ANIKA HAQUE, 163403

San Francisco Bay University

EE488 - Computer Architecture
Homework Assignment #4

Due day: 3/27/2025

Instructions:

1. **The homework answer sheet should contain the original questions and corresponding answers.**
2. **The answer sheet must be in MS-Word file format with Github links for the programming questions. As follows is the answer sheet name format.** ***<course_id>_week<week_number>_StudentID_FirstName_LastName.pdf***
3. **The program name in Github must follow the format like** ***<course_id>_week<week_number>_q<question_number>_StudentID_FirstName_LastName***
4. **Show screenshot of all running results, including the system date/time.**
5. **The calculation process must be typed if needed, handwriting can't be accepted.**
6. **Only accept homework submission uploaded via Canvas.**
7. **Overdue homework submission can't be accepted.**
8. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. Implement the following subroutine function in the *utils.asm* file, properly documenting them, and include programs to test them.

    a. *Mult10* - take an input parameter and return that parameter multiplied by *10* using ONLY shift and add operations.

    b. *ToUpper* - take a *32*-bits input which is *3* characters and a null, or a *3*-characters string. Convert the *3* characters to upper case if they are lower case or do nothing if they are already upper case.

    c. *ToLower* - take a *32*-bits input which is 3 characters and a null, or a *3*-characters string. Convert the *3* characters to lower case if they are upper case or do nothing if they are already lower case.

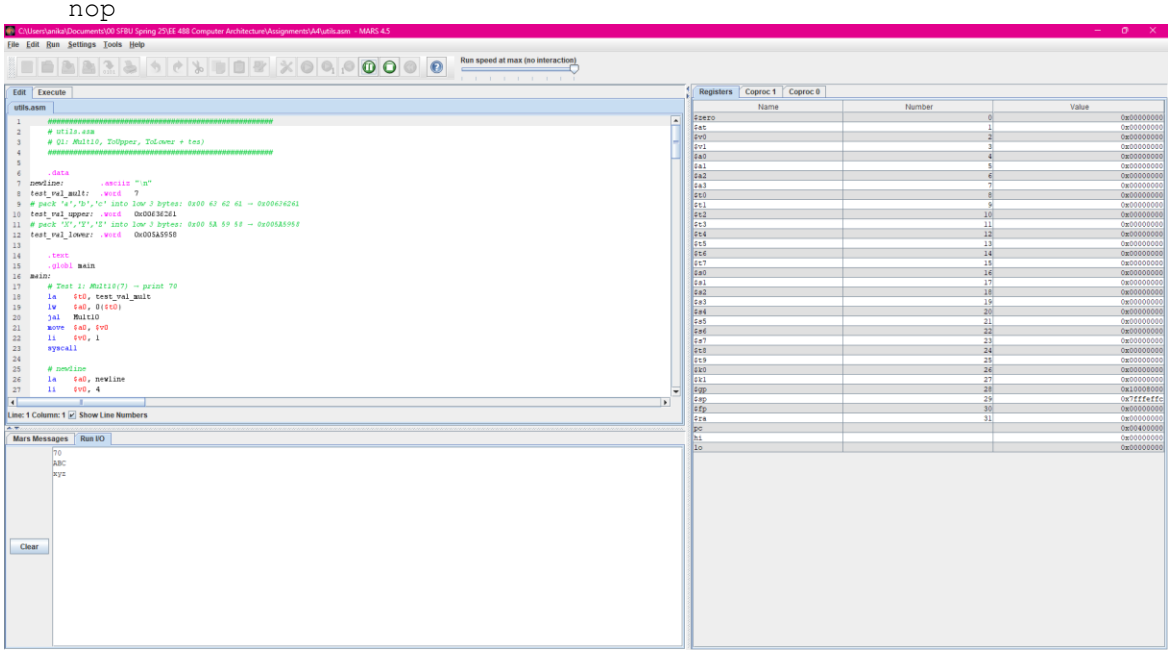
```
#####################################################
# utils.asm
# Q1: Mult10, ToUpper, ToLower + tes)
#####################################################

    .data
newline:        .asciiz "\n"
test_val_mult:  .word   7
```

```
# pack 'a','b','c' into low 3 bytes: 0x00 63 62 61 → 0x00636261
test_val_upper: .word   0x00636261
# pack 'X','Y','Z' into low 3 bytes: 0x00 5A 59 58 → 0x005A5958
test_val_lower: .word   0x005A5958

    .text
    .globl main
main:
    # Test 1: Mult10(7) → print 70
    la    $t0, test_val_mult
    lw    $a0, 0($t0)
    jal   Mult10
    move  $a0, $v0
    li    $v0, 1
    syscall

    # newline
    la    $a0, newline
    li    $v0, 4
    syscall

    # Test 2: ToUpper("abc") → print "ABC"
    la    $t0, test_val_upper
    lw    $a0, 0($t0)
    jal   ToUpper
    move  $t2, $v0

    # print byte 0
    andi  $t1, $t2, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # print byte 1
    srl   $t1, $t2, 8
    andi  $t1, $t1, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # print byte 2
    srl   $t1, $t2, 16
    andi  $t1, $t1, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # newline (ASCII 10)
    li    $a0, 10
    li    $v0, 11
    syscall

    # Test 3: ToLower("XYZ") → print "xyz"
    la    $t0, test_val_lower
    lw    $a0, 0($t0)
    jal   ToLower
    move  $t2, $v0

    # print byte 0
```

```
    andi  $t1, $t2, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # print byte 1
    srl   $t1, $t2, 8
    andi  $t1, $t1, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # print byte 2
    srl   $t1, $t2, 16
    andi  $t1, $t1, 0xFF
    move  $a0, $t1
    li    $v0, 11
    syscall

    # newline
    li    $a0, 10
    li    $v0, 11
    syscall

hang:
    j     hang
    nop

# ------------------------------------------------------------
# Mult10:
#   Multiply $a0 by 10 → $v0 using only shifts & adds
#   Clobbers: $t0, $t1
# ------------------------------------------------------------
Mult10:
    sll   $t0, $a0, 3
    sll   $t1, $a0, 1
    add   $v0, $t0, $t1
    jr    $ra
    nop

# ------------------------------------------------------------
# ToUpper:
#   Lowercase→uppercase for up to three ASCII bytes in $a0 → $v0
#   Clobbers: $t0-$t4
# ------------------------------------------------------------
ToUpper:
    move  $t2, $a0
    li    $t3, 0x20

    # byte 0
    andi  $t0, $t2, 0xFF
    li    $t1, 'a'
    li    $t4, 'z'
    blt   $t0, $t1, U0skip
    bgt   $t0, $t4, U0skip
    sub   $t0, $t0, $t3
U0skip:
    andi  $t2, $t2, 0xFFFFFF00
    or    $t2, $t2, $t0
```

```
    # byte 1
    srl   $t0, $a0, 8
    andi  $t0, $t0, 0xFF
    li    $t1, 'a'
    li    $t4, 'z'
    blt   $t0, $t1, U1skip
    bgt   $t0, $t4, U1skip
    sub   $t0, $t0, $t3
U1skip:
    sll   $t0, $t0, 8
    andi  $t2, $t2, 0xFFFF00FF
    or    $t2, $t2, $t0

    # byte 2
    srl   $t0, $a0, 16
    andi  $t0, $t0, 0xFF
    li    $t1, 'a'
    li    $t4, 'z'
    blt   $t0, $t1, U2skip
    bgt   $t0, $t4, U2skip
    sub   $t0, $t0, $t3
U2skip:
    sll   $t0, $t0, 16
    andi  $t2, $t2, 0xFF00FFFF
    or    $t2, $t2, $t0

    move  $v0, $t2
    jr    $ra
    nop

# ------------------------------------------------------------
# ToLower:
#   Uppercase→lowercase for up to three ASCII bytes in $a0 → $v0
#   Clobbers: $t0-$t4
# ------------------------------------------------------------
ToLower:
    move  $t2, $a0
    li    $t3, 0x20

    # byte 0
    andi  $t0, $t2, 0xFF
    li    $t1, 'A'
    li    $t4, 'Z'
    blt   $t0, $t1, L0skip
    bgt   $t0, $t4, L0skip
    add   $t0, $t0, $t3
L0skip:
    andi  $t2, $t2, 0xFFFFFF00
    or    $t2, $t2, $t0

    # byte 1
    srl   $t0, $a0, 8
    andi  $t0, $t0, 0xFF
    li    $t1, 'A'
    li    $t4, 'Z'
    blt   $t0, $t1, L1skip
    bgt   $t0, $t4, L1skip
    add   $t0, $t0, $t3
```

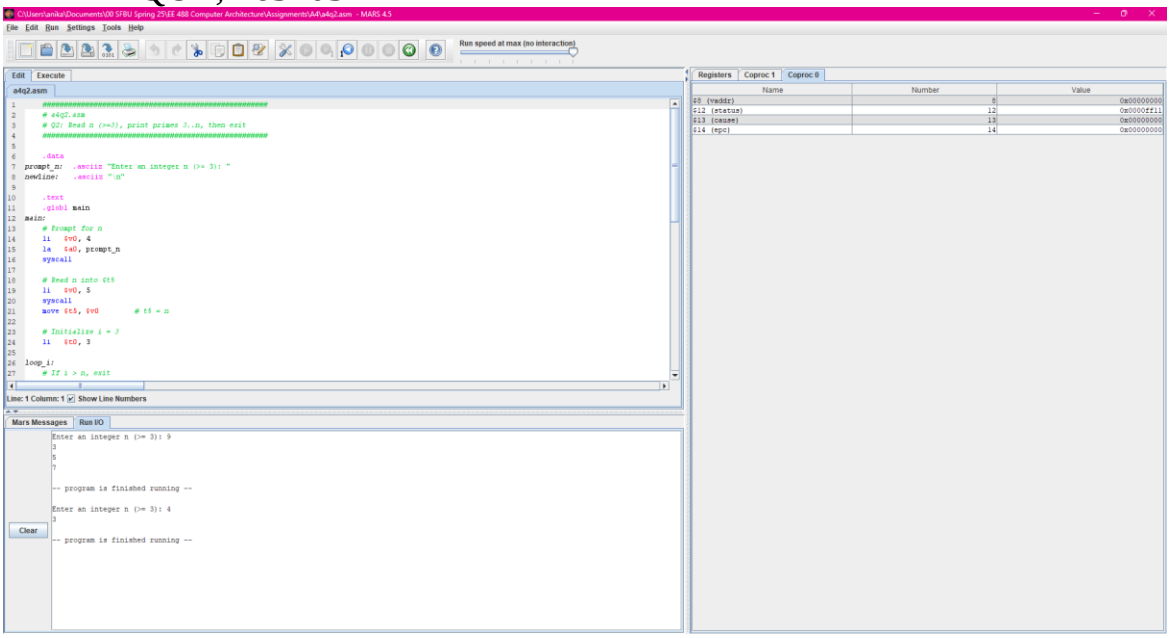
```
L1skip:
    sll   $t0, $t0, 8
    andi  $t2, $t2, 0xFFFF00FF
    or    $t2, $t2, $t0

    # byte 2
    srl   $t0, $a0, 16
    andi  $t0, $t0, 0xFF
    li    $t1, 'A'
    li    $t4, 'Z'
    blt   $t0, $t1, L2skip
    bgt   $t0, $t4, L2skip
    add   $t0, $t0, $t3
L2skip:
    sll   $t0, $t0, 16
    andi  $t2, $t2, 0xFF00FFFF
    or    $t2, $t2, $t0

    move  $v0, $t2
    jr    $ra
    nop
```



2. Write a program to find prime numbers from *3* to *n* in a loop in MIPS assembly.

```
    ######################################################
    # a4q2.asm
    # Q2: Read n (>=3), print primes 3..n, then exit
    ######################################################

    .data
prompt_n:  .asciiz "Enter an integer n (>= 3): "
newline:   .asciiz "\n"

    .text
    .globl main
main:
    # Prompt for n
    li   $v0, 4
    la   $a0, prompt_n
    syscall
```

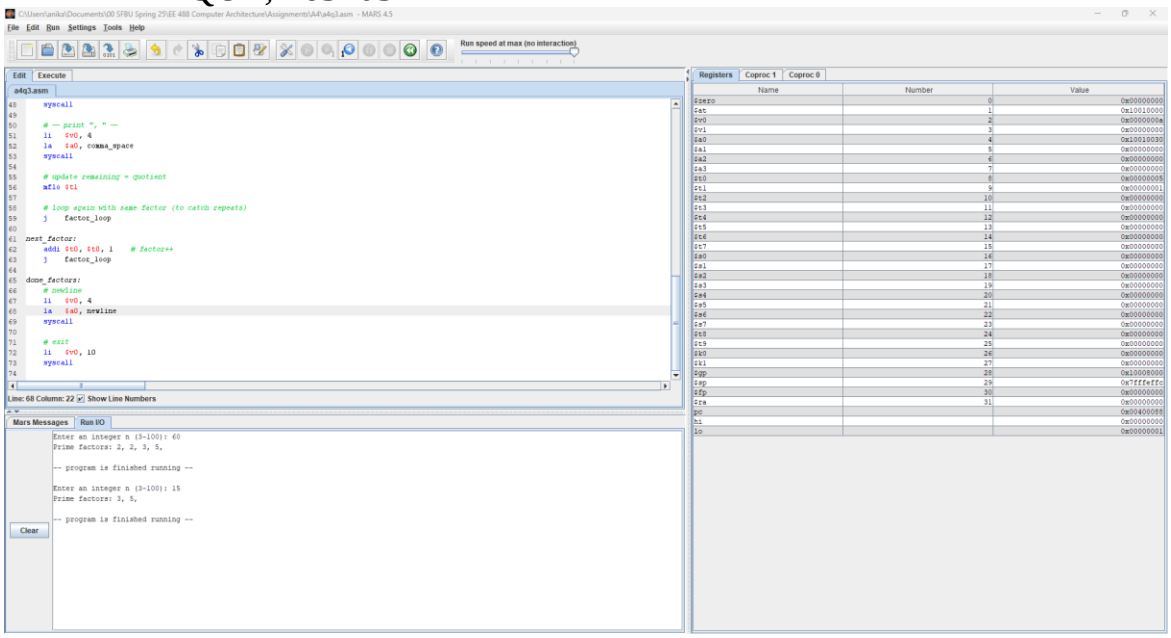
```
    # Read n into $t5
    li   $v0, 5
    syscall
    move $t5, $v0        # t5 = n

    # Initialize i = 3
    li   $t0, 3

loop_i:
    # If i > n, exit
    bgt  $t0, $t5, exit

    # Test divisors j = 2..√i
    li   $t1, 2          # j = 2

test_j:
    mul  $t2, $t1, $t1   # t2 = j * j
    bgt  $t2, $t0, is_prime  # if j^2 > i, it's prime

    div  $t0, $t1        # divide i by j
    mfhi $t3             # t3 = remainder
    beq  $t3, $zero, next_i # divisible → not prime

    addi $t1, $t1, 1     # j++
    j    test_j

is_prime:
    # Print i
    li   $v0, 1
    move $a0, $t0
    syscall
    # Print newline
    li   $v0, 4
    la   $a0, newline
    syscall

next_i:
    addi $t0, $t0, 1     # i++
    j    loop_i

exit:
    # Clean exit
    li   $v0, 10
    syscall
```



3. Prompt the user for a number from *3…100* and determine the prime factors for that number. For example, *15* has prime factors *3* and *5*. *60* has prime factors *2*, *3*, and *5*. You ONLY have to print out the prime factors.

```
    ####################################################
    # a4q3.asm
    # Q3: Read n (3-100), print its prime factors (with repeats),
    #     then exit.
    ####################################################

    .data
prompt_n:     .asciiz "Enter an integer n (3-100): "
prompt_fact:  .asciiz "Prime factors: "
comma_space:  .asciiz ", "
newline:      .asciiz "\n"

    .text
    .globl main
main:
    # — Prompt for n —
    li   $v0, 4
    la   $a0, prompt_n
    syscall

    # — Read n into $t1 —
    li   $v0, 5
    syscall
    move $t1, $v0        # t1 = n

    # — Print header —
    li   $v0, 4
    la   $a0, prompt_fact
    syscall

    # — Initialize factor = 2 —
    li   $t0, 2

factor_loop:
```

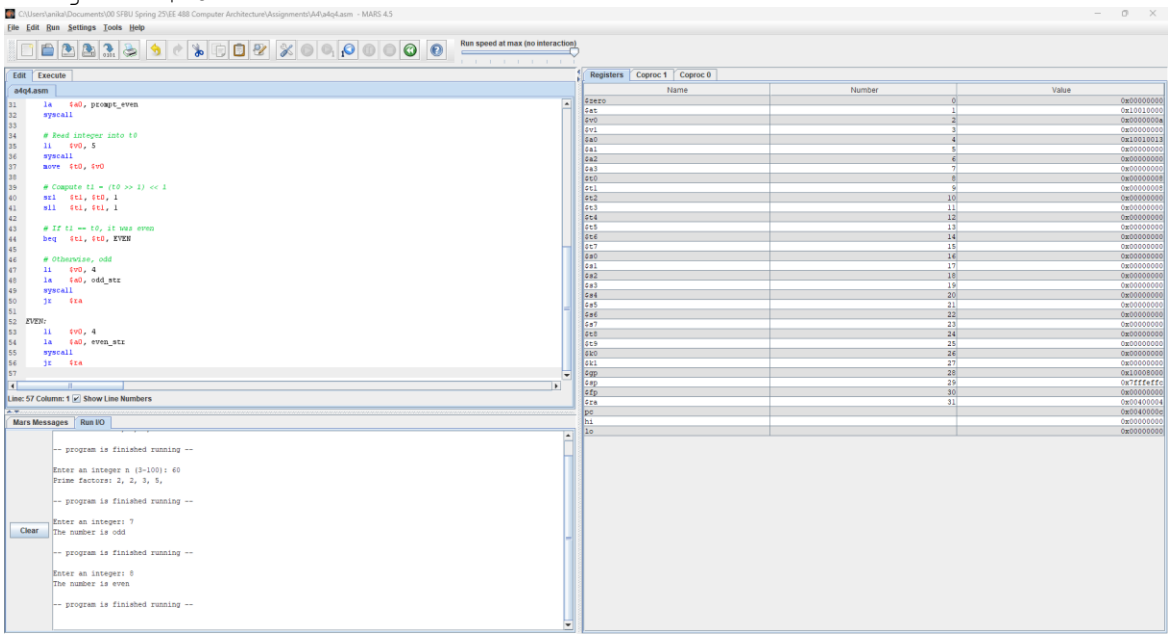
```
    # if factor > remaining (t1), we're done
    bgt  $t0, $t1, done_factors

    # divide remaining by factor
    div  $t1, $t0
    mfhi $t2            # t2 = remainder

    # if not divisible, bump factor
    bne  $t2, $zero, next_factor

    # — divisible: print factor —
    li   $v0, 1
    move $a0, $t0
    syscall

    # — print ", " —
    li   $v0, 4
    la   $a0, comma_space
    syscall

    # update remaining = quotient
    mflo $t1

    # loop again with same factor (to catch repeats)
    j    factor_loop

next_factor:
    addi $t0, $t0, 1    # factor++
    j    factor_loop

done_factors:
    # newline
    li   $v0, 4
    la   $a0, newline
    syscall

    # exit
    li   $v0, 10
    syscall
```



4. Using only *sll* and *srl*, implement a program to check if a user input value is even or odd. The program should read a user input integer and print out "The number is even" if the number is even, or "The number is odd", if the number is odd.

```
    ######################################################
    # a4q4.asm
    # Q4: Read an integer and print “even” or “odd”
    #       using only srl/sll for the test.
    ######################################################

    .data
prompt_even:  .asciiz "Enter an integer: "
even_str:     .asciiz "The number is even\n"
odd_str:      .asciiz "The number is odd\n"

    .text
    .globl main
main:
    # Call the even/odd tester
    jal   even_odd

    # Exit program
    li    $v0, 10
    syscall

# ---------------------------------------------------------------
# even_odd:
#   Prompts for an integer, reads it, and prints
#   even_str if it’s even, odd_str if it’s odd.
#   Only uses srl/sll to test the low bit.
# ---------------------------------------------------------------
even_odd:
    # Print prompt
    li    $v0, 4
    la    $a0, prompt_even
    syscall

    # Read integer into t0
```

```
    li     $v0, 5
    syscall
    move   $t0, $v0

    # Compute t1 = (t0 >> 1) << 1
    srl    $t1, $t0, 1
    sll    $t1, $t1, 1

    # If t1 == t0, it was even
    beq    $t1, $t0, EVEN

    # Otherwise, odd
    li     $v0, 4
    la     $a0, odd_str
    syscall
    jr     $ra

EVEN:
    li     $v0, 4
    la     $a0, even_str
    syscall
    jr     $ra
```



5. Prompt the user for a number *n*, *0 < n < 100*. Print out the smallest number of coins (quarters, dimes, nickels, and pennies) which will produce *n*. For example, if the user enters *"66"*, your program should print out "*2* quarters, *1* dime, *1* nickel, and *1* penny".

```
    ######################################################
    # a4q5.asm
    # Q5: Read an amount (1-99 cents), compute change in
    #     quarters, dimes, nickels, pennies, then exit.
    ######################################################

    .data
prompt_amt:   .asciiz "Enter amount in cents (1-99): "
quarter_lbl:  .asciiz " quarters, "
dime_lbl:     .asciiz " dimes, "
nickel_lbl:   .asciiz " nickels, "
```

```
penny_lbl:    .asciiz " pennies\n"

    .text
    .globl main
main:
    # Call the coin_change routine
    jal  coin_change

    # Exit program
    li   $v0, 10
    syscall

# ------------------------------------------------------------
# coin_change:
#   Prompts for an amount in cents, reads it, then computes
#   the number of quarters, dimes, nickels, and pennies, and
#   prints each count with its label.
#   Uses only div/mflo/mfhi for arithmetic.
# ------------------------------------------------------------
coin_change:
    # Prompt the user
    li   $v0, 4
    la   $a0, prompt_amt
    syscall

    # Read the amount into t0
    li   $v0, 5
    syscall
    move $t0, $v0        # t0 = amount

    # Compute quarters
    li   $t1, 25
    div  $t0, $t1
    mflo $t2             # t2 = #quarters
    mfhi $t0             # t0 = remainder

    # Compute dimes
    li   $t1, 10
    div  $t0, $t1
    mflo $t3             # t3 = #dimes
    mfhi $t0             # t0 = remainder

    # Compute nickels
    li   $t1, 5
    div  $t0, $t1
    mflo $t4             # t4 = #nickels
    mfhi $t0             # t0 = remainder

    # Remaining pennies
    move $t5, $t0        # t5 = #pennies

    # Print quarters
    li   $v0, 1
    move $a0, $t2
    syscall
    li   $v0, 4
    la   $a0, quarter_lbl
    syscall
```

```
    # Print dimes
    li   $v0, 1
    move $a0, $t3
    syscall
    li   $v0, 4
    la   $a0, dime_lbl
    syscall

    # Print nickels
    li   $v0, 1
    move $a0, $t4
    syscall
    li   $v0, 4
    la   $a0, nickel_lbl
    syscall

    # Print pennies
    li   $v0, 1
    move $a0, $t5
    syscall
    li   $v0, 4
    la   $a0, penny_lbl
    syscall

    jr   $ra
    nop
```