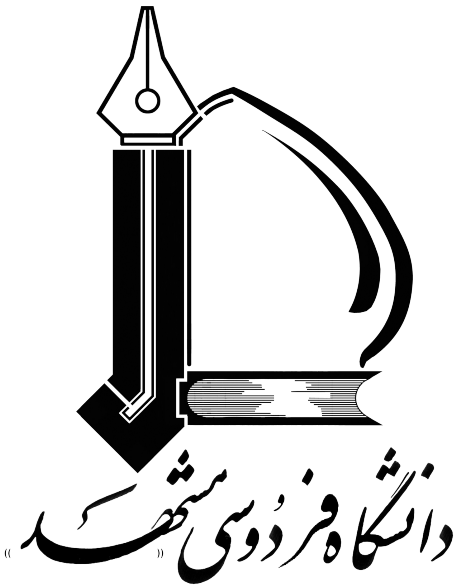

گروه مهندسی کامپیوتر

استاد درس: سرکار خانم دکتر ارشادی نسب

بهار ۱۴۰۲

تمرین سری اول
درس معماری کامپیوتر

مهلت تحویل تا: ۱۴۰۲/۰۱/۱۵ **تمرین MIPS سری اول**

- پاسخ تمرین را به صورت یک فایل با فرمت FirstnameLastname_StudentNumber.zip بارگذاری کنید.(مثال MohammadMohammadi_۹XXXXXX.zip)
- برای انجام تمرین دیدن ویدئوی آموزش کار با syscall ها 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.

- کامنت گذاری تمام خطوط کد الزامی است.
- دقت کنید برای نوشتن برنامه های مورد نظر فقط باید از دستورات مجاز برای همان برنامه استفاده کنید.
- 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 نامساوی مثلث

به زبان Assembly MIPS کدی بنویسید که نامساوی مثلث را برای اندازه های دلخواه بررسی کند و با چاپ خروجی مناسب ما را از امکان ساخت مثلثی با آن اضلاع مطلع سازد.
نکات :

۱. ورودی که همان اندازه سه ضلع مثلث است می‌بایست در قسمت داده ها در کد شما تعریف شود. (.data segment)

۲. خروجی برنامه با چاپ پیغامی مناسب در کنسول باید بگوید که این سه ضلع تشکیل یک مثلث می دهند و یا خیر

۳. فرض می کنیم کاربر اعداد حسابی را می‌تواند وارد کند یعنی ورودی فقط اعداد حسابی خواهد بود.

جدول ۱: دستورات مجاز برای استفاده در نامساوی مثلث

Example	Description	Instruction
add $t0, $s0, $s1	Add	add
la $t, label	Load Address	la
li $t, imm	Load Immediate	li
and $t0, $s0, $s1	Bitwise AND	and
slt $t0, $s0, $s1	Set Less Than	slt
lw $t0, 4($s0)	Load Word	lw
sw $t0, 4($s0)	Store Word	sw
beq $s0, $s1, label	Branch If Equal Zero	beqz
j label	Jump	j
syscall	Triggers a System Call	syscall

پاسخ:

طول هر یک از ضلع‌ها در بخش data. تعریف شده است. در بخش text. برنامه ، ابتدا آدرس ضلع a به $t۰ اختصاص داده شده و سپس مقدار a در $t۰ قرار داده می‌شود. به همین ترتیب برای b و c نیز عمل

می‌شود. برای بررسی عدم تساوی مثلث، با استفاده از دستور SLT، ابتدا مقدار c + b با استفاده از دستور ADD در $t۳ قرار داده شده و سپس با استفاده از دستور SLT، ارزش ۱ یا ۰ در $t۴ قرار می‌گیرد، به ترتیبی که اگر c + b > a ، $t۴ برابر با ۱ خواهد بود و در غیر این صورت $t۴ برابر با ۰ خواهد بود. در مراحل بعدی نیز برای بررسی عدم تساوی مثلث، مقادیر $t۵ و $t۶ با استفاده از دستور ADD و SLT محاسبه می‌شوند. سپس مقادیر $t۴، $t۵ و $t۶ به ترتیب با استفاده از دستور AND با هم ادغام می‌شوند تا ارزش ۱ یا ۰ در $t۷ قرار گیرد. اگر $t۷ برابر با ۱ باشد به برچسب valid پرش می‌شود و در غیر این صورت به برچسب invalid پرش می‌شود. برچسب valid، یک رشته "The triangle is valid" را چاپ کرده و برنامه را خاتمه می‌دهد. برچسب invalid نیز یک رشته "The triangle is invalid" را چاپ کرده و برنامه را خاتمه می‌دهد.

```
.data
    a: .word 4   # length of side a
    b: .word 3   # length of side b
    c: .word 5   # length of side c
    result: .asciiz "The triangle is valid\n"
    error: .asciiz "The triangle is invalid\n"

.text
    main:
        la $t0, a  # load address of a into $t0
        lw $t0, ($t0)  # load value of a into $t0

        la $t1, b  # load address of b into $t1
        lw $t1, ($t1)  # load value of b into $t1

        la $t2, c  # load address of c into $t2
        lw $t2, ($t2)  # load value of c into $t2

        # check inequality of triangle
        add $t3, $t1, $t2  # t3 = b + c
        slt $t4, $t0, $t3  # if a < b + c, $t4 = 1, else $t4 = 0

        add $t3, $t0, $t2  # t3 = a + c
        slt $t5, $t1, $t3  # if b < a + c, $t5 = 1, else $t5 = 0

        add $t3, $t0, $t1  # t3 = a + b
        slt $t6, $t2, $t3  # if c < a + b, $t6 = 1, else $t6 = 0

        and $t7, $t4, $t5  # if a < b + c and b < a + c, $t7 = 1, else $t7 = 0
        and $t7, $t7, $t6  # if a < b + c and b < a + c and c < a + b, $t7 = 1, else $t7 = 0

        beqz $t7, invalid  # if $t7 = 0, branch to invalid

        # if $t7 = 1, branch to valid
        j valid

    valid:
        # print "The triangle is valid"
        la $a0, result
        li $v0, 4
        syscall

        # exit program
        li $v0, 10
        syscall

    invalid:
        # print "The triangle is invalid"
        la $a0, error
        li $v0, 4
        syscall

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

Listing ۱: Inequality of the Triangle

۲ دنباله فیبوناچی

به زبان Assembly MIPS کدی بنویسید که دو عدد را از کاربر دریافت کند و اعضای دنباله فیبوناچی که بین این دو عدد است را چاپ کند.
نکات :

۱. ورودی باید از طریق کنسول از کاربر گرفته شود. برای دریافت هر ورودی از کاربر از طریق کنسول باید پیغام مناسب چاپ شود.

۲. فرض می‌کنیم که کاربر فقط اعداد حسابی را وارد می‌کند و عدد اولی که وارد می‌کند عدد کران پایین دامنه و عدد دوم عدد کران بالای دامنه‌ی چاپ دنباله باشد.

۳. خروجی برنامه باید اعضای دنباله فیبوناچی که بین دو کران داده شده‌اند هستند را به شکلی خوانا در کنسول چاپ کند.

جدول ۲: دستورات مجاز برای استفاده در دنباله فیبوناچی

Example	Description	Instruction
add $t0, $s0, $s1	Add	add
addi $t0, $s0, imm	Add With Immediate	addi
la $t, label	Load Address	la
li $t, imm	Load Immediate	li
move $t0, $s0	Move Value	move
bgt $t1, $t0, Label	Branch on Greater Than	bgt
bge $t1, $t0, Label	Branch on Greater Than Equal	bge
j label	Jump	j
syscall	Triggers a System Call	syscall

پاسخ:

این کد MIPS برای محاسبه دنباله فیبوناچی بین دو عدد است. در بخش ،data. چند رشته ثابت تعریف شده است که در بخش text. استفاده می شوند. رشته های prompt۱ و prompt۲ برای پرسیدن از کاربر عدد اول و دوم مورد استفاده قرار می گیرند. رشته output برای چاپ عنوان دنباله فیبوناچی و رشته های comma و newline برای چاپ کاما و خط جدید مورد استفاده قرار می گیرند. در بخش ،text. در بخش main ابتدا از کاربر عدد اول و دوم پرسیده می شود و سپس دنباله فیبوناچی بین دو عدد محاسبه می شود. در ابتدای حلقه ،Loop اگر شماره فیبوناچی کنونی بیشتر از عدد دوم باشد، حلقه بسته می شود. در صورتی که شماره فیبوناچی کنونی بین دو عدد باشد، آن را چاپ می کنیم و به شماره بعدی می رویم. در صورتی که شماره فیبوناچی کنونی کمتر از عدد دوم باشد، ما به شماره بعدی می رویم. پس از پایان حلقه، یک خط جدید چاپ می شود و برنامه خاتمه می یابد.

```
.data
prompt1: .asciiz "Enter the first number: "
prompt2: .asciiz "Enter the second number: "
output: .asciiz "Fibonacci sequence: "
comma: .asciiz ", "
newline: .asciiz "\n"
.text
.globl main

main:
# prompt the user to enter the first number
li $v0, 4
la $a0, prompt1
syscall

# read the first number
li $v0, 5
syscall
move $t0, $v0

# prompt the user to enter the second number
li $v0, 4
la $a0, prompt2
syscall

# read the second number
li $v0, 5
syscall
move $t1, $v0

# initialize the Fibonacci sequence
addi $t2, $zero, 1
addi $t3, $zero, 1

# print out the header for the sequence
li $v0, 4
la $a0, output
syscall

# print out the Fibonacci sequence between the two numbers
Loop:
    # check if we've gone past the second number
    bgt $t2, $t1, Done

    # check if the current Fibonacci number is within range
    bge $t2, $t0, Print

    # calculate the next Fibonacci number
    add $t4, $t2, $t3
    move $t2, $t3
```

Listing ۲: Fibonacci

```
    move $t3, $t4

    # loop back to the start of the loop
    j Loop

Print:
    # print out the current Fibonacci number
    li $v0, 1
    move $a0, $t2
    syscall

    # print a comma and a space
    li $v0, 4
    la $a0, comma
    syscall

    # calculate the next Fibonacci number
    add $t4, $t2, $t3
    move $t2, $t3
    move $t3, $t4

    # loop back to the start of the loop
    j Loop

Done:
    # print a newline character and exit
    li $v0, 4
    la $a0, newline
    syscall

    li $v0, 10
    syscall
```

Listing ۳: Fibonacci

۳ محاسبه میانگین

به زبان MIPS Assembly کدی بنویسید که میانگین یک آرایه را محاسبه کند.
نکات :

۱. ورودی که شامل یک آرایه و اندازه آن است باید در قسمت داده ها در کد تعریف شود. (.data segment)

۲. فرض این است که اعضای آرایه و اندازه آن اعدادی حسابی اند.

۳. در خروجی، برنامه باید مقدار میانگین آرایه را در کنسول چاپ کند. (مقدار میانگین نیازی به دقت اعشاری ندارد یعنی میانگین به صورت عددی صحیح چاپ می‌شود.)

جدول ۳: دستورات مجاز برای استفاده در محاسبه میانگین

Example	Description	Instruction
add $t0, $s0, $s1	Add	add
addi $t0, $s0, imm	Add With Immediate	addi
la $t, label	Load Address	la
li $t, imm	Load Immediate	li
div $t0, $s1, $s0	Divide (with overflow)	div
blt $t1, $t0, Label	Branch on Less Than	blt
mflo $a0	Move From LO of the Register	mflo
lw $t0, 4($s0)	Load Word	lw
syscall	Triggers a System Call	syscall

پاسخ:

این کد یک برنامه MIPS است که میانگین اعداد موجود در یک آرایه را محاسبه می‌کند. این برنامه ابتدا آرایه ای با ۵ عنصر ایجاد می‌کند و سپس مقدار اندازه آرایه را در متغیر arraySize ذخیره می‌کند. سپس در قسمت main، آدرس ابتدایی آرایه در $t۰ قرار می‌گیرد و اندازه آرایه در $t۱ قرار می‌گیرد. سپس یک شمارنده به نام counter در $t۲ و یک متغیر جمع در $t۳ ایجاد می‌شوند. در ادامه، یک حلقه به نام loop ایجاد می‌شود که اعداد آرایه را یکی یکی بررسی می‌کند و با هر بار بررسی، شمارنده یکی افزایش پیدا می‌کند و مقدار آن عضو را به مجموع اضافه می‌کند. در پایان حلقه، میانگین با تقسیم مجموع بر اندازه آرایه محاسبه می‌شود و نتیجه در $a۰ قرار داده شده و با استفاده از سرویس سیستمی ۱ چاپ می‌شود. در نهایت، با استفاده از سرویس سیستمی ۱۰، برنامه پایان می‌یابد.

```
.data
myArray: .word 5, 10, 15, 20, 25    # initialize an array with 5 elements
arraySize: .word 5                  # initialize a variable with
                                    # the size of the array

.text
.globl main

main:
    la $t0, myArray         # load the base address of the array into $t0
    lw $t1, arraySize       # load the size of the array into $t1
    li $t2, 0               # initialize a counter variable to 0
    li $t3, 0               # initialize a sum variable to 0

loop:
    lw $t4, ($t0)           # load the next element of the array into $t4
    add $t2, $t2, 1         # increment the counter
    add $t3, $t3, $t4       # add the element to the sum
    addi $t0, $t0, 4        # increment the address to point to the next element
    blt $t2, $t1, loop      # branch back to loop if the counter
                            # is less than the size of the array

    div $t5, $t3, $t1       # divide the sum by the size of
                            # the array to get the average
    mflo $a0                # move the result into $a0 for printing
    li $v0, 1               # set the syscall number to 1 for printing integers
    syscall

    li $v0, 10              # set the syscall number to 10 to exit the program
    syscall
```

Listing :۴ Average

۴ محاسبه عبارات بولی

به زبان Assembly MIPS برای هر مورد از عبارات زیر کدی بنویسید که کاربر با مقدار دادن به پارامتر های a, b, c, ... بتواند پاسخ عبارت را محاسبه کند.

توجه :

۱. ورودی که شامل مقادیر متناظر هر پارامتر است باید در قسمت داده ها در کد تعریف شود. (.data segment)

۲. لطفاً عبارات را بدون تغییر پیاده سازی کنید. (عبارات را ساده نکنید.)

۳. منظور از علامت $\oplus$ همان طور که در درس مدارهای منطقی و ریاضیات گسسته خواندید XOR است، علامت $\vee$ به معنای OR منطقی، $\wedge$ به معنای AND منطقی و علامت $\neg$ به معنای NOT منطقی است.

۴. خروجی برنامه باید پاسخ متناظر با ورودی های پارامتر ها را در کنسول چاپ کند.

جدول ۴: دستورات مجاز برای استفاده در محاسبه عبارات بولی

Example	Description	Instruction
add $t0, $s0, $s1	Add	add
addi $t0, $s0, imm	Add With Immediate	addi
and $t0, $s0, $s1	Bitwise AND	and
or $t0, $s0, $s1	Bitwise OR	or
nor $t0, $s0, $s1	NOR	nor
xor $t0, $s0, $s1	XOR	xor
lw $t0, 4($s0)	Load Word	lw
syscall	Triggers a System Call	syscall

۱.۴ عبارت اول

$$(a \wedge b) \oplus (c \wedge d) \vee (e \wedge f)$$

پاسخ:

از دستور addi برای بارگذاری مقادیر ۱ و ۱۰ در رجیستر $v۰ استفاده می‌شود، که برای کد فراخوانی سیستم برای چاپ یک عدد صحیح و خروج از برنامه مورد استفاده قرار می‌گیرد. برای خروجی دادن به ترمینال، از دستور add برای کپی کردن محتویات $t۴ به $a۰ استفاده می‌شود که آرگومان دستور فراخوانی سیستم برای چاپ یک عدد صحیح را نگه می‌دارد. مقادیر ورودی از بخش داده با استفاده از دستور lw بارگذاری می‌شوند و عبارت با استفاده از دستورات and، xor و or محاسبه می‌شود. در نهایت، برنامه با استفاده از دستورات addi و syscall خاتمه می‌یابد.

```
.data
a: .word 1
b: .word 0
c: .word 0
d: .word 1
e: .word 1
f: .word 1

.text
.globl main
main:
    # load input values from data section
    lw $s0, a
    lw $s1, b
    lw $s2, c
    lw $s3, d
    lw $s4, e
    lw $s5, f

    # compute the expression
    and $t0, $s0, $s1     # t0 = a AND b
    and $t1, $s2, $s3     # t1 = c AND d
    and $t2, $s4, $s5     # t2 = e AND f
    xor $t3, $t0, $t1     # t3 = (a AND b) XOR (c AND d)
    or  $t4, $t3, $t2     # t4 = (a AND b) XOR (c AND d) OR (e AND f)

    # output the result to the terminal
    addi $v0, $zero, 1    # set $v0 to 1 (for syscall 1 - print integer)
    add  $a0, $t4, $zero  # copy t4 to a0
    syscall

    # exit the program
    addi $v0, $zero, 10   # set $v0 to 10 (for syscall 10 - exit)
    syscall
```

Listing ۵: Logical Expression One

۲.۴ عبارت دوم

$(a \vee b) \wedge \neg(c \vee d)$

پاسخ:

این برنامه برای محاسبه عبارت منطقی "$(a \vee b) \wedge \neg(c \vee d)$" به زبان Assembly MIPS نوشته شده است. در ابتدا، مقادیر ورودی برنامه در بخش data. ذخیره شده اند. سپس در بخش ،text. محاسبات مورد نیاز برای محاسبه عبارت منطقی انجام می‌شوند. در این برنامه، ابتدا مقادیر ورودی از بخش data. با استفاده از دستور lw بارگذاری می‌شوند. سپس عملگر or برای محاسبه عبارت "$(a \vee b)$" استفاده می‌شود. در مرحله بعد، عملگر or دوم برای محاسبه عبارت "$(c \vee d)$" به کار می‌رود و سپس عملگر nor برای محاسبه عکس این عبارت استفاده می‌شود. در نهایت، با استفاده از عملگر ،and نتیجه نهایی محاسبه می‌شود. برای چاپ نتیجه نهایی، از دستورات ،addi add و syscall استفاده می‌شود. با استفاده از دستور ،add مقدار نتیجه در $a۰ قرار داده می‌شود. سپس با استفاده از دستور ،addi کد سیستمی را برای چاپ نتیجه به ۱ تنظیم می‌کنیم. در نهایت با فراخوانی دستور ،syscall نتیجه نهایی بر روی ترمینال چاپ می‌شود. در انتها، با استفاده

از دستورات addi و syscall برنامه به اتمام می‌رسد.

```
# This section is for loading the input values
.data
a: .word 0
b: .word 1
c: .word 1
d: .word 0

.text
main:
    # Load input values
    lw $t0, a
    lw $t1, b
    lw $t2, c
    lw $t3, d

    # Compute (a  b) using the or instruction
    or $t4, $t0, $t1

    # Compute (c  d) using the or instruction
    or $t5, $t2, $t3

    # Compute the negation of (c  d) using the nor instruction
    nor $t6, $t5, $zero

    # Compute the final result using the and instruction
    and $t7, $t4, $t6

    # Output the result to the terminal
    add $a0, $t7, $zero
    addi $v0, $zero, 1
    syscall

    # Exit the program
    addi $v0, $zero, 10
    syscall
```

Listing ۶: Logical Expression Two

۳.۴ عبارت سوم

$$\neg(a \oplus b) \wedge (c \oplus d)$$

پاسخ:

ابتدا مقادیری که در بخش داده‌ای تعریف شده‌اند به متغیرهای مختلفی که در رجیسترهای MIPS قرار می‌گیرند، لود می‌شوند. سپس با استفاده از دستور XOR دو مقدار a و b با هم XOR می‌شوند و نتیجه در رجیستر $t۲$ ذخیره می‌شود. سپس با استفاده از دستور NOR، بیت‌های مقدار $t۲ اگر یک باشند به صفر تبدیل شده و اگر صفر باشند به یک تبدیل (not) می‌شوند. سپس با دستور ADDI یک به نتیجه برگردانده شده توسط دستور NOR اضافه شده و در نهایت با استفاده از دستور ANDI، بیت‌های این مقدار با مقدار ۱ آند شده و به این صورت بیت‌های آن نوشته می‌شوند. این عمل برای محاسبه $\neg(a \oplus b)$ استفاده می‌شود. در

مرحله بعد، دو مقدار c و d با دستور XOR با یکدیگر XOR می‌شوند و نتیجه در رجیستر $t۶ ذخیره می‌شود. در نهایت، با استفاده از دستور AND، نتیجه‌ی $\neg(a \oplus b)$ را با مقدار $(c \oplus d)$ عمل AND می‌شود و نتیجه در رجیستر $t۷ ذخیره می‌شود. سپس با استفاده از دستور ADD، مقدار $t۷ به $a۰ منتقل شده و سپس با استفاده از دستور ADDI، کد سیستمی به ۱ تغییر داده می‌شود تا برنامه عدد موجود در $a۰ را به صفحه نمایش بفرستد. در نهایت با استفاده از دستور ADDI و مقدار ۱۰ به کد سیستمی دیگر، یعنی خروج از برنامه تغییر داده می‌شود تا برنامه پس از نمایش عدد مورد نظر، خاتمه یابد.

```
.data
a: .word 0x8       # initialize a to 1000 binary
b: .word 0x3       # initialize b to 0011 binary
c: .word 0x5       # initialize c to 0101 binary
d: .word 0xA       # initialize d to 1010 binary

.text
main:
    lw $t0, a          # load a into $t0
    lw $t1, b          # load b into $t1
    xor $t2, $t0, $t1 # compute a xor b and store the result in $t2

    # negate the result of a xor b
    nor $t3, $t2, $zero  # invert the bits of $t2
    addi $t3, $t3, 1    # add 1 to $t3
    andi $t3, $t3, 0x1  # set $t3 to the least significant bit of $t3

    lw $t4, c          # load c into $t4
    lw $t5, d          # load d into $t5
    xor $t6, $t4, $t5 # compute c xor d and store the result in $t6

    # compute the final result
    and $t7, $t3, $t6

    # print the result to the terminal
    add $a0, $t7, $zero  # move the result to $a0, which holds the argument for the system call
    addi $v0, $zero, 1  # set the system call code to print an integer
    syscall              # print the result to the terminal

    # exit the program
    addi $v0, $zero, 10 # set the system call code to exit the program
    syscall              # exit the program
```

Listing ۷: Logical Expression Three