System Hardware
COMP 228/4 Section DD
Winter 2024
Assignment 2

Date Assigned: Thursday, April 4, 2024, 11:30 PM
Date Due: Sunday, April 21, 2024, 11:59 PM

Write a complete Intel x86 assembly language program which determines whether a byte sized operand stored in memory location `number` is prime or not. The program will write the value of 0 into the byte sized memory location `answer` if the number is **not** prime, otherwise the initial value of 1 will be left unmodified (indicating that the number is prime). The program is to make use of the `div` instruction to determine the value of quotient and remainder when dividing the number by 2,3,4,... (number -1) .

The program is to implement the following high-level pseudocode (the student is free to modify this pseudocode) :

```
prime = true ;
divisor = 2 ;
while ( divisor < number )
{
   if ( the remainder of number / divisor is 0 )
   {
      prime = false ; // we found a divisor which
                      //evenly divides number
   }                  // so therefore number
                      //cannot be prime

   divisor = divisor + 1 ;  // check the next divisor
                            // and keep looping
}

if (the number is prime)
{
  Display the message “Number is prime”
}
else
{
  Display the message “Number is NOT prime”
}
```

You are to make use of the following `section .data` within your program:

```
section .data

number db  5   ; you should check that your program
               ; works with different values
answer db  1   ; 1 means number is prime,
               ; 0 means number is not prime

; ASCII 0x0a = LINE FEED (for new line)
prime_msg db 'Number is prime', 0x0a
not_prime_msg db 'Number is NOT prime', 0x0a
```

Your assembly language program **MUST** include at two subroutines which makes use of the `int 80h` set of system calls to perform output of character strings. One subroutine is to display the `Number is prime` message and another subroutine will be used to display the `Number is NOT prime` message.

Some useful Intel x86 assembly language insructions

It may be helpful to make use of the following Intel x86 assembly language instructions within your program:

```
div bl  ;  performs ax / bl  with quotient in al
        ;  and remainder  in ah

and ax, 1111111100000000b  ; performs logical and of ax
                           ; with specified immediate data
                           ; ax <-- ax AND immediate data
                           ; the 1111111100000000b is a MASK
                           ; used to isolate specific bits
                           ; we learned about the use of masks
```

Life-long learning

As life-long learning is one of the listed graduate attributes as identified by Canadian Information Processing Society (CIPS), this programming assignment has designed such that students are expected to search for and located relevant online reference material in order to complete the assignment. As such, not all of the details of certain aspects of writing Intel x86 assembly languages programs have been presented in class. Students are to use online resources to familiarize themselves with:

• using the `int 80h` set of Linux system calls to display a message

• where to place the assembly language code for one or more subroutines within the `section .text` portion of the source code.

• the exact syntax of the Intel x86 assembly language instructions used to call and return from a subroutine.

Some good online starting points are:

```
/home/t/ted/PUBLIC/COMP228/Sample_NASM_Programs/NASM_PRIMER.txt

/home/t/ted/PUBLIC/COMP228/Sample_NASM_Programs/
How_to_perform_input_output_using_int80_system_call.txt

/home/t/ted/PUBLIC/COMP228/Sample_NASM_Programs/
max_subroutine.asm

/home/t/ted/PUBLIC/COMP228/NOTES/PDF/intro_to_x86_asm.pdf
```

Other online tools include Google and ChatGPT. The use of ChatGPT is only allowed for learning how to use a particular assembly language instruction or other syntactical issues. It is strictly prohibited to use any AI tools for the purposes of completely writing the assembly language code required for this assignment.

Group Work

Students shall form groups of 3-4 members. Each group shall determine the division of labor within a group.

Assignment submission

Assignment submission will be by Moodle. Two files are to be submitted: the assembly language source code with filename **prime.asm** and the executable file with filename **prime** . There shall be one designated group leader, this person shall submit the two required files for the entire group. **The full names and student ID numbers of all the group members are to be included as comments placed in the first lines of the prime.asm source code file.** For example:

```
; Group Members
; Keith Richards  ID 19691969
; Mick Jagger  ID 12345678
; Ron Wood  ID 99999999
```

You are to use the Linux `nasm`, `ld`, and `gdb` tools available on the GCS Linux computers to develop and test your program. It is strongly suggested that you test your program with two dif-

ferent values stored in number: one which is prime and another which is not prime. Typical user interaction when running the executable from the Linux command prompt would be similar to:

```
ted@deadflowers NASM 10:10pm > prime
Number is prime
ted@deadflowers NASM 10:36pm >
```

Note: Depending upon how your Linux account has been setup by the AITS system administrators, it may be necessary to prepend a `./` (commonly referred to as ‘dot slash’ in the parlance of Linux) in front of the name of the executable:

```
ted@deadflowers NASM 10:36pm >./prime
Number is prime
ted@deadflowers NASM 10:36pm >
```

Topic for life-long learning: Enter in Google the search string “Why do we use dot slash?”

NOTE: If your prime.asm file does not assemble, a grade of zero shall be given.
Does not assemble, in this context means that the `nasm` assembler generates one or more error messages. For example:

```
ted@deadflowers NASM 10:48pm >nasm -f elf move_eip.asm -l
move_eip.lst
move_eip.asm:20: error: symbol ‘eip’ not defined
```