



San Francisco Bay University

EE488 - Computer Architecture Homework Assignment #5

Due day: 4/4/2025

Instruction:

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 a subprogram that prompt the user for 3 numbers, finds the median (middle value) of the 3, and returns that value to the calling program.

```
.data
prompt1: .ascii "Enter first number: "
prompt2: .ascii "Enter second number: "
prompt3: .ascii "Enter third number: "
result:  .ascii "Median is: "
newline: .ascii "\n"

.text
.globl main

main:
    # --- Read a, b, c into $s0, $s1, $s2 ---
    li    $v0, 4
    la    $a0, prompt1
    syscall
    li    $v0, 5
    syscall
    move  $s0, $v0          # s0 = a

    li    $v0, 4
    la    $a0, prompt2
    syscall
    li    $v0, 5
    syscall
```

ANIKA HAQUE, 163403

```
    move $s1, $v0          # s1 = b

    li    $v0, 4
    la    $a0, prompt3
    syscall
    li    $v0, 5
    syscall
    move  $s2, $v0          # s2 = c

    # --- Compute sum = a + b + c in $t4 ---
    add   $t4, $s0, $s1
    add   $t4, $t4, $s2

    # --- Find max(a,b) into $t0 ---
    slt   $t1, $s0, $s1     # t1 = 1 if a < b
    bne   $t1, $zero, L1    # if a<b, go to L1 (b is larger)
    move  $t0, $s0           # else a ≥ b → t0 = a
    j     L2
L1:      move $t0, $s1       # t0 = b
L2:      # t0 now holds max(a,b)
    slt   $t1, $t0, $s2     # t1 = 1 if max(a,b) < c
    bne   $t1, $zero, L3    # if so, c is the overall max
    # else max(a,b) ≥ c
    # t0 already max(a,b)
    j     GotMax
L3:      move $t0, $s2       # c is the overall max
GotMax:  # --- Now t0 = M = max(a,b,c) ---

    # --- Find min(a,b) into $t1 ---
    slt   $t2, $s1, $s0     # t2 = 1 if b < a
    bne   $t2, $zero, L4    # if b<a, go to L4
    move  $t1, $s0           # else a ≤ b → t1 = a
    j     L5
L4:      move $t1, $s1       # t1 = b
L5:      # t1 now holds min(a,b)
    slt   $t2, $s2, $t1     # t2 = 1 if c < min(a,b)
    bne   $t2, $zero, L6    # if so, c is the overall min
    # else min(a,b) ≤ c
    j     GotMin
L6:      move $t1, $s2       # c is the overall min
GotMin:  # --- Now t1 = m = min(a,b,c) ---

    # --- median = sum - max - min → $t2 ---
    sub   $t2, $t4, $t0     # t2 = sum - max
    sub   $t2, $t2, $t1     # t2 = (a+b+c) - max - min

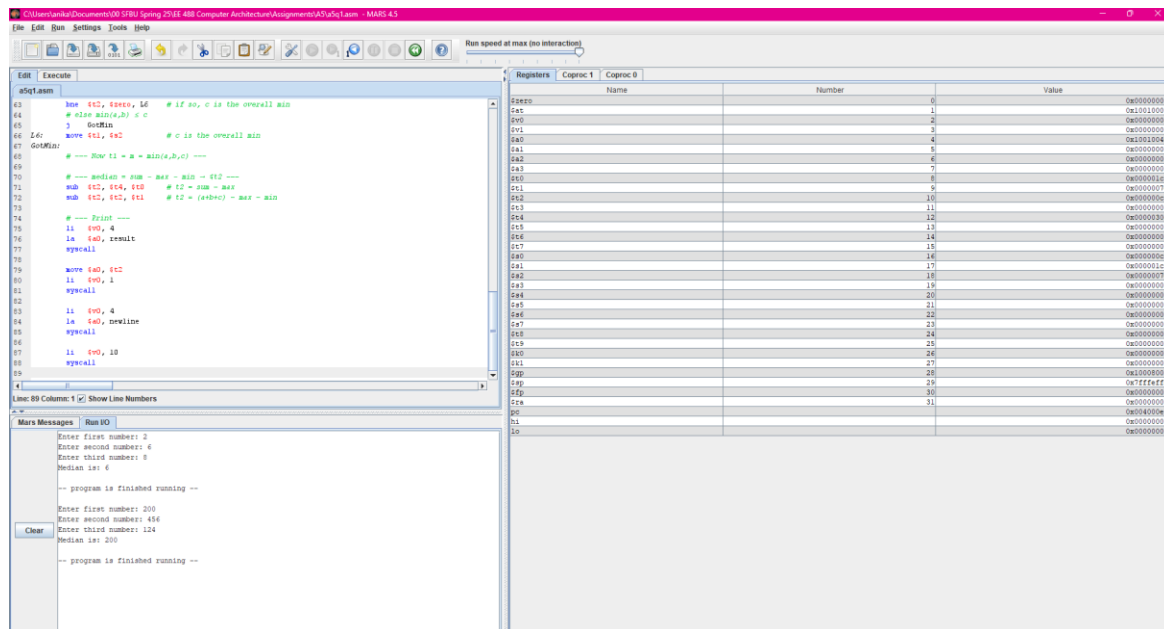
    # --- Print ---
    li    $v0, 4
    la    $a0, result
    syscall

    move  $a0, $t2
    li    $v0, 1
    syscall
```

ANIKA HAQUE, 163403

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

li    $v0, 10
syscall
```



2. Implement a **recursive** program that takes in a number and finds the square of that number through addition. For example if the number 3 is entered, you would add $3+3+3=9$. If 4 is entered, you would add $4+4+4+4=16$. This program must be implemented using **recursion** to add the numbers together.

```

.data
prompt:  .ascii "Enter n: "
result_msg: .ascii "Square is: "
newline:  .ascii "\n"

```

```
.text
.globl main
```

```
main:
    # --- Prompt & read n into $v0 ---
    li    $v0, 4
    la    $a0, prompt
    syscall

    li    $v0, 5
    syscall
    move  $s1, $v0          # $s1 = original n (saved)
    move  $a0, $v0          # $a0 = countdown = n

    # --- Compute n*n via recursion; result in $v0 ---
    jal   square_rec

    # --- Preserve the true result before printing the label ---
    move  $t0, $v0          # t0 = n*n

    # --- Print "Square is: " ---
```

ANIKA HAQUE, 163403

```
    li    $v0, 4
    la    $a0, result_msg
    syscall

    # --- Print the numeric result from $t0 ---
    move  $a0, $t0
    li    $v0, 1
    syscall

    # --- Newline & exit ---
    li    $v0, 4
    la    $a0, newline
    syscall
    li    $v0, 10
    syscall

# square_rec:
#   in:  $a0 = how many copies of n to add
#        $s1 = original n (never changed)
#   out: $v0 = original_n * (initial_count)
square_rec:
    addiu $sp, $sp, -4
    sw    $ra, 0($sp)

    blez  $a0, base_zero    # if count ≤ 0 → square = 0
    addiu $a0, $a0, -1      # decrement count
    jal   square_rec        # recurse
    addu  $v0, $v0, $s1      # add one copy of n
    j     unwind

base_zero:
    li    $v0, 0            # base case: 0 copies → 0

unwind:
    lw    $ra, 0($sp)
    addiu $sp, $sp, 4
    jr    $ra
```

The screenshot displays the MARS MIPS simulator interface. The main window shows the assembly code for 'a5q2.asm', which implements a recursive function to calculate the square of a number 'n'. The code includes comments explaining the logic: 'square_rec' calculates the square by adding 'n' to itself 'n' times, using a base case of 0. The registers window on the right shows the state of the MIPS registers, with 'a0' containing the input 'n' and 'v0' containing the result. The Run I/O window at the bottom shows the program's execution output, including prompts for 'Enter n:' and the resulting 'Square is: 25' for n=5, and 'Square is: 49' for n=7.

3. Write a **recursive** program to calculate factorial numbers. Use the definition of factorial as $F(n) = n * F(n-1)$

```
.data
prompt:      .asciiz "Enter n: "
result_msg:  .asciiz "Factorial is: "
newline:     .asciiz "\n"

.text
.globl main

main:
    # --- Prompt & read n into $a0 ---
    li      $v0, 4
    la      $a0, prompt
    syscall

    li      $v0, 5
    syscall
    move    $a0, $v0          # $a0 = n

    # --- Compute factorial(n) -> $v0 ---
    jal     factorial

    # --- Preserve result before printing label ---
    move    $t0, $v0          # t0 = n!
```

ANIKA HAQUE, 163403

```
# --- Print "Factorial is: " ---
li    $v0, 4
la    $a0, result_msg
syscall

# --- Print the numeric result from $t0 ---
move  $a0, $t0
li    $v0, 1
syscall

# --- Newline & exit ---
li    $v0, 4
la    $a0, newline
syscall
li    $v0, 10
syscall

# factorial:
#   in:  $a0 = n
#   out: $v0 = n!
factorial:
    addiu $sp, $sp, -8
    sw    $ra, 4($sp)
    sw    $a0, 0($sp)        # save original n

    li    $t1, 1
    ble   $a0, $t1, fact_base # if n ≤ 1, return 1

    # recursive case: compute fact(n-1)
    addiu $a0, $a0, -1
    jal   factorial
    lw    $t2, 0($sp)        # reload original n
    mul   $v0, $v0, $t2      # v0 = (n-1)! * n
    j     fact_done

fact_base:
    li    $v0, 1             # 0! = 1! = 1

fact_done:
    lw    $ra, 4($sp)
    addiu $sp, $sp, 8
    jr    $ra
```

The screenshot shows the MARS 4.5 MIPS simulator. The main window displays the assembly code for a factorial program. The code is as follows:

```

41
42 # factorial:
43 # in: $a0 = n
44 # out: $v0 = n!
45 factorial:
46     addiu $sp, $sp, -8
47     sw    $ra, 4($sp)
48     sw    $a0, 0($sp)    # save original n
49
50     li    $t1, 1
51     ble   $a0, $t1, fact_base    # if n ≤ 1, return 1
52
53     # recursive case: compute fact(n-1)
54     addiu $a0, $a0, -1
55     jal   factorial
56     lw    $t2, 0($sp)    # reload original n
57     mul   $v0, $v0, $t2    # v0 = (n-1)! * n
58     j     fact_done
59
60 fact_base:
61     li    $v0, 1    # 0! = 1! = 1
62
63 fact_done:
64     lw    $ra, 4($sp)
65     addiu $sp, $sp, 8
66     jr    $ra
67

```

The right panel shows the Register window with the following values:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x10010019
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x000002d0
\$t1	9	0x00000001
\$t2	10	0x00000006
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400058
hi		0x00000000
lo		0x000002d0

The bottom panel shows the Run I/O window with the following output:

```

Enter n: 5
Factorial is: 120

-- program is finished running --

Enter n: 6
Factorial is: 720

-- program is finished running --

```

The status bar at the bottom indicates "Line: 67 Column: 1" and "Show Line Numbers" is checked.

4. The following pseudo code converts an input value of a single decimal number from $1 \leq n \leq 15$ into a single hexadecimal digit. Translate this pseudo code into MIPS assembly.

```

main{
    String a[16]
    a[0] = "0x0"
    a[1] = "0x1"
    a[2] = "0x2"
    a[3] = "0x3"
    a[4] = "0x4"
    a[5] = "0x5"
    a[6] = "0x6"

```

ANIKA HAQUE, 163403

```
    a[7] = "0x7"
    a[8] = "0x8"
    a[9] = "0x9"
    a[10] = "0xa"
    a[11] = "0xb"
    a[12] = "0xc"
    a[13] = "0xd"
    a[14] = "0xe"
    a[15] = "0xf"

    int i = prompt("Enter a number from 0 to 15 ")
    print("your number is " + a[i])
}

.data
hex0:  .ascii "0x0"
hex1:  .ascii "0x1"
hex2:  .ascii "0x2"
hex3:  .ascii "0x3"
hex4:  .ascii "0x4"
hex5:  .ascii "0x5"
hex6:  .ascii "0x6"
hex7:  .ascii "0x7"
hex8:  .ascii "0x8"
hex9:  .ascii "0x9"
hexa:  .ascii "0xa"
hexb:  .ascii "0xb"
hexc:  .ascii "0xc"
hexd:  .ascii "0xd"
hexe:  .ascii "0xe"
hexf:  .ascii "0xf"

table: .word hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7
       .word hex8,hex9,hexa,hexb,hexc,hexd,hexe,hexf

newline: .ascii "\n"
prompt:  .ascii "Enter a value 0-15: "

.text
.globl main
main:
    # --- prompt for digit ---
    li    $v0, 4
    la    $a0, prompt
    syscall

    # --- read integer into t0 ---
    li    $v0, 5
```


ANIKA HAQUE, 163403

```
syscall
move $t0, $v0          # t0 = user input (0-15)

# --- compute address: table + t0*4 ---
la    $t1, table
sll   $t2, $t0, 2       # byte offset = input*4
add   $t1, $t1, $t2

# --- load the address of the right string into a0 ---
lw    $a0, 0($t1)

# --- print the hex string ---
li    $v0, 4
syscall

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

# --- exit ---
li    $v0, 10
syscall
```

Text Segment

Bkpt	Address	Code	Basic
	0x00400000	0x24020004	addiu \$2,\$0,0x00000004 29: li \$v0, 4
	0x00400004	0x3c011001	lui \$1,0x00001001 30: la \$a0, prompt
	0x00400008	0x34240082	ori \$4,\$1,0x00000082
	0x0040000c	0x0000000c	syscall 31: syscall
	0x00400010	0x24020005	addiu \$2,\$0,0x00000005 34: li \$v0, 5
	0x00400014	0x0000000c	syscall 35: syscall
	0x00400018	0x00024021	addu \$8,\$0,\$2 36: move \$t0, \$v0 #
	0x0040001c	0x3c011001	lui \$1,0x00001001 39: la \$t1, table
	0x00400020	0x34290040	ori \$9,\$1,0x00000040
	0x00400024	0x00085080	sll \$10,\$8,0x00000002 40: sll \$t2, \$t0, 2 #
	0x00400028	0x012a4820	add \$9,\$9,\$10 41: add \$t1, \$t1, \$t2
	0x0040002c	0x8d240000	lw \$4,0x00000000(\$9) 44: lw \$a0, 0(\$t1)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00307830	0x00317830	0x00327830	0x00337830
0x10010020	0x00387830	0x00397830	0x00617830	0x00627830
0x10010040	0x10010000	0x10010004	0x10010008	0x1001000c
0x10010060	0x10010020	0x10010024	0x10010028	0x1001002c
0x10010080	0x6e45000a	0x20726574	0x61762061	0x2065756c
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$8 (vaddr)	8	0x00000000
\$12 (status)	12	0x0000ff11
\$13 (cause)	13	0x00000000
\$14 (epc)	14	0x00000000

Mars Messages

```

Enter a value 0-15: 2
0x2

-- program is finished running --

Enter a value 0-15: 10
0xa

-- program is finished running --
  
```

5. The following pseudo code program calculates the Fibonacci numbers from $1 \dots n$, and stores them in an array. Translate this pseudo code into MIPS assembly, and use the PrintIntArray subprogram to print the results.

```

main{
    int size = PromptInt("Enter a max Fibonacci number to calc: ")
    int Fibonacci[size]

    Fibonacci[0] = 0
    Fibonacci[1] = 1

    for (int i = 2; i < size; i++){
        Fibonacci[i] = Fibonacci[i-1] + Fibonacci[i-2]
    }
}
  
```

ANIKA HAQUE, 163403

```
    }
    PrintIntArray(Fibonacci, size)
}

.data
prompt: .ascii "Enter max Fibonacci index: "
space:  .ascii " "
newline: .ascii "\n"

.text
.globl main
main:
    # --- Prompt & read n into $s0 ---
    li    $v0, 4
    la    $a0, prompt
    syscall
    li    $v0, 5
    syscall
    move  $s0, $v0          # length = n

    # --- Allocate n words on stack for Fib[0..n-1] ---
    sll   $t0, $s0, 2       # t0 = n*4
    subu  $sp, $sp, $t0
    move  $s1, $sp          # s1 = base of Fib[]

    # --- Base cases ---
    sw    $zero, 0($s1)     # Fib[0] = 0
    li    $t1, 1
    li    $t2, 2
    blt   $s0, $t2, skip1   # if n<2 skip Fib[1]
    sw    $t1, 4($s1)       # Fib[1] = 1
skip1:

    # --- Fill Fib[2..n-1] ---
    li    $t3, 2           # i = 2
fill_loop:
    bge   $t3, $s0, done_fill
    sll   $t4, $t3, 2
    add   $t5, $s1, $t4     # &Fib[i]

    # load Fib[i-1]
    addi  $t6, $t3, -1
    sll   $t6, $t6, 2
    add   $t6, $s1, $t6
    lw    $t7, 0($t6)

    # load Fib[i-2]
    addi  $t8, $t3, -2
    sll   $t8, $t8, 2
    add   $t8, $s1, $t8
    lw    $t9, 0($t8)

    add   $t7, $t7, $t9     # Fib[i] = Fib[i-1] + Fib[i-2]
    sw    $t7, 0($t5)

    addi  $t3, $t3, 1
    j     fill_loop
done_fill:
```

ANIKA HAQUE, 163403

```
# --- Print the array ---
move $a0,$s1      # base pointer
move $a1,$s0      # length
jal  PrintIntArray

# --- Restore stack & exit ---
move $sp,$s1
li   $v0,10
syscall

#-----
# PrintIntArray: prints each element of the int array on one line.
# In:  $a0 = base address, $a1 = length
PrintIntArray:
    addiu $sp,$sp,-12
    sw    $ra,8($sp)
    sw    $s0,4($sp)
    sw    $s1,0($sp)

    move  $s0,$a1      # counter = length
    move  $s2,$a0      # save base pointer in s2
    li    $t0,0        # index i = 0

print_loop:
    beq   $t0,$s0,end_print
    sll   $t1,$t0,2
    add   $t2,$s2,$t1   # t2 = address of Fib[i]
    lw    $a0,0($t2)    # load Fib[i]
    li    $v0,1
    syscall                # print integer

    addi  $t0,$t0,1
    blt   $t0,$s0,do_space
    j     print_loop

do_space:
    li    $v0,4
    la    $a0,space
    syscall                # print a space
    j     print_loop

end_print:
    li    $v0,4
    la    $a0,newline
    syscall                # final newline

    lw    $s1,0($sp)
    lw    $s0,4($sp)
    lw    $ra,8($sp)
    addiu $sp,$sp,12
    jr    $ra
```

CAUsers\anika\Documents\00 SFBU Spring 25\EE 488 Computer Architecture\Assignments\A5\A5q5.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit **Execute**

a5q5.asm

```

81      sll    $t1,$t0,2
82      add    $t2,$s2,$t1    # t2 = address of Fib[i]
83      lw     $a0,0($t2)     # load Fib[i]
84      li     $v0,1
85      syscall                # print integer
86
87      addi   $t0,$t0,1
88      blt    $t0,$s0,do_space
89      j      print_loop
90
91  do_space:
92      li     $v0,4
93      la     $a0,space
94      syscall                # print a space
95      j      print_loop
96
97  end_print:
98      li     $v0,4
99      la     $a0,newline
100     syscall                # final newline
101
102     lw      $s1,0($sp)
103     lw      $s0,4($sp)
104     lw      $ra,8($sp)
105     addiu   $sp,$sp,12

```

Line: 88 Column: 22 ☒ Show Line Numbers

Registers **Coproc 1** **Coproc 0**

Name	Number	Value
\$8 (vaddr)	8	0x00000000
\$12 (status)	12	0x0000ff11
\$13 (cause)	13	0x00000000
\$14 (epc)	14	0x00000000

Mars Messages **Run I/O**

Enter max Fibonacci index: 1
0
-- program is finished running --

Enter max Fibonacci index: 2
0 1
-- program is finished running --

Enter max Fibonacci index: 5
0 1 1 2 3
-- program is finished running --

Enter max Fibonacci index: 10
0 1 1 2 3 5 8 13 21 34
-- program is finished running --

Clear