      mov cl, ah             ; CL = AH
      add cl, 0x30            ; AX%10 += ascii code for '0'
      push ecx               ; push the digit
      inc bl                ; count of digits in stack
      mov ah, 0              ; restore dividend
      cmp eax, 0x0           ; loop termination condition
      jne AX2dec.loop

 ; overwrite result over input
 mov ecx, 0
 mov dword [inputLength], ebx
 .loop2: pop dword [input + ecx] ; Take first digit from stack
       inc ecx              ; increment index pointer
       cmp ecx, ebx
       jl AX2dec.loop2        ; loop termination condition

 pop ecx                     ; restoring CX after using it
 pop ebx                     ; restoring BX after using it
 ret
```

```
section .bss
  inputLength resd 1
  input resb 100

section .data
  mEnterN db 'Enter a natural number N: '
  mEnterNLength equ $ - mEnterN

  mSum db 'Sum of first N natural numbers is: '
  mSumLength equ $ - mSum

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine

  ten db 0xA
```

Output

```
>make
nasm -f macho sumOfFirstN.asm
ld -o sumOfFirstN -e start sumOfFirstN.o
~/work/code/CollegePrograms/SystemProgramming/sumOfFirstN 04:43:49 AM
>./sumOfFirstN
Enter a natural number N: 10
Sum of first N natural numbers is: 55
~/work/code/CollegePrograms/SystemProgramming/sumOfFirstN 04:43:52 AM
>./sumOfFirstN
Enter a natural number N: 71
Sum of first N natural numbers is: 2556
```

Experiment 6

Code

```
%include "../macros.asm"

; Program to print sum of N
global start

section .text
start:
  ; Taking input N
  print mEnterN, mEnterNLength
  read input, 100
  dec eax               ; ignoring 0xA from length
  mov dword [inputLength], eax
  call dec2AX

  mov ecx, eax          ; count of numbers to process
  mov ebx, 0            ; sum of these numbers

  ; Inputing N numbers
  .loop: print mEnterNext, mEnterNextLength

        ; Taking ith number
          read input, 100
          dec eax
          mov dword [inputLength], eax

        call dec2AX    ; Getting ith number in AX

        add ebx, eax     ; Adding to the sum
        dec ecx         ; decrementing numbers to be read
        jnz start.loop    ; loop termination condition

  mov eax, ebx          ; saving result in AX
  call AX2dec         ; getting result in input

  ; Printing result
  print mSum, mSumLength
  print input, [inputLength]
  print mNewLine, mNewLineLength

  exit 0
```

```
; Converts ascii decimal number string [input] -> [input + inputLength] to hex number in EAX
dec2AX:
  push ebx                    ; saving BX before overwriting
  push ecx                    ; saving CX before overwriting
  push edx                    ; saving DX before overwriting

  mov eax, 0                  ; stores final number
  mov ebx, 0
  mov ecx, 0                  ; our index variable
  mov edx, dword [inputLength]     ; DX =
  dec edx                     ;     length - 1

  .loop: mul byte [ten]           ; AX = AX * 10d
       mov bl, byte [input + ecx] ; bl = ascii digit
       sub bl, 0x30            ; bl = digit
       add eax, ebx            ; AX = AX + digit
       inc ecx                 ; set index pointer to next digit
       cmp ecx, edx            ; if index pointer points to last digit
       jle dec2AX.loop         ; loop if CX <= DX

  pop edx                     ; restoring DX after using it
  pop ecx                     ; restoring CX after using it
  pop ebx                     ; restoring BX after using it
  ret

; Converts EAX hex number to ascii decimal number string in [input] -> [input + inputLength]
AX2dec:
  push ebx                    ; saving BX before overwriting
  push ecx                    ; saving CX before overwriting

  mov ebx, 0x0                ; count of digits pushed to stack
  mov ecx, 0x0                ; holds the digit before pushing to stack

  ; converting to ascii
  .loop: div byte [ten]           ; AL = AX/10, AH = AX%10
       mov cl, ah              ; CL = AH
       add cl, 0x30            ; AX%10 += ascii code for '0'
       push ecx                ; push the digit
       inc bl                  ; count of digits in stack
       mov ah, 0               ; restore dividend
       cmp eax, 0x0            ; loop termination condition
       jne AX2dec.loop
```

```
  ; overwrite result over input
  mov ecx, 0
  mov dword [inputLength], ebx
  .loop2: pop dword [input + ecx] ; Take first digit from stack
        inc ecx              ; increment index pointer
        cmp ecx, ebx
        jl AX2dec.loop2        ; loop termination condition

  pop ecx                    ; restoring CX after using it
  pop ebx                    ; restoring BX after using it
  ret

section .bss
  inputLength resd 1
  input resb 100
  temp resd 1

section .data
  mEnterN db 'Enter a number N: '
  mEnterNLength equ $ - mEnterN

  mEnterNext db 'Enter next positive number: '
  mEnterNextLength equ $ - mEnterNext

  mSum db 'Sum of above N numbers is: '
  mSumLength equ $ - mSum

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine

  ten db 0xA
```

Output

```
>make
nasm -f macho sumOfInput.asm
ld -o sumOfInput -e start sumOfInput.o
~/work/code/CollegePrograms/SystemProgramming/sumOfInput 04:46:46 AM
>./sumOfInput
Enter a number N: 4
Enter next positive number: 1
Enter next positive number: 2
Enter next positive number: 3
Enter next positive number: 4
Sum of above N numbers is: 10
```

Experiment 7

Code

```
%include "../macros.asm"

global start

section .text
start:
  print mUnsortedArray, mUnsortedArrayLength
  call printArray
  call bubbleSort
  print mSortedArray, mSortedArrayLength
  call printArray
  exit 0

bubbleSort:
  mov edx, dword [arrayLength]  ; DX contains length
  ; for( i = 0; i < length; i++ )
  mov eax, 0                    ; AX is i
  .outer cmp eax, edx
  jge bubbleSort.end
    ; for ( j = 1; j < length; j++ )
    mov ebx, 1                  ; BX is j
    .inner: cmp ebx, edx
    jge bubbleSort.outEnd
      mov ch, byte [array + ebx - 1d]
      mov cl, byte [array + ebx]
      ; if array[j - 1] > array[j]
      cmp ch, cl
      jle bubbleSort.inEnd
        ; swap a[j - 1] & a[j]
        mov byte [array + ebx], ch
        mov byte [array + ebx - 1], cl
      ; end of if
    ; end of inner loop
    .inEnd: inc ebx
    jmp bubbleSort.inner
  ; end of outer loop
  .outEnd: inc eax
  jmp bubbleSort.outer
.end ret
```

```
printArray:
  mov eax, 0                ; count of numbers seen yet
  mov edx, 0                ; count of numbers seen yet * 2
  mov ecx, dword [arrayLength]
  .loop: nop
        mov bl, byte [array + eax]
        add bl, 0x30
        mov byte [output + edx], bl
        mov byte [output + edx + 1], 32d
        add edx, 2
        inc eax
        cmp eax, ecx
        jl printArray.loop

  print output, edx
  print mNewLine, mNewLineLength
  ret

section .bss
  output resb 100

section .data
  ;array db 1, 2, 5, 6, 1, 2, 9, 7
  array db 9, 1, 6, 2, 3, 5, 2, 6
  ;array db 9, 8, 7, 6, 5, 3, 2, 1
  arrayLength dd 8d

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine

  mSortedArray db 'Sorted Array:   '
  mSortedArrayLength equ $ - mSortedArray

  mUnsortedArray db 'Unsorted Array: '
  mUnsortedArrayLength equ $ - mUnsortedArray
```

Output

```
>./bubbleSort
Unsorted Array: 9 8 7 6 5 3 2 1
Sorted Array:   1 2 3 5 6 7 8 9
>./bubbleSort
Unsorted Array: 9 1 6 2 3 5 2 6
Sorted Array:   1 2 2 3 5 6 6 9
```

Experiment 8

Code <MASM, DOS>

```
.model small
.data
   newline     db 13, 10, '$'
   anykey      db "Press [Enter] to exit...$"

.stack 256

.code
start:
   mov ax, @data
   mov ds, ax
   mov es, ax
   mov ax, @stack
   mov ss, ax

   mov ch, 0
   mov cl, 7
   mov ah, 1
   int 10h

   call exit_program

print_newline:
   lea dx, newline
   mov ah, 09h
   int 21h
   ret

exit_program:
   call print_newline

   lea dx, anykey
   mov ah, 09h
   int 21h

   mov ah, 01h
   int 21h

   ; exit to operating system.
   mov ah, 4ch
   int 21h

end start
```

Experiment 9

Code

```
%include "../macros.asm"

global start
; Program to take 4 digit hex input from keyboard and save it in [temp]

section .text
start:
  print mInput, mInputLength
  read input, 100
  dec eax
  mov dword [inputLength], eax

  call hex2AX

  mov dword [temp], eax
  mov eax, dword [temp]

  call AX2hex

  print mTempContents, mTempContentsLength
  print input, [inputLength]
  print mNewLine, mNewLineLength

  exit 0

; Converts ascii hex number string [input] -> [input + inputLength] to hex
number in EAX
hex2AX:
  push ebx                    ; saving BX before overwriting
  push ecx                    ; saving CX before overwriting
  push edx                    ; saving DX before overwriting

  mov eax, 0x0                  ; stores final number
  mov ebx, 0x0
  mov ecx, 0x0                  ; our index variable
  mov edx, dword [inputLength]     ; DX =
  dec edx                     ;     length - 1
```

```
 .loop: mul byte [sixteen]        ; AX = AX * 16d
      mov bl, byte [input + ecx] ; bl = ascii digit
      cmp bl, 64d               ; if bl < 'A'

      jge hex2AX.sub55
       sub bl, 48d            ;   bl = digit (0 - 9)
       jmp hex2AX.addBX

 .sub55 sub bl, 55d             ; else bl = digit (10 -15)
 .addBX add eax, ebx              ; AX = AX + digit

      inc ecx                ; set index pointer to next digit
      cmp ecx, edx             ; if index pointer points to last digit
      jle hex2AX.loop          ; loop if CX <= DX

 pop edx                     ; restoring DX after using it
 pop ecx                     ; restoring CX after using it
 pop ebx                     ; restoring BX after using it
 ret

; Converts EAX hex number to ascii hex number string in [input] -> [input +
[inputLength]]
AX2hex:
 push ebx                    ; saving BX before overwriting
 push ecx                    ; saving CX before overwriting

 mov ebx, 0x0                  ; count of digits pushed to stack
 mov ecx, 0x0                  ; holds the digit before pushing to stack

 ; converting to ascii
 .loop: mov edx, 0x0
      div word [sixteen]        ; AX = DXAX / 16d, DX = DXAX % 16d
      mov cl, dl              ; CL = DL
      cmp cl, 9d               ; if cl < 9
      jg AX2hex.addA
      add cl, 48d              ; AX % 16d += ascii code for '0'
      jmp AX2hex.pushCX
 .addA: add cl, 55d             ; AX % 16d += ascii code for 'A' - 10
.pushCX: push ecx                ; push the digit
      inc bl                ; count of digits in stack
      cmp eax, 0x0             ; loop termination condition
      jne AX2hex.loop
```

```
  ; overwrite result over input
  mov ecx, 0
  mov dword [inputLength], ebx
  .loop2: pop dword [input + ecx]   ; Take first digit from stack
        inc ecx                 ; increment index pointer
        cmp ecx, ebx
        jl AX2hex.loop2         ; loop termination condition

  pop ecx                       ; restoring CX after using it
  pop ebx                       ; restoring BX after using it
  ret

section .bss
  temp resd 1
  input resb 100
  inputLength resd 1

section .data
  mInput db 'Enter a 4 digit hex number: 0x'
  mInputLength equ $ - mInput

  mTempContents db '[temp] = 0x'
  mTempContentsLength equ $ - mTempContents

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine

  sixteen dw 16d
  ten dw 10d
```

Output

```
>./hexFromKeyboard
 Enter a 4 digit hex number: 0x01AF
 [temp] = 0x1AF
```

Experiment 10

Code

```
%include "../macros.asm"

; Program to print hex value in AX register
global start

section .text
start:
  mov eax, 0x0690

  call AX2hex

  print mAXContents, mAXContentsLength
  print input, [inputLength]
  print mNewLine, mNewLineLength

  exit 0

; Converts EAX hex number to ascii hex number string in [input] -> [input +
[inputLength]]
AX2hex:
  push ebx                  ; saving BX before overwriting
  push ecx                  ; saving CX before overwriting

  mov ebx, 0x0              ; count of digits pushed to stack
  mov ecx, 0x0              ; holds the digit before pushing to stack

  ; converting to ascii
  .loop: mov edx, 0x0
        div word [sixteen]      ; AX = DXAX / 16d, DX = DXAX % 16d
        mov cl, dl              ; CL = DL
        cmp cl, 9d              ; if cl < 9
        jg AX2hex.addA
        add cl, 48d             ; AX % 16d += ascii code for '0'
        jmp AX2hex.pushCX
  .addA: add cl, 55d            ; AX % 16d += ascii code for 'A' - 10
.pushCX: push ecx               ; push the digit
        inc bl                  ; count of digits in stack
        cmp eax, 0x0            ; loop termination condition
        jne AX2hex.loop

  ; overwrite result over input
  mov ecx, 0
  mov dword [inputLength], ebx
```

```
  .loop2: pop dword [input + ecx] ; Take first digit from stack
        inc ecx              ; increment index pointer
        cmp ecx, ebx
        jl AX2hex.loop2      ; loop termination condition

  pop ecx                    ; restoring CX after using it
  pop ebx                    ; restoring BX after using it
  ret

section .bss
  input resb 100
  inputLength resd 1

section .data
  mAXContents db 'AX = 0x'
  mAXContentsLength equ $ - mAXContents

  mNewLine db 0xA
  mNewLineLength equ $ - mNewLine

  sixteen dw 16d
```

Output

```
[>make
 nasm -f macho hexPrint.asm
 ld -o hexPrint -e start hexPrint.o
 ~/work/code/CollegePrograms/SystemProgramming/hexPrint 04:50:54 AM
[>./hexPrint
 AX = 0x690
[>./hexPrint
 AX = 0xB008
```