

Microprocessors ETI 2407

Assignment I

Mugo Mwaura ENE221-0280/2016

August 31, 2020

1 Question One

Write an assembly program that displays whole numbers and their squares. A user should input the last number. The output below shows what would happen if a user entered 5. What is the highest value you could enter that gave correct results? Explain why this number is the limit and what can be done to improve this limit:

x	x^2
1	1
2	4
3	9
4	16
5	25

1.1 Answer Explanation

Highest value entered that gave the correct result is 181 i.e., $\text{floor}\left(\sqrt{\frac{2^{16}}{2}}\right)$.

Highest number giving result = 2^{16} . Registers (AX) store 16 bit values. However, if this program was using a signed representation, the maximum would be:

$$\frac{2^{16}}{2} = 32,767$$

For a *32bit* answer, the result is placed in DX:AX or effectively, if using a single 32 bit register, EAX. This program does compute the answer upto a max of 65,536 therefore, but since we are only printing from AX (using

`emu8086 print_num` procedure), and haven't manually implemented printing from `DX:AX`, the maximum result we get is similar to that of a signed value.

What can be done to improve the limit? Print the entire result from `EAX` for a 2^{16} value. Beyond that, we can't go as the addresses only access *64KB* of addressable memory

1.2 Pseudo-code

1. Get maximum value x from user
2. initialize loop counter c to 1
3. **while** $c \leq x$
 - print c
 - print $c \times c$
 - **end loop**

1.3 Code

```
; Tab Character: 09
org 100h
include "emu8086.inc"

.code
    jmp start

start:
    ; Print input prompt to stdout
    mov AH, 9
    mov DX, offset msg_prompt
    int 21h

    ; Read number
    ; Stored in CX
    call scan_num

    ; Print column header to stdout
    mov AH, 9
    mov DX, offset col_header
    int 21h

    mov number, CX
```

```

    print_and_mul:
        call @print_new_line

        ; Multiple the number (X * X)
        call multiply_

    inc count ; count ++

    ; If count <= number, multiply and print
    cmp count, CX
    jle print_and_mul

    jmp @exit

; Multiplication procedure
multiply_ proc
    mov AX, count
    push AX ; Preserve AX. Calling @print_tab overwrites AX
    call print_num ; Print X

    call @print_tab ; Print tab

    pop AX
    MUL count ; X * X

    ; Print X^2 from AX
    call print_num
    ret
multiply_ endp

; Prints a new line
@print_new_line proc

    mov DL, 0xa
    mov AH, 2
    int 21h

    mov DL, 0xd
    mov AH, 2
    int 21h

```

```

        ret
@print_new_line endp

; Prints a tab character
@print_tab proc
    mov DL, 09
    mov AH, 2
    int 21h

    ret
@print_tab endp

.data
    msg_prompt db "Enter max value: ", 0xa, 0xd
    col_header db 0xa, 0xd, 0xa, 0xd, "x*****x**2", "$"
    number dw ?
    count dw 1

@exit:
    ret

    DEFINE_SCAN_NUM
    DEFINE_PRINT_NUM
    DEFINE_PRINT_NUM_UNS

    end

; Highest number giving result = 2^16
; Registers (AX) store 16 bit values. However, if this program was using a signed
; representation, the maximum would be  $(2^{16})/2 = 32,767$ 

; For a 32bit answer, the result is placed in DX:AX or effectively in EAX (32 bit)
; This program does compute the answer upto a max of 65,556 therefore
; but since we are only printing from AX, and haven't manually implemented
; printing from DX:AX, the maximum result we get is similar to that of assigned

; WHAT CAN BE DONE TO IMPROVE THE LIMIT?
; Print the entire result from EAX for a 2^16 value. Beyond that, we can't go as
; only access 64KB of addressible memory

```

1.4 Program Output

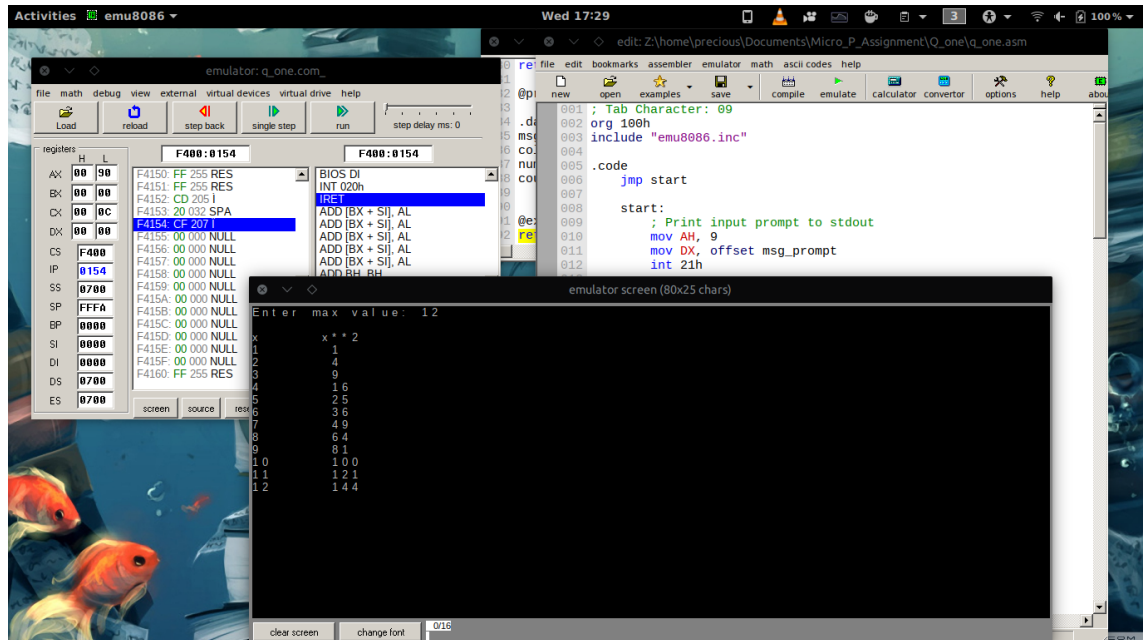


Figure 1: A screenshot of the output from running script 1.3

2 Question Two

Write a program that allows you convert from a number to the corresponding ASCII value.

E.g., if a user enters 35, the output should be #.

The program should also have a provision for displaying the ASCII value of an entered character.

E.g., if a user enters *, the output should be 42.

At the start of the program, a user should choose the option between the 2 modes (either ASCII value to character OR character to ASCII value). NB: Your code should be able to deal with erroneous input appropriately

E.g. entering a number above 255.

2.1 Psedo-code

1. Select program mode
 - a) ASCII to Decimal
 - b) Decimal to ASCII
2. Get input from user *input*
 - If mode (*b*) and $\text{input} > 255$, **error**
3. Initialize ASCII counter *c*. Initialize DEC counter *d*
4. **Start loop:**
 - If $\text{input} == \text{ASCII}_i$, (where *ASCII* is the set of all ASCII characters)
print ASCII_i , print DEC_i , (where *DEC* is an enumeration of the *ASCII* set)
end loop

2.2 Code

```
org 100h
include "emu8086.inc"

.code
    jmp start

start:
    ; Print input prompt to stdout
    mov AH, 9
    mov DX, offset msg_prompt
```

```

    int 21h

    ; Print choice menu
    mov AH, 9
    mov DX, offset choice_prompt
    int 21h

    ; Print choice_in final prompt
    mov AH, 9
    mov DX, offset choice_in
    int 21h

    ; Read number
    ; Stored in CX
    ; call scan_num

    ; Read user choice
    ; Char stored in AL
    mov AH, 01h
    int 21h

    ; This part was testing inc & printing of ascii chars
    ;mov CL, char

    ;do_it:
    ;    mov AH, 2
    ;    mov DL, CL
    ;    int 21h
    ;
    ;    mov AX, dec_counter
    ;    call print_num

    ;    inc dec_counter

    ;loop do_it

    call @process_choice

    jmp @exit

; Determine the selected user mode
@process_choice proc
    cmp AL, '1'
    je ascii_to_dec

```

```

    cmp AL, '2'
    je dec_to_ascii

    jmp unknown_entry

    ret
@process_choice endp

unknown_entry:
    mov AH, 09
    mov DX, offset unknown_prompt
    int 21h

    jmp @exit

ascii_to_dec:
    mov AH, 09
    mov DX, offset a_d_prompt
    int 21h

    ; read character
    ; Stored in AL
    mov AH, 1
    int 21h

    mov ascii_input, AL
    ; Convert ascii character to dec

    mov DL, 0 ; This is a flag used by the below callee
    call @get-equivalent-ascii

    jmp @exit

dec_to_ascii:
    mov AH, 09
    mov DX, offset d_to_a_prompt
    int 21h

    ; Read number
    ; Stored in CX
    call scan_num

    ; Beyond 255? Err
    cmp CX, 255

```



```

    jg illegal_dec

    ; Mov read value to variable
    ; CX [CL] will be used for loop
    mov dec_input, CX

    mov DL, 1 ; Flag used by below callee
    call @get_equivalent_ascii

    jmp @exit

; Prints out Error and halts process
; for decimal values beyond 255
illegal_dec:
    mov AH, 09
    mov DX, offset illegal_dec_p
    int 21h

    jmp @exit

; Prints a tab character
@print_tab proc
    mov DL, 09
    mov AH, 2
    int 21h

    ret

@print_tab endp

; Prints a new line
@print_new_line proc

    mov DL, 0xa
    mov AH, 2
    int 21h

    mov DL, 0xd
    mov AH, 2
    int 21h

    ret
@print_new_line endp

```

```

; Finds the ASCII equivalent of DECIMAL
; Loops through all ASCII characters
; O(n) Time complexity where n is 255
@get-equivalent-ascii proc

    mov BX, dec_counter ; For comparison, store addr in 16-bit reg
    mov CL, char ; ASCII 255 controls loop

do_it:
    ; DL = 0 -> ASCII to DEC, DL = 1 -> DEC to ASCII
    cmp DL, 0
    je cmp_character

    ; Check if BX matches input
    ; Yes? Break loop, print matching Char
    ; No? Try next character
    cmp BX, dec_input
    je print_answ

    jmp dec_step ; skip the cmp character part since we are doing DEC to ASCII

cmp_character:
    cmp CL, ascii_input
    je print_answ

dec_step:
    dec BX

loop do_it

print_answ:
    call @print_new_line
    ; Print the character the loop stopped at
    mov AH, 2
    mov DL, CL
    int 21h

    mov AH, 09
    mov DX, offset space_eq
    int 21h

    ; Print its DEC equivalent
    mov AX, BX
    call print_num

```

```

        ret
@get-equivalent_ascii endp

.data
msg_prompt db 0xa, 0xd, "Choose one option below [e.g. 1:] $"
unknown_prompt db 0xa, 0xd, "That's a strange choice dude/dudelady. GoodBye"
choice_prompt dw 0xa, 0xd, "1. ASCII to Decimal", 0xa, 0xd, "2. Decimal to A
choice_in db 0xa, 0xd, "Your choice: ", "$"
illegal_dec_p db 0xa, 0xd, 0xa, 0xd, "Oopsy! ASCII shouldn't exceed 255D$"
space_eq db "==$"

choice db ?
char db 255d
dec_counter dw 255d

; Stores user inputs
dec_input dw ?
ascii_input db ?

a_d_prompt db 0xa, 0xd, "Enter ascii character: $"
d_to_a_prompt db 0xa, 0xd, "Enter decimal value of character: $"

@exit:
    ret

DEFINE_SCAN_NUM
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS

end

```

2.3 Program Output

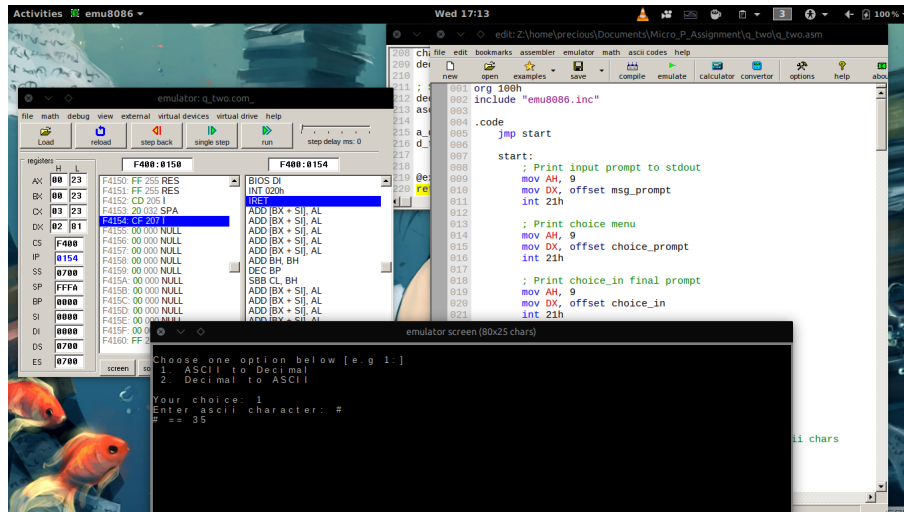


Figure 2: A screenshot of the output from running script 1.3. **Converting ASCII to decimal**

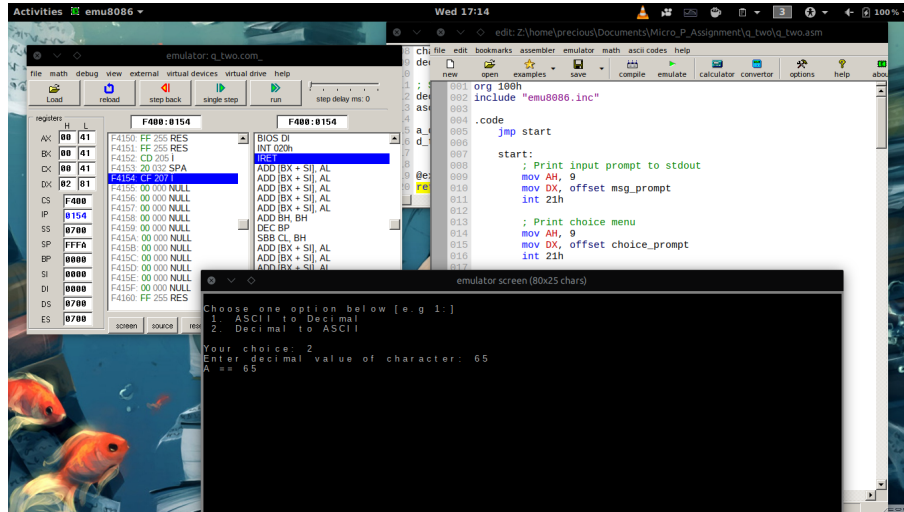


Figure 3: A screenshot of the output from running script 1.3. **Converting decimal to ASCII**

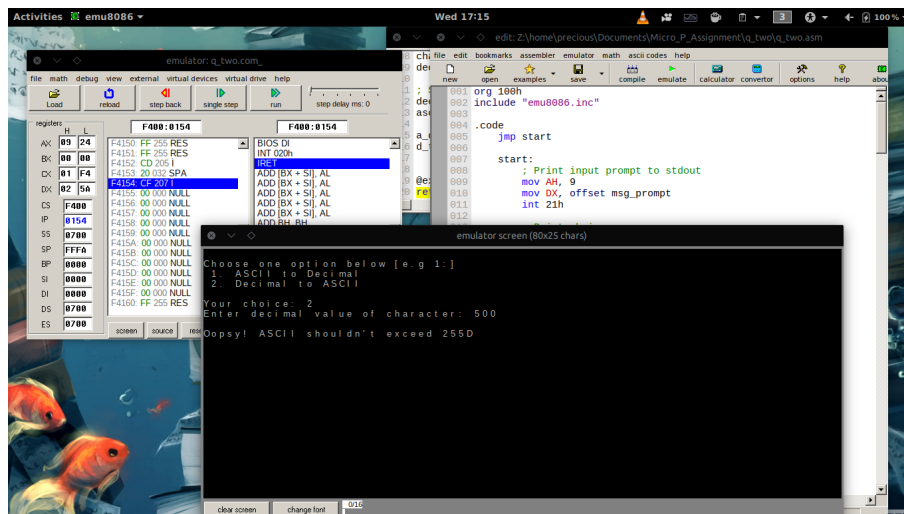


Figure 4: A screenshot of the output from running script 1.3. **Error on Decimal value greater than 255**

3 Question Three

Develop a virus that changes the letters being typed randomly into other letters and thus frustrates the user. E.g. if he intends to type *Hello boss*, it could instead display something like *Sdupl dwsp*. (For simplicity, assume that the email will be written into your running program. i.e., you do not need to write code for attacking the browser, **though if you do that, come get a cookie**).

After writing your code, describe what would happen if a user typed 1234?

3.1 Answer Explanation

If the user types "1234?" the program will output "1234?" to stdout (See Figure 5). The program only changes alphabetical ASCII inputs. i.e., 65 – 90 and 97 – 122. The rest of the inputs are printed as keyed in by the user.

3.2 Pseudo-code

1. **Loop:**
 - (a) Get character *c* from user without echo
 - **If** *c* is an alphabet:
 - Alter the value of *c* randomly
 - Print the value of *c*
 - Go back to (a)

3.3 Code

```
org 100h
include "emu8086.inc"

.code
    jmp start

start:
    ; Print input prompt
    mov AH, 09
    mov DX, offset prompt
    int 21h

    mov CX, 10000 ; Loop counter
get_input:
    ; Get std input
    mov AH, 07
    int 21h
```

```

; CTRL^C -> Terminate
cmp AL, 03
je terminate

; For Carriage returns, print a new line
cmp AL, 0xd
jne print_c

print_new_line_label:
    mov DL, 0xa
    mov AH, 2
    int 21h

    mov DL, 0xd
    mov AH, 2
    int 21h

print_c:
    ; Print character
    ; Stored in AL

    mov char, AL
    call @randomize_char

; Create infinite loop
; If CX = 1, set to to 10000 again
cmp CX, 1
jg continue_loop
mov CX, 10000

continue_loop:
    loop get_input

jmp @exit

@randomize_char proc
    mov AL, char
    cmp AL, 'A'
    jle not_alphabet

    cmp AL, 'Z'
    jg lower_case

    mov upper_b, 090
    mov lower_b, 065

```

```

jmp make_cmp

lower_case:
    cmp AL, 'a'
    jl not_alphabet

    cmp AL, 'z'
    jg not_alphabet

    mov upper_b, 122
    mov lower_b, 097

make_cmp:

    mov BL, upper_b
    mov DL, lower_b

    sub BL, DL

    sub upper_b, AL
    mov AL, upper_b
    ; div BL

    add AL, lower_b ; random (mod) BL + lower_b

    ; Avoid non chars between 090 and 097
    cmp AL, 090
    jle print_chr

check_if_char:
    cmp AL, 097
    jl add_char

    jmp print_chr

add_char:
    add AL, 20

print_chr:
    mov AH, 02
    mov DL, AL
    int 21h

```



```

        jmp end_rand

; Print non-alphabets without change
not_alphabet:
    mov AH, 02
    mov DL, AL
    int 21h

end_rand:
    ret
@randomize_char endp

; Terminates program
terminate:
    mov AH, 09
    lea DX, prompt_term
    int 21h

    jmp @exit

.data
    prompt db "Yo!_Start_typing_[Press_Control+C_to_terminate]", 0xa, 0xd, 0xa,
    prompt_term db 0xa, 0xd, 0xa, 0xd, 09,
        "**_Process_terminated_by_user._Bye_dude/dudelady_**" , 0xa, 0xd, "$"
    char db ?
    upper_b db ?
    lower_b db ?

@exit:
    ret
end

```

3.4 Program Output

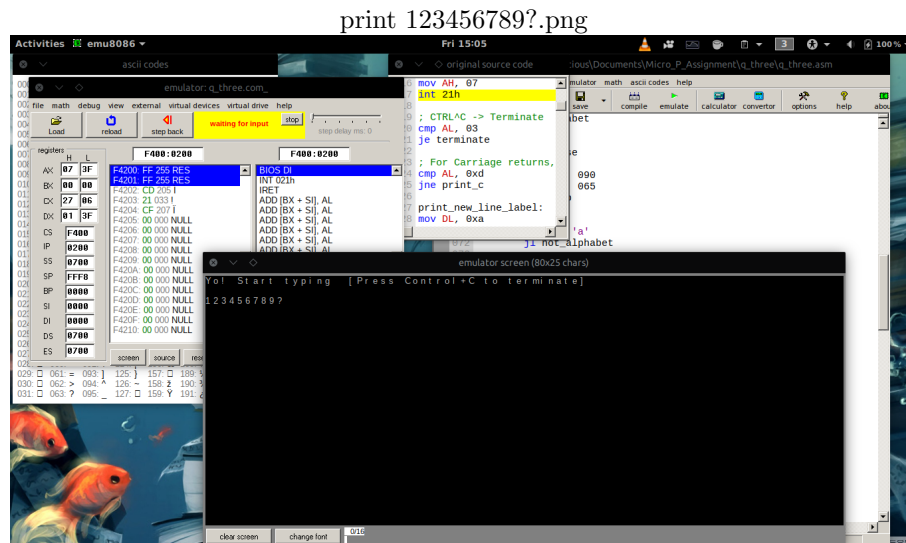


Figure 5: A screenshot of the output from running script 3.3. An input of "1234?" outputs "1234?"

a noisy machine.png

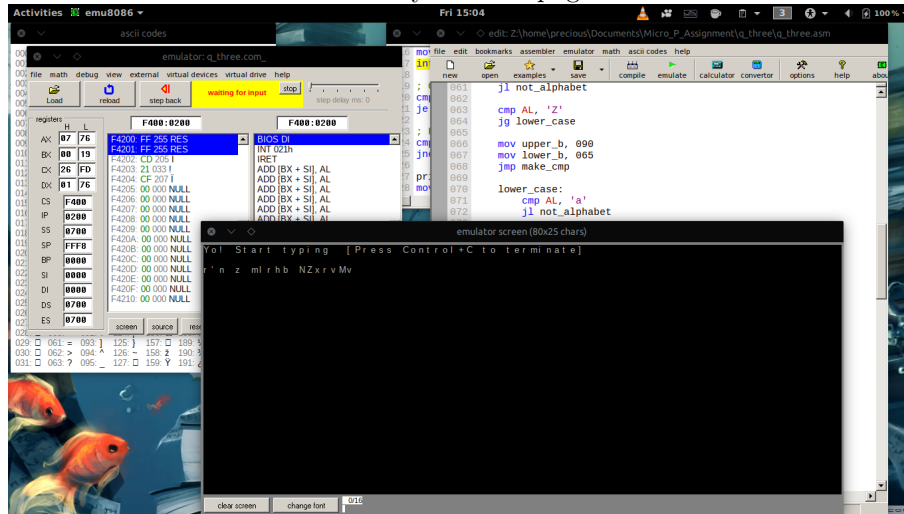


Figure 6: A screenshot of the output from running script 3.3. The output of the input "I'm a noisy machine"

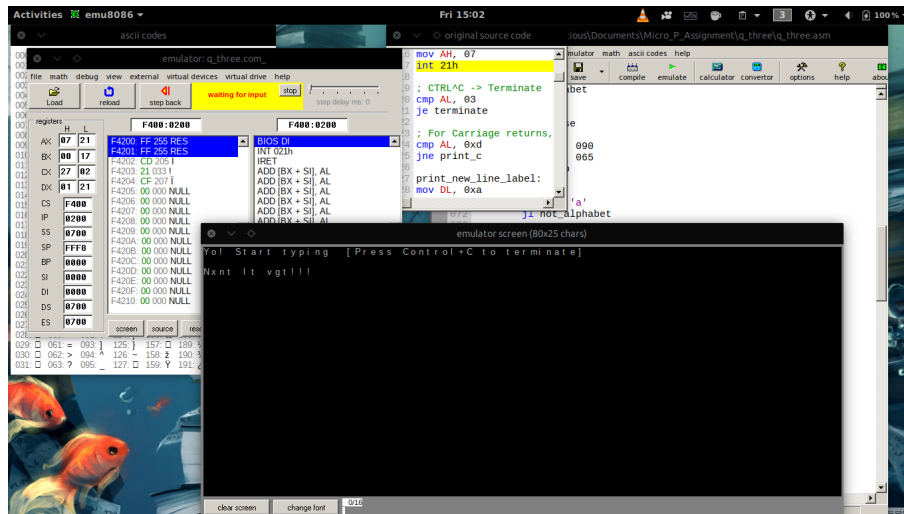


Figure 7: A screenshot of the output from running script 3.3. The output of the input "make me cry!!!"