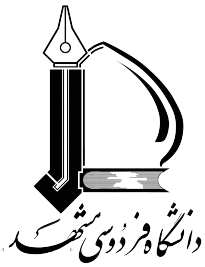

گروه مهندسی کامپیوتر

استاد درس: سرکار خانم دکتر ارشادی نسب

بهار ۱۴۰۲

تمرین سری دوم
درس معماری کامپیوتر

مهلت تحویل تا: ۱۴۰۲/۰۲/۲۳ **تمرین MIPS سری دوم**

- این تمرین به صورت گروهی است. (گروه های دو نفره)
- برای انجام تمرین دیدن ویدئوهای آموزش کار با syscall ها و آموزش کار با حافظه پویا که در ویو وجود دارد توصیه می‌شود.
- دقت داشته باشید که برای چاپ مقادیر در خروجی و همین طور برای گرفتن ورودی از کاربر باید با syscall ها آشنا باشید.
- برای تمرین مرتب سازی ادغامی نیز کار با Stack نیاز است بنابراین دیدن ویدئوی آموزش کار با حافظه‌ی پویا مفید خواهد بود.
- پاسخ تمرین را به صورت یک فایل با فرمت : FirstnameLastname_StudentNumber_FirstnameLastname_StudentNumber.zip بارگذاری کنید.
(مثال MohammadMohammadi_۹XXXXXX_RezaRezaei_۹XXXXXX.zip)
- کامنت گذاری تمام خطوط کد الزامی است.

- تحویل تکلیف بعد از مهلت مشخص شده نمره ای نخواهد داشت.
- در صورت اثبات کپی برداری، نمره تکالیف کپی شده و کپی شونده هر دو از ۱۰۰ نمره، ۱۰۰- خواهد بود.
- زمانبندی تحویل آنلاین تمرین پس از اتمام مهلت ارسال اعلام خواهد شد.
- تحویل تمرینات از طریق تلگرام، ایمیل و ... امکان پذیر نیست.

۱ محیط دایره

به زبان Assembly MIPS کدی بنویسید که محیط دایره با اندازه شعاع دلخواه اندازه گیری کند.
($\pi \approx 3$، دقت کنید که مقدار شعاع انتخابی عددی حسابی باشد.)
نکات :

۱. ورودی که همان شعاع دایره است به دلخواه شما می تواند در قسمت داده ها (.data segment) تعریف و یا از کاربر با چاپ پیام مناسب از طریق کنسول گرفته شود.

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول باید بگوید که محیط دایره با شعاع ورودی چه مقداری دارد.

پاسخ:

در این کد، مقادیر شعاع و π را به ترتیب در رجیسترهای $s۰ و $s۱ از حافظه بارگذاری می‌کنیم. سپس محیط دایره را با ضرب ۲، π و شعاع با استفاده از دستور mul محاسبه می‌کنیم. نتیجه را در رجیستر $t۰ ذخیره می‌کنیم. در نهایت، با استفاده از دستور syscall با پارامترهای مناسب، محیط دایره را چاپ می‌کنیم. سپس با استفاده از دستور دیگری به نام syscall با کد خروجی مناسب، برنامه را خاتمه می‌دهیم.
توجه: این کد فرض می‌کند که مقدار π برابر با ۳ است. در صورت نیاز به دقت بیشتر، مقدار π در حافظه باید به‌روز شود.

```
.data
radius: .word 10       # radius of the circle
pi:     .word 3        # value of pi

.text
main:
    # load radius and pi into registers
    lw $s0, radius
    lw $s1, pi

    # calculate circumference of the circle
    addi $t0, $zero, 2      # $t0 = 2
    mul $t0, $t0, $s1       # $t0 = 2 * pi
    mul $t0, $t0, $s0       # $t0 = 2 * pi * radius

    # print the circumference of the circle
    addi $v0, $zero, 1      # $v0 = 1 (print integer)
    add $a0, $t0, $zero     # $a0 = $t0 (circumference)
    syscall

    # exit program
    addi $v0, $zero, 10     # $v0 = 10 (exit)
    syscall
```

Listing ۱: Circumference of the Circle

۲ تبدیل اعداد دودویی به دهدهی

به زبان Assembly MIPS کدی بنویسید که اعداد دودویی (binary) را به اعداد دهدهی (decimal) تبدیل کند.

نکات :

۱. ورودی همان عدد مورد نظر در مبنای دو هست که باید در قسمت داده ها (.data segment) تعریف شود. (می توان تعداد ارقام عدد ورودی را هم در ورودی تعیین کرد.)

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول باید معادل دهدهی عدد باینری ورودی را چاپ کند.

پاسخ:

این کد MIPS، یک عدد دودویی که به صورت یک رشته در حافظه ذخیره شده، به معادل دهدهی آن تبدیل می کند. ابتدا آدرس رشته دودویی در حافظه در $s۰ قرار داده می شود و سپس طول رشته در $t۰ ذخیره می شود. سپس یک حلقه شروع می شود که به تمامی کاراکترهای رشته دودویی می رود و هر کدام را به معادل دهدهی آن تبدیل می کند. مقدار دهدهی به صورت پیوسته به وسیله یک حلقه در $t۱ ذخیره می شود. در نهایت، مقدار دهدهی به وسیله سیستم کال بر روی کنسول چاپ می شود.

```
# Assume that the binary number is stored in a string in memory
# starting from address "bin_num". The length of the string is "len".

.data
bin_num: .asciiz "10101101"
len: .word 8

.text
.globl main
main:
    # Load the address of the binary string into $s0
    la $s0, bin_num

    # Load the length of the string into $t0
    lw $t0, len

    # Initialize decimal value to zero
    li $t1, 0

loop:
    # Load the current character of the binary string into $t2
    lb $t2, ($s0)

    # If we've reached the end of the string, exit the loop
    beq $t0, $zero, end_loop

    # Convert the character to its corresponding integer value
    sub $t2, $t2, 48

    # Multiply the current decimal value by 2
    sll $t1, $t1, 1

    # Add the current binary digit to the decimal value
    add $t1, $t1, $t2

    # Move to the next character of the string
    addi $s0, $s0, 1

    # Decrement the string length
    addi $t0, $t0, -1

    # Go back to the beginning of the loop
    j loop

end_loop:
    # The decimal value is now in $t1
    # Do something with it, such as print it
    li $v0, 1
    move $a0, $t1
    syscall

    # Terminate the program
    li $v0, 10
    syscall
```

Listing ۲: Binary Number to Decimal

۳ پیداکردن بزرگترین مقسوم علیه مشترک

به زبان Assembly MIPS کدی بنویسید که بزرگترین مقسوم علیه مشترک دو عدد را محاسبه کند.
نکات :

۱. ورودی که دو عدد مورد نظر برای محاسبه بزرگترین مقسوم علیه مشترک آن ها است به دلخواه شما می تواند در قسمت داده ها (.data segment) تعریف و یا از کاربر با چاپ پیام مناسب از طریق کنسول گرفته شود.

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول بزرگترین مقسوم علیه مشترک دو عدد ورودی را چاپ می کند.

۳. فرض می کنیم اعداد ورودی فقط اعداد حسابی خواهد بود.

پاسخ:

این برنامه دو عدد را دریافت کرده و بزرگترین مقسوم علیه آن‌ها را پیدا می‌کند و چاپ می‌کند. برای این کار ابتدا دو عدد با استفاده از دستور lw از حافظه خوانده می‌شود و سپس یک حلقه برای محاسبه مقسوم علیه بزرگترین شروع می‌شود. در هر مرحله در ابتدا بررسی می‌شود که آیا یکی از این دو عدد صفر است؟ اگر یکی از آن‌ها صفر باشد، عدد دیگر مقسوم علیه بزرگترین است. در غیر این صورت، دو عدد بررسی می‌شوند تا بزرگترین را پیدا کنند و سپس از کوچک‌ترین به بزرگ‌ترین عدد کم می‌شود. اگر عدد اول کوچک‌تر باشد، آن‌ها جا به جا می‌شوند تا در مرحله بعدی عدد اول بزرگتر باشد. پس از پیدا کردن مقسوم علیه بزرگترین، پیغامی رشته‌ای که نشان‌دهنده مقسوم علیه بزرگترین است، با استفاده از دستور li و la چاپ می‌شود. سپس مقسوم علیه بزرگترین با استفاده از دستور move در $t۱ قرار داده می‌شود و در نهایت یک کاراکتر خط جدید چاپ می‌شود و برنامه خاتمه می‌یابد.

```
.data
num1: .word 48
num2: .word 60
newline: .asciiz "\n"
result: .asciiz "The GCD is: "

.text
.globl main
main:
    # load the two numbers into registers
    lw $t0, num1
    lw $t1, num2

    # set up a loop to calculate the GCD
    loop:
        beq $t0, $0, end # if $t0 = 0, GCD = $t1
        beq $t1, $0, end # if $t1 = 0, GCD = $t0

        # check if $t0 > $t1
        slt $t2, $t0, $t1
        bne $t2, $0, swap # if $t0 < $t1, swap them

        # subtract $t1 from $t0
        sub $t0, $t0, $t1

        j loop # jump back to loop

    swap:
        move $t2, $t0
        move $t0, $t1
        move $t1, $t2
        j loop # jump back to loop

    end:
        # print the result
        li $v0, 4 # system call for printing a string
        la $a0, result
        syscall

        li $v0, 1 # system call for printing an integer
        move $a0, $t1 # print the GCD (in $t1)
        syscall

        # print a newline character
        li $v0, 4 # system call for printing a string
        la $a0, newline
        syscall

        # exit the program
        li $v0, 10 # system call for exiting a program
        syscall
```

Listing ۳: GCD of Two Numbers

۴ محاسبه فاکتوریل

به زبان Assembly MIPS کدی بنویسید که فاکتوریل یک عدد را محاسبه کند .
نکات :

۱. ورودی که عدد مورد نظر برای محاسبه فاکتوریل آن است به دلخواه شما می تواند در قسمت داده ها (.data segment) تعریف و یا از کاربر با چاپ پیام مناسب از طریق کنسول گرفته شود.

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول فاکتوریل عدد ورودی را چاپ می کند.

۳. فرض می کنیم اعداد ورودی فقط اعداد طبیعی خواهد بود.

پاسخ:

این کد برنامه‌ای را برای محاسبه فاکتوریل یک عدد غیر منفی نوشته است. اولین بخش کد، بخش داده‌ها است. در این بخش، سه رشته حاوی متن‌های " :Enter a non-negative integer"، "The factorial is: " و "\n" تعریف شده است. بخش دیگر کد، بخش متن است. در این بخش، تابع main نوشته شده است. این تابع با چاپ متن " :Enter a non-negative integer" کار خود را شروع می‌کند و عدد ورودی کاربر را دریافت می‌کند. سپس، متغیرهای $t۱ و $t۲ به ترتیب برای نگه‌داری فاکتوریل و شمارنده حلقه اولیه مقداردهی می‌شوند. سپس، در حلقه‌ای با استفاده از دستورات بررسی شرطی ،(bgt) ضرب (mul) و جمع یکی ،(addi) فاکتوریل عدد ورودی محاسبه می‌شود. در هر دور از حلقه، شمارنده حلقه به مقدار یکی اضافه می‌شود و بررسی می‌شود که آیا شمارنده از عدد ورودی بزرگتر شده است یا خیر؟ در صورتی که شمارنده از عدد ورودی بزرگتر شده باشد، از حلقه خارج شده و فاکتوریل محاسبه شده را چاپ می‌کند. در پایان، برنامه با دستور li $v0, 10 و syscall خاتمه می‌یابد و اجرای برنامه به پایان می‌رسد.

```
.data
prompt: .asciiz "Enter a non-negative integer: "
result: .asciiz "The factorial is: "
newline: .asciiz "\n"

.text
.globl main

main:
    # Print prompt and read input
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $t0, $v0  # Store input in $t0

    # Initialize variables
    li $t1, 1     # $t1 stores the factorial
    li $t2, 1     # $t2 stores the loop counter

loop:
    # Check if loop counter is greater than input
    bgt $t2, $t0, print_result

    # Multiply factorial by loop counter
    mul $t1, $t1, $t2

    # Increment loop counter
    addi $t2, $t2, 1

    # Repeat loop
    j loop

print_result:
    # Print result
    li $v0, 4
    la $a0, result
    syscall
    li $v0, 1
    move $a0, $t1
    syscall
    # Print newline
    li $v0, 4
    la $a0, newline
    syscall

    # Exit program
    li $v0, 10
    syscall
```

Listing :۴ Factorial of a Number

۵ مرتب‌سازی ادغامی

به زبان Assembly MIPS کدی بنویسید که با گرفتن اندازه و اعضای آرایه مورد نظر، آرایه را به روش مرتب سازی ادغامی مرتب کند.

نکات :

۱. ورودی که تعداد اعضای آرایه و خود آرایه مورد نظر برای مرتب کردن است باید از کاربر با چاپ پیام مناسب از طریق کنسول گرفته شود.

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول آرایه مرتب شده را چاپ می کند.

۳. فرض می کنیم اعداد ورودی فقط اعداد حسابی خواهد بود.

پاسخ:

این برنامه ابتدا از کاربر تعداد عناصر آرایه را دریافت می‌کند، سپس عناصر آرایه را از کاربر می‌خواهد و در یک آرایه اولیه ذخیره می‌کند. در ادامه، برنامه با استفاده از تابع Mergesort آرایه را مرتب می‌کند و سپس آرایه مرتب شده را چاپ می‌کند. در بخش data. ابتدا سه آرایه و چند متغیر تعریف شده است، این آرایه‌ها برای نگهداری موقتی داده‌ها هستند و متغیرها برای ذخیره اطلاعاتی مانند تعداد عناصر آرایه و اندازه آرایه و آدرس پایان آرایه تعریف شده‌اند. در بخش text. ابتدا برنامه از کاربر تعداد عناصر آرایه را دریافت می‌کند و سپس عناصر آرایه را در یک حلقه از کاربر می‌خواهد و در آرایه‌ای اولیه ذخیره می‌کند. سپس آدرس پایان آرایه و اندازه آرایه محاسبه شده و تابع Mergesort صدا زده می‌شود. تابع Mergesort به طور بازگشتی آرایه را به دو بخش تقسیم می‌کند تا زمانی که بخش‌های آرایه یک عنصر داشته باشند، سپس این بخش‌ها را به صورت مرتب شده با هم ترکیب می‌کند. تابع Merge نیز دو زیرآرایه را به یکدیگر می‌چسباند و نتیجه را در آرایه‌ی اصلی ذخیره می‌کند. در نهایت، تابع printArray آرایه را چاپ می‌کند.

```
.data
        list:       .space 100000 # original array of
                                  # unsorted values
        left:       .space 100000 # temporary array to hold
                                  # left half of main array
        right:      .space 100000 # temporary array to hold
                                  # right half of main array
        n:      .word 0 # Holds the number of values to be sorted
        arraySize:      .word 0
        arrayEndAddress:      .word 0
        prompt1:      .asciiz "Enter n: "
        prompt2:      .asciiz "Enter element "
        dialog:      .asciiz "The sorted list is: "
        space:      .asciiz " "
        colon:      .asciiz ": "
        eol:      .asciiz "\n"

.text
main:
        la $a0, prompt1
        li $v0, 4
        syscall

        li $v0, 5
        syscall
        sw $v0, n

        lw $t1, n
        sll  $t1, $t1, 2
        li $t2, 1
        li $t0, 0
        writeElements:
                bge $t0, $t1, done

                la $a0,prompt2
                li $v0, 4
                syscall

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

                la $a0,colon
                li $v0, 4
                syscall

                li $v0, 5
                syscall

                sw $v0, list($t0)
                add $t0, $t0, 4

                j writeElements
        done:
```

Listing :۵ Merg Sort

```
                sw $t0, arrayEndAddress
                la $t1, list
                sub $t0, $t0, $t1
                sw $t0, arraySize
                jal mergesort
        jal printArray
exit:
        li $v0, 10
        syscall

# Function: Mergesort
# Description: Splits the Array (or subarray) into smaller subarray
# Receives: Nothing
# Returns: Nothing
mergesort:
        add $sp, $sp, -4
        sw $ra, 0($sp)
        lw $t0, n
        sub $t0, $t0, 1
        li $t1, 1
        for1:
                li $t2, 0
                li $t3, 0
                li $t4, 0
                for2:
                        addu $t3, $t2, $t1
                        sub $t3, $t3, 1

                        sll $t4, $t1, 1
                        addu $t4, $t4, $t2
                        sub $t4, $t4, 1

                        findmin:
                                 blt  $t4, $t0, setmin
                                 move $t4, $t0
                                setmin: move $t4, $t4

                        jal merge

                        sll $t5, $t1, 1
                        addu $t2, $t2, $t5
                        blt $t2, $t0, for2
                sll $t1, $t1, 1
                ble $t1, $t0, for1
mergesortDone:
        lw $ra, 0($sp)
        add $sp, $sp, 4
        jr $ra


# Function: Merge
# Description: combines two subarrays
#              into one sorted array and updates the original array
# Receives: -$t2, $t3, $t4
#           - $t2 has lower index of subarray
```

Listing :۶ Merg Sort

```
111 #           - $t3 has the middle index of the subarray
112 #           - $t4 has the index of the end of the subarray
113 # Returns: Nothing
114 merge:
115         addi $sp, $sp, -20
116         sw $t0, 0($sp)
117         sw $t1, 4($sp)
118         sw $t2, 8($sp)
119         sw $t3, 12($sp)
120         sw $t4, 16($sp)
121
122         lw $s2, 8($sp)
123         lw $s3, 12($sp)
124         lw $s4, 16($sp)
125
126         addi $t0, $s3,1
127         sub $t0, $t0, $s2
128         sub $t1, $s4, $s3
129
130         li $t3, 0
131         li $t4, 0
132         for3: #copy left half of array
133                 add $s5, $s2, $t3
134                 sll $s5, $s5, 2
135                 lw $t5, list($s5)
136                 sll $s6, $t3, 2
137                 sw $t5, left($s6)
138
139                 addiu $t3, $t3, 1
140                 sltu $t9,$t3, $t0
141                 bne $t9, $0, for3
142
143
144         for4: #copy right half of the array
145                 addi $s5, $s3,1
146                 add $s5, $s5, $t4
147                 sll $s5, $s5, 2
148
149                 lw $t5, list($s5)
150                 sll $s6, $t4, 2
151
152                 sw $t5, right($s6)
153                 addi $t4, $t4, 1
154                 sltu $t9, $t4, $t1
155                 bne $t9, $0, for4
156
157         li $t3, 0
158         li $t4, 0
159         move $s0, $s2
160         while1: #loop to sort the subarrays
161                 slt $a2, $t3, $t0
162                 slt $a3, $t4, $t1
163                 and $v1,$a2, $a3
164                 beqz $v1, while2
165
```

Listing :۷ Merg Sort

```
                sll $s2, $t3, 2
                lw $t5, left($s2)

                sll $s3, $t4, 2
                lw $t6, right($s3)

                sll $s4, $s0, 2
                if:
                        bgt $t5, $t6, else
                        sw $t5, list($s4)
                        addi $t3, $t3, 1
                        b loop
                else:
                        sw $t6, list($s4)
                        addi $t4, $t4, 1
                        b loop
                loop:
                        addi $s0, $s0, 1
                        j while1
        while2: #copy the remaining values of
                #the left subarray into the original array
                bge $t3, $t0, while3
                sll $s2, $t3, 2
                lw $t5, left($s2)
                sll $s4, $s0, 2
                sw $t5, list($s4)
                addi $t3, $t3, 1
                addi $s0, $s0, 1
                j while2
        while3: #copy the remaining values of
                #the right subarray into the original array
                bge $t4, $t1, doneMerge
                sll $s2, $t4, 2
                lw $t5, right($s2)
                sll $s4, $s0, 2
                sw $t5, list($s4)
                addi $t4, $t3, 1
                addi $s0, $s0, 1
                j while3
doneMerge:
        lw $t2, 8($sp)
        lw $t1, 4($sp)
        lw $t0, 0($sp)
        addi $sp, $sp, 20
        jr $ra


# Function: printArray:
# Description: Displays the sorted numbers on the screen
# Receives: Nothing
```

Listing ۸: Merg Sort

```
# Returns: Nothing
printArray:
        add $sp, $sp,-4
        sw $ra, 0($sp)
        la $a0, dialog
        li $v0, 4
        syscall

        li $t0, 0
        lw $t1, n
        loop1:
                bge $t0, $t1, donePrint
                sll $s0, $t0, 2
                lw $a0, list($s0)
                li $v0, 1
                syscall

                la $a0, space
                li $v0, 4
                syscall

                add $t0, $t0, 1
                j loop1
        la $a0, eol
        li $v0, 4
        syscall
donePrint:
        lw $ra, 0($sp)
        jr $ra

```

Listing ۹: Merg Sort