به نام خدا

پروژه درس ساختار و زبان ماشین

استاد : دکتر سربازی

شماره گروه : ٦

اعضای گروه : ایمان محمدی – سروش شرافت بفروئی – سهیل نظری مندجین – شایان صالحی

ترم بهار ۰۰-۰۱

۱) برای حل این مسئله با استفاده از قوائد ماتریس(جمع زدن ، ضرب کردن و جا به جا کردن ستون یا سطرها ...) ماتریس مورد نظر را به یک ماتریس همانی I به دست میاوریم و با اعمال همان تغییرات روی یک ماتریس همانی تبدیل به ماتریس وارون ما خواهد شد! کد ۸۰۸۶ و میپس این سوال زده شده به علت اختصار فقط کد میپس را بررسی میکنیم.

حال قسمت های کد میپس را بررسی میکنیم!

```
.data
enter: .ascii "\n"
space: .ascii " "

.macro exit
    li $v0, 10
    syscall
.end_macro


.macro push (%arg)
addi $sp,$sp,-4
sw %arg,0($sp)
.end_macro
.macro push_m (%a1,%a2,%a3)
push(%a1)
push(%a2)
push(%a3)
.end_macro


.macro pop(%arg)
lw %arg,0($sp)
addi $sp,$sp,4
.end_macro

.macro pop_m(%a1,%a2,%a3)
pop(%a1)
pop(%a2)
pop(%a3)
.end_macro


.macro element_pos (%row,%column) #get row and column, return position byte in v0
mul $v0,%row,$s0
add $v0,$v0,%column
sll $v0,$v0,2
.end_macro
```

در این قسمت تکه های ماکرویی برای سهولت کار در کد اصلی تعریف کرده exit برای خروج و قطع شدن پروگرم و پوش و پاپ برای افزودن و برداشتن از استک موردنظر و المنت پوز برای برگرداندن محتوای مختصات دریافتی!

```
.macro load_element(%array,%row,%column) #get row and column and array and load word in v0
element_pos (%row,%column)
add $v0,$v0,%array
lw $v0,0($v0)
.end_macro

.macro load_element_into(%array,%row,%column,%dest) #get row and column and array and load word in %dest
element_pos (%row,%column)
add $v0,$v0,%array
lw %dest,0($v0)
.end_macro

.macro store_element(%array,%row,%column) #store a0 into position
element_pos(%row,%column)
add $v0,$v0,%array
sw $a0,0($v0)
.end_macro

.macro print(%matrix)
push($t0)
push($t1)
move $t0,$zero #row
fori:
    move $t1,$zero #column
    forj:
        load_element(%matrix,$t0,$t1)
        move $a0,$v0
        li $v0,1
        syscall
        lb $a0,space
        li $v0,11
        syscall
        addi $t1,$t1,1
        bne  $t1,$s0,forj
    lb $a0,enter
    li $v0,11
    syscall
    addi $t0,$t0,1
    bne $t0,$s0,fori
lb $a0,enter
li $v0,11
syscall
```

این قسمت نیز همانند قسمت قبل است. ماکروهای تعریف شده برای سهولت کار استفاده میکنیم.

در لود المنت ها خانه ۰ اررِی ورودی را در جای ۷۰ ذخیره میکند.

در استور المنت برعکس این کار را داریم.

در ماکرو پرینت ابتدا مقادیر داخل رجیستر t۰ و t۱را داخل استک ذخیره کرده و و در ادامه دو فور روی ماتریس ذخیره شده زده رو آن پیمایش کرده و آن ها را دونه به دونه چاپ می کند.

```
.macro muladd_row(%matrix,%f_row,%s_row,%mul) # f_row <- f_row + s_row*mul
push_m($t0,$t1,$t2)
push($t3)
push($t4)
move $t2,%f_row
move $t3,%s_row
move $t4,%mul
move $t0,$zero
fori:
    load_element(%matrix,$t2,$t0)
    move $t1,$v0
    load_element(%matrix,$t3,$t0)
    mul $v0,$v0,$t4
    add $a0,$t1,$v0
    store_element(%matrix,$t2,$t0)
    addi $t0,$t0,1
    bne $t0,$s0,fori
print(%matrix)
pop($t4)
pop($t3)
pop_m($t2,$t1,$t0)
.end_macro


# i represents immediate data
.macro muladd_rowi(%matrix,%f_rowi,%s_rowi,%muli) # f_rowi <- f_rowi + s_rowi*muli
push_m($t7,$t6,$t5)
li $t7,%f_rowi
li $t6,%s_rowi
li $t5,%muli
muladd_row(%matrix,$t7,$t6,$t5)
pop_m($t5,$t6,$t7)
.end_macro

.macro mul_row(%matrix,%row,%mul) # row <- row*mul
push(%mul)
addi %mul,%mul,-1
muladd_row(%matrix,%row,%row,%mul)
pop(%mul)
.end macro
```

در این بخش از ماکروها اعمال و خواص ماتریس را پیاده سازی میکنیم

```
.text
main:
li $v0,5
syscall             #read n in s0

move $s0,$v0        # s0 <- n
mul $a0,$s0,$s0         # s1 <- n^2
sll $a0,$a0,2       # s1 <- (n^2)*4

li $v0,9
syscall
move $s1,$v0        # allocating memory in s1 for matrix

li $v0,9
syscall         #allocating memory in s2 for identity marix
move $s2,$v0

read_matrix:
move $t0,$zero #t0 <- row           reading matrix elements
fori:
    move $t1,$zero #t1 <- column
    forj:
        li $v0,5
        syscall
        move $a0,$v0 #read aij
        store_element($s1,$t0,$t1) # build in matrix
        li $a0,0
        bne $t0,$t1,not_diameter
        li $a0,1
        not_diameter:
        store_element($s2,$t0,$t1) #build identity matrix
        addi $t1,$t1,1
        bne  $t1,$s0,forj
    addi $t0,$t0,1
    bne $t0,$s0,fori
```

در این بخش اصلی کد مین ابتدا ورودی ها را لود کرده و داخل ماتریس گذاشته سپس با استفاده از خواص ماتریس ها و روش جردن-گاوس وارون ماتریس را در نهایت به دست میاوریم. در به دست اوردن وارون ماتریس از خواص ماتریس استفاده میکنیم که برای ان نیاز به توابعی مانند پیدا کردن gcd اعداد داشته که آن ها را نیز به صورت دسترسی near پیاده سازی میکنیم.

```
make_matrix_paiin_mosalasi:
move $t0,$zero      #t0 <- column:0~n-1
sub $t2,$s0,1
pm_fori:
    addi $t1,$t0,1      #t1 <- row:i+1 ~ n-1
    pm_forj:
        move $a0,$t1
        move $a1,$t0
        jal make_element_zero
        addi $t1,$t1,1
        bne $t1,$s0,pm_forj
    addi $t0,$t0,1
    bne $t0,$t2,pm_fori
exit_pm:


make_matrix_bala_mosalasi:
subi $t0,$s0,1   #t0 <- column:n-1~1
bm_fori:
    subi $t1,$t0,1   #t1 <- row: i-1~0
    bm_forj:
        move $a0,$t1
        move $a1,$t0
        jal make_element_zero
        subi $t1,$t1,1
        bne $t1,-1,bm_forj
    subi $t0,$t0,1
    bne $t0,$zero,bm_fori

exit
```

در این بخش دو تابع پیاده سازی شده برای بالا مثلثی و پایین مثلثی کردن ماتریس ها داریم.

۲) در این برنامه دو عدد به فرمت قطبی میگیریم و به کمک بسط سری تیلور یک عبارت ریاضی را محاسبه میکنیم پیاده سازی عملگر های ریاضی مهم ترین بخش پیاده سازی این برنامه بوده است. این سوال به دو زبان میپس و ۸۰۸۶ پیاده سازی شده است.

بخش های اصلی کد میپس

```
.text

  li $v0, 4
  la $a0, t
  syscall #print t

# Getting user input
  li $v0, 5
  syscall
  move $s0,$v0

  li $v0, 5
  syscall
  move $s1,$v0

  li $v0, 5
  syscall
  move $s2,$v0

  li $v0, 5
  syscall
  move $s3,$v0

 mul $t0,$s0,$s2
 mul $t1,$s1,$s3
 add $t0,$t0,$t1  #t0=a1a2+b1b2
 mul $t1,$s1,$s2
 mul $t2,$s3,$s0
 sub $t1,$t1,$t2  #t1=b1a2-a1b2
 mul $t2,$s2,$s2
 mul $t3,$s3,$s3
 add $t2,$t2,$t3 #t2=s2*s2+s3*s3

 mtc1 $t0, $f12
 cvt.s.w $f12, $f12 #t0=f12

 mtc1 $t1, $f11
 cvt.s.w $f11, $f11 #t1=f11

 mtc1 $t2, $f10
 cvt.s.w $f10, $f10 #t2=f10


 div.s $f0,$f12,$f10 #f0=t0/t2
 div.s $f1,$f11,$f10 #f1=t1/t2
 mul.s $f3,$f0,$f0
 mul.s $f4,$f1,$f1
 add.s $f3,$f3,$f4 #f3=t0^2+t1^2
 sqrt.s $f3,$f3  #=r

 div.s $f0,$f1,$f0 #f0=f1/f0=x
```

```
   la $t0,z
   lwc1 $f1,0($t0) #f1=0=res

   la $t0,one
   lwc1 $f2,0($t0) #f2=1=neg

   la $t0,mo
   lwc1 $f30,0($t0) #f30=-1

   la $t0,two
   lwc1 $f28,0($t0) #f28=2

   mov.s $f4,$f0 #f4=x=power
   mov.s $f5,$f2 #f5=1=div

   li $s3,0 #counter
arctan_loop:
   beq $s3,100,end
   mul.s $f7,$f2,$f4
   div.s $f8,$f7,$f5
   add.s $f1,$f1,$f8 #res+=neg*power/div

   mul.s $f2,$f2,$f30 #neg*=-1

   mul.s $f29,$f0,$f0

   mul.s $f4,$f4,$f29   #power*=x^2

   add.s $f5,$f5,$f28  #div+=2

   add $s3,$s3,1   #counter++

   j arctan_loop #loop
end:

   la $s0,dr
   swc1 $f3,0($s0) #store r

   la $s0,db
   swc1 $f1,0($s0) #store beta


  li $v0, 2
  mov.s $f12, $f3
  syscall

  li $v0, 4
   la $a0, s
   syscall #print t

  li $v0, 2
  mov.s $f12, $f1
  syscall
```

پیاده سازی عملگر ارکتانژانت

پیاده سازی ۸۰۸۶:

```
ORG    100h

    call scan_num
    mov x , cx
    call scan_num
    mov y , cx
    call scan_num
    mov xp , cx
    call scan_num
    mov yp , cx
    mov ax , x
    mov bx , x
    imul bx
    mov s_low , ax
    mov s_high , dx
    mov ax , y
    mov bx , y
    imul bx
    add s_low , ax
    add s_high , dx
    mov ax , xp
    mov bx , xp
    imul bx
    mov m_low , ax
    mov m_high , dx
    mov ax , yp
    mov bx , yp
    imul bx
    add m_low , ax
    add m_high , dx
    mov ax , s_low
    mov dx , s_high
    mov bx , m_low
    idiv bx
    call sqrt
    mov z , cx
    mov ax , x
    mov bx , yp
    imul bx
    mov s_low , ax
    mov s_high , dx
    mov ax , y
    mov bx , xp
    imul bx
    sub s_low , ax
    sub s_high , dx
    mov ax , x
    mov bx , xp
    imul bx
    mov m_low , ax
    mov m_high , dx
    mov ax , y
    mov bx , yp
    imul bx
    add m_low , ax
    add m_high , dx
    mov dx , s_high
    mov ax , s_low
    mov bx , m_low
    idiv bx
    call arctg
    mov r , bx
    ; z and r are ready to print!
;    call print_num_uns
```

```
RET
x dw 0
y dw 0
xp dw 0
yp dw 0
z dw 0
r dw 0
s_low dw 0
s_high dw 0
m_low dw 0
m_high dw 0

sqrt proc
    mov cx , 0000
    mov bx , -1
    lable:
        add bx , 02
        inc cx
        sub ax , bx
        jns lable
        ret
endp

arctg proc
    push ax
    mov cx , 10
    mov bx , 0
    mov si , 3
    mov di , -1
    imul ax
    imul di
    mov di , ax
    pop ax
    loop_lable:
        push ax
        imul di
        idiv si
        add bx , ax
        imul si
        add si , 2
        loop loop_lable
    ret

endp
DEFINE_SCAN_NUM
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ; required for print_num.

END
```

۳) با ورودی گرفتن چندجمله ای به صورت ضرایب با استفاده از روش نیوتون-رافسون ریشه آن را محاسبه کرده و در صورت نداشتن هم آن را چاپ خواهیم کرد این کد به تنها به صورت میپس پیاده سازی گردیده است.

```
.data

    text:  .asciiz "Enter a number: "
    array: .float 400
    array1: .float 400
    array2: .float 400
    array3: .float 400
    array4: .float 400
    array5: .float 400
    array6: .float 400
    array7: .float 400
    array8: .float 400
    array9: .float 400
    array10: .float 400
    array11: .float 400
    array12: .float 400
    array13: .float 400
    array14: .float 400

    one: .float 1.0
    dx: .float 0.0001
    s:  .float 0.0
    nope:.asciiz "\nthere is no root "


.text
```

```
main:


   # Printing out the text
    li $v0, 4
    la $a0, text
    syscall

    # Getting user input
    li $v0, 5
    syscall
    la $t0,array

    # Moving the integer input to another register
    move $s0, $v0  #s0=n
    move $s1,$s0  #counter
l1: beq $s1,-1,end_input
    li $v0, 6
    syscall
    swc1 $f0,($t0)
    addi $t0,$t0,4
    addi $s1,$s1,-1
    j l1

end_input:
la $s4,s       #f0=x=0
lwc1 $f0,0($s4)


jal get_result
la $s3,dx
lwc1 $f8,0($s3) #f8=dx

li $s5,0  #counter
cal:
abs.s $f5,$f4
c.le.s $f5,$f8 #if result<=dx jump to l3
bc1t print_root
mov.s $f6,$f4 #res
add.s $f0,$f0,$f8 #point=x+dx
jal get_result
sub.s $f9,$f4,$f6
div.s $f9,$f9,$f8   #f9=dy/dx
div.s $f6,$f6,$f9   #f6=res/D
sub.s $f0,$f0,$f8
sub.s $f0, $f0,$f6 #point-= res/D
jal get_result
addi $s5,$s5,1
beq $s5,200,no_ans
j cal

no_ans:
# Printing out the text
    li $v0, 4
    la $a0, nope
    syscall
    j terminate
print_root:
li $v0, 2
mov.s $f12, $f0   # Move contents of register $f3 to register $f12
syscall

terminate:
li $v0,10
syscall
```

۴) در این سوال یک ماشین حساب را پیاده سازی میکنیم. اعمال ریاضی جمع ضرب تفریق تقسیم پیاده سازی شده اند این سوال را با استفاده از الگوریتم پولیش نوتیشن برای اجرای دقیق و راحت تر عملیات های ریاضی حل کرده ایم. خطاهای سرریز اشتباهات در وارد کردن عملیات ریاضی در ورودی را نیز باید شناسایی میکردیم. این سوال تنها به زبان ۸۰۸۶ پیاده سازی شده.

```
    mulordiv:
        cmp sp, bp
        jge emptystack
        pop cx
        push cx
        cmp cl, "*"
        je popop
        cmp cl, "/"
        je popop

    popop:
        pop cx
        xchg ax, cx
        stosb
        xchg ax, cx
        jmp op

    emptystack:
        cmp al, "("
        je parantheses_err
        push ax
        mov al, " "
        stosb
        jmp getchar

    leftp:
        push ax
        jmp getchar

    rightp:
        cmp sp, bp
        jge parantheses_err
        pop cx
        push cx
        cmp cl, "("
        je matchedp
        pop cx
        xchg ax,cx
        stosb
        xchg ax,cx
        jmp rightp

    matchedp:
        pop cx
        jmp getchar


    endstr:
        cmp sp, bp
        jge stackend
        pop ax
        cmp al, "("
        je parantheses_err
        stosb
        jmp endstr

    stackend:
        mov dx, offset rpn
        ret

convert2rpn endp


evaluaterpn proc near

        mov si, offset rpn
```

```
cseg    segment

        assume  cs:cseg, ds:dseg,ss:sseg
    main:
        mov ax, dseg
        mov ds, ax
        mov es, ax
        mov ax, sseg
        mov ss, ax
        mov sp, offset words + 98
        mov bp, sp
        call convert2rpn
        mov dx, offset rpn
        call evaluaterpn
        mov bx, value

        mov ax, 4C00h
        int 21h


    convert2rpn proc near ; shunting-yard algorithm
        mov bp, sp
        mov si, offset evalstr
        mov di, offset rpn
        cld

    getchar:
        lodsb
        cmp al, "$"
        je endstr
        cmp al, 30h
        jb porop
        cmp al, 39h
        ja porop
        stosb
        jmp getchar

    porop:
        cmp al, "("
        je leftp
        cmp al, ")"
        je rightp
    op:
        cmp al, "+"
        je addorsub
        cmp al, "-"
        je addorsub
        cmp al, "*"
        je mulordiv
        cmp al, "/"
        je mulordiv
        jmp invalidstr_err

    addorsub:
        cmp sp, bp
        jge emptystack
        pop cx
        push cx
        cmp cl, "+"
        je popop
        cmp cl, "-"
        je popop
```

۵)در این برنامه می خواهیم با پیاده سازی تابع فیبوناچی مقدار y را بیابیم

$$y(x) = fact(x) - fib(x),$$
$$fact(x) = x * fact(x-1), fact(0) = 1,$$
$$fib(x) = fib(x-1) + fib(x-2), fib(1) = fib(2) = 1$$

print : تابع

در ابتدا cx و dx را برابر با صفر قرار میدهم. سپس چک میکنیم آیا مقدار ax ورودی برابر با صفر هست یا نه. اگه صفر بود مستقیم مقدار صفر را چاپ میکنیم. تا وقتی مقدار ax برابر با صفر شود تقسیم بر ده میکنیم و رقم به رقم در استک ذخیره میکنیم. تعداد ارقام در cx ذخیره میکنیم تا بدونیم تا چند مرتبه استک ارقام را که در استک از طریق dxذخیره کردیم پرینت کنیم. برای قرار دادن مقدار dx با صفر آن را با خودش ایکسور میکنیم زیرا dx باید هنگام انجام عمل div برابر با صفر باشد. برای اینکه مقادیر رو بتونیم پرینت کنیم با h۴۸ جمع میکنیم تا بتونیم مقدار آن را به صورت عدد پرینت کنیم زیرا اسکی کد زیرو برای با d۵۶ است.

facto: تابع

در ابتدا مقادیر bp و si و cx و dx و ax و di را در تابع تو استک ذخیره کرده و سپس هنگام خروج از تابع از استک پاپ میکنیم. در این تابع بازگشتی در si آدرس رو ذخیره کرده و در cx مقدار n که ورودی تابع هست ذخیره میکنیم. مقایسه میکنیم cx با یک برابر است که بیس کیس تابع بازگشتی ماست. در فور lop مقدار را در ax ذخیره میکنه و با cwd ورد رو به دابل تغییر می دهیم. در این تابع مقدار(fact(x - ۱ در ax ذخیره شده و مقدار x در cx ذخیره می شود و mul cx و x * (fact(x- ۱ در ax ذخیره میکند.

fibo: تابع

در ابتدا مقادیر bp و si و cx و dx و ax و di را در تابع داخل استک ذخیره کرده و و آدرس استک(sp) در (bp) ذخیره میکنیم.مقدار n را در cx از استک لود کرده و مقدار تابع fib را در si ذخیره میکنیم. اگر مقدار cx برابر دو شود به بیس کیس میرسیم. در لوپ ریکرسیو دوبار تابع فیبوناچی کال میشود. یه بار fibo(n-۱) بعد از یک مرتبه dec کردن cx و یه بار هم fibo(n- ۲) را بعد از دومین مرتبه dec کردن cx حساب میکنیم.

در ابتدای کد اول مقدار تابع fac را در ax موو کرده و و سپس مقدار sub ax, fib باعث مقدار نهایی یعنی y(x) که برابر با fact(x) – fib(x) می شود.

```
FACT PROC
    push bp
    push cx
    push dx
    push ax
    push si
    push di
    mov bp,sp
    mov cx,word ptr[bp+14] ;n
    mov si,word ptr[bp+16] ;fac add
    mov ax,1
    mov dx,0
    cmp cx,1
    jnz lop
    pop di
    pop si
    pop ax
    pop dx
    pop cx
    pop bp
    ret
lop:
    mov ax,word ptr[si]
    cwd
    mul cx
    mov word ptr[si],ax
    push si
    dec cx
    push cx
    call FACT
    pop trash
    pop trash
    pop di
    pop si
    pop ax
    pop dx
    pop cx
    pop bp
    ret
FACT ENDP

fibo PROC

    push bp
    push cx
    push ax
    push si
    push di
    mov bp,sp
```

```
start:  MOV AX,@DATA
  MOV DS,AX
  mov fib,0
  push offset fib
  push n
  call fibo

  push offset fac
  push n
  call FACT

  mov ax,fac
  sub ax,fib


  CALL PRINT


  ;interrupt to exit
  MOV AH,4CH
  INT 21H


PRINT PROC
```

٦) در این برنامه صرفا کافی است انتخابn از m+n را محاسبه کنیم که .

n را ورودی می گیریم و در حافظه ذخیره می کنیم. سپس به کمک فراخوانی زیرروال fact که فاکتوریل ورودی را محاسبه می کند $n!$ را محاسبه می کنیم.
پس از آن همین کار را برای m هم میکنیم. یعنی آن را ورودی می گیریم، در حافظه ذخیره می کنیم و به کمک زیرروال fact مقدار $m!$ را محاسبه می کنیم.

در نهایت با صدا زدن زیرروال fact برای $n+m$ مقدار $(n+m)!$ را محاسبه می کنیم.

```
call scan_num
mov n,cx
mov bx,n
call fact
mov n_high,dx
mov n_low,ax
call scan_num
mov m,cx
mov bx,m
call fact
mov m_high,dx
mov m_low,ax
mov bx,n
add bx,m
call fact
mov sum_high,dx
mov sum_low,ax
mov bx,n_low
div bx
mov bx,m_low
div bx
call print_num_uns

RET
```

در نهایت با یک بار تقسیم کردن $(n+m)!$ بر $n!$ و تقسیم کردن حاصل بر $m!$ پاسخ نهایی را محاسبه کرده و در خروجی چاپ می کنیم.

نحوه کارکرد تابع بازگشتی *fact* به شکل زیر است:

```
fact proc
    push bx
    cmp bx, 0
    ja end_fact
    dec bx
    call fact
    inc bx
    mul bx
    jmp fact_ret
    end_fact:
        mov ax, 1
    fact_ret:
        pop bx
        ret
endp

DEFINE_SCAN_NUM
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS

END
```

۷)در این کد سه رشته ورودی گرفته و میخواهیم ابتدا تعداد تکرار های رشته a را در رشته x پیدا کرده و اگر صفر نبود رشته های پیدا شده a را با رشتهb تعویض میکنیم. ورودی های گرفته شده ما نیز کیس سنسیتیو نیستند.

این سوال به زبان های ibm و mipsو ۸۰۸۶ پیاده سازی شده است.

```
;proc
countReplace proc
        lea     di,strA
        lea     si,strX
        lea     bx,strB
        lea     bp,strNew
        mov     last,si
        mov     ah,0
    loop1:
        cmp     [di], ah
        jz      found
        cmp     [si],ah
        jz      finished
        mov     ch,[di]
        cmp     [si],ch
        jz      continue
        sub     ch, 20h
        cmp     [si],ch
        jz      continue
        add     ch,40h
        cmp     [si],ch
        jnz     notMatched
    continue:
        inc     di
        inc     si
        jmp     loop1
    found:
        inc     num
        lea     di,strA
        lea     bx,strB
        mov     last,si
    loop2:
        cmp     [bx], ah
        jz      loop1
        mov     ch, [bx]
        mov     [bp],ch
        inc     bx
        inc     bp
        jmp     loop2
    notMatched:
        lea     di,strA
        mov     si,last
        mov     ch,[si]
        mov     [bp],ch
        inc     bp
        inc     si
        mov     last,si
        jmp     loop1
    finished:
        ret
endp


DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS
DEFINE_GET_STRING
END
```

```
ORG 100h

    lea     si,msg1
    call    print_string
    lea     di,strA
    mov     dx,buffsize
    call    get_string
    LEA     si, newln
    call    print_string
    lea     si,msg2
    call    print_string
    lea     di,strB
    mov     dx,buffsize
    call    get_string
    LEA     si, newln
    call    print_string
    lea     si,msg3
    call    print_string
    lea     di,strX
    mov     dx,buffsize
    call    get_string
    LEA     si, newln
    call    print_string
    call    countReplace
    mov     ax,num
    call    print_num_uns
    cmp     num, 0
    jz      done
    LEA     si, newln
    call    print_string
    lea     si,strNew
    call    print_string
    done:   RET


    ;data
    msg1    db    "Enter string a : ",0
    msg2    db    "Enter string b : ",0
    msg3    db    "Enter string x : ",0
    strA    db    100 dup(0)
    strB    db    100 dup(0)
    strX    db    100 dup(0)
    strNew  db    100 dup(0)
    last    dw    ?
    num     dw    0
    newln   db    13, 10, 0
    buffsize = $-buffer
```

```
1   Replace start 0
2                           ; START OF INIT
3   STM R14, R12, 12(R13)
4   BALR R12, R0
5   using *, R12
6   ST R13, Reg13
7   LA R13, RegSaveArea
8
9   XR R6,R6
10
11  LA R9,1
12  XR R10,R10
13  LA R1,0
14  lop:
15  LR R2,R1
16  XR R3,R3
17  chk:
18  XR R4,R4
19  XR R5,R5
20  IC R4,x(R2)
21  IC R5,a(R3)
22  CR R4,R5
23  BNZ bad
24  AR R2,R9
25  AR R3,R9
26  XR R4,R4
27  IC R4,a(R3)
28  CR R4,R10
29  BZ good
30  B chk
31
32  good:
33  L R4,cnt
34  AR R4,R9
35  ST R4,cnt
36  LR R1,R2
37
38  XR R4,R4
39  lop_add:
40  XR R5,R5
41  IC R5,b(R4)
42  STC R5,ans(R6)
43  AR R6,R9
44  AR R4,R9
45  XR R5,R5
46  IC R5,b(R4)
47  CR R5,R10
48  BNZ lop_add
49
50  B aft
51  bad:
52  XR R4,R4
53  IC R4,x(R1)
54  STC R4,ans(R6)
55  AR R6,R9
56  AR R1,R9
57
58
59  aft:
60  XR R4,R4
61  IC R4,x(R1)
62  CR R4,R10
63  BNZ lop
64
65
66
67  EXIT:
68  L  R13, Reg13
69  LM R14, R12, 12(R13)
70  BR R14
71
72  RegSaveArea DS 15 F
73  Reg13 DS F
74  x DC C'abcabc'
75  a DC C'ab'
76  b DC C'abc'
77  cnt DS F
78  ans DS 50C
79  end
```

کد ۸۰۸۶

سوال ۸)

در این سوال عددی در مبنای a ورودی گرفته و آن را به مبنای b می بریم فرض شده مبنا های زیر ده داده شده است. این سوال به زبان های ۸۰۸۶ و mips و ibm پیاده سازی شده است.

```
start:  MOV AX,@DATA
  MOV DS,AX

  push 10
  push offset temp
  push x
  push a

  call toTEN

  pop trash
  pop trash
  pop trash
  pop trash

  mov ax,temp
  mov x,ax
  mov temp,0

  push 10
  push offset temp
  push x
  push b

  call toX



  mov ax,temp

  CALL PRINT

  ;interrupt to exit
  MOV AH,4CH
  INT 21H


PRINT PROC
```

```
  ;initialize count
  mov cx,0
  mov dx,0
  label1:
    ; if ax is zero
    cmp ax,0
    je print1

    ;initialize bx to 10
    mov bx,10

    ; extract the last digit
    div bx

    ;push it in the stack
    push dx

    ;increment the count
    inc cx

    ;set dx to 0
    xor dx,dx
    jmp label1
  print1:
    ;check if count
    ;is greater than zero
    cmp cx,0
    je exit

    ;pop the top of stack
    pop dx

    ;add 48 so that it
    ;represents the ASCII
    ;value of digits
    add dx,48

    ;interrupt to print a
    ;character
    mov ah,02h
    int 21h

    ;decrease the count
    dec cx
    jmp print1
exit:
ret
PRINT ENDP
```

۹) این برنامه با استفاده از روش بازگشتی تعداد حالات قرار گرفتن n وزیر در یک صفحه ۹*۹ را بدون آنکه یکدیگر را تهدید کنند می یابیم.

```
Continue:
    addi $s4,$s4,1
    j Loop
AddAns:
    addi $s2,$s2,1
Exit:
    jr $ra
.globl check_cell
check_cell:
    move $t1,$a1
    move $t2,$a1
    la $v0,0
check_loop:
    addi $a0, $a0, -1
    addi $t1, $t1, -1
    addi $t2, $t2, 1
    beq $a0,-1,exit_check

    la $t3,4
    mult $t3,$a0
    mflo $t3
    add $t3, $s1, $t3

    lw $t4,0($t3)

    beq $t4,$a1,return_1

    beq $t4,$t1,return_1

    beq $t4,$t2,return_1

    j check_loop
return_1:
    la $v0,1
exit_check:
    jr $ra
.data
n:  .word 0
ans: .word 0
msgn:   .asciiz "Enter n: \n\r"
msgres: .asciiz "Answer is: "
```

```
Loop:
    beq $s4,$s0,Exit

    move $a0,$s3
    move $a1,$s4

    addi $sp,$sp, -4
    sw $ra,0($sp)

    jal check_cell

    lw $ra,0($sp)
    addi $sp, $sp, 4

    beq $v0,1,Continue

    la $t0, 4
    mult $t0,$s3
    mflo $t0
    add $t0, $t0, $s1
    sw $s4,0($t0)

    addi $sp,$sp, -12
    sw $s3,8($sp)
    sw $s4,4($sp)
    sw $ra,0($sp)

    addi $a0, $s3, 1
    jal place_queens

    lw $ra,0($sp)
    lw $s4,4($sp)
    lw $s3,8($sp)
    addi $sp, $sp, 12


    la $t0, 4
    mult $t0,$s3
    mflo $t0
```

۱۰)در این سوال تعدادی نقطه در فضای دوبعدی با اعداد صفر و یک برچسب خورده اند. میخواهیم با عبور خطی در این فضا، نقاط با برچسب های مختلف را از هم جدا کنیم. بدین صورت که در حالتِ ایده آل، همه نقاطِ بالای خط برچسب یک، و همه نقاطِ پایین خط برچسب صفر خورده باشند. بهترین خط، خطی است که کمترین تعداد نقطه را اشتباه برچسب بزند (یعنی کمترین تعداد صفرِ بالای خط و یک پایین خط داشته باشیم. بخشی از کد این سوال به زبان ای بی ام در زیر هست.

```
FIND_XY_ANSWER:

    LA      R10,arr
    LA      R5,0
LOOP_1_XY:
    LA      R6,0
    LA      R11,0
LOOP_2_XY:
    LR      R1,R5
        L       R15,=V(GET)
        BALR        R14,R15
    LR      R7,R2
    LR      R8,R3
    LR      R9,R4
    LR      R1,R6
    L       R15,=V(GET)
        BALR        R14,R15
    LA      R1,REG_SAVE
    STM         R2, R11, 0(R1)
    MR      R6,R3
    LR      R9,R8
    MR      R8,R2
    CR      R7,R9
    LM      R2,R11,0(R1)
    BL      LESS_XY
    BH      HIGH_XY
    LA      R11,1(R11)
    B       CONTINUE_XY
LESS_XY:
    C       R4,=F'0'
    BNE     CONTINUE_XY
    LA      R11,1(R11)
    B       CONTINUE_XY
HIGH_XY:
    C       R4,=F'1'
    BNE     CONTINUE_XY
    LA      R11,1(R11)
    B       CONTINUE_XY
CONTINUE_XY:
    LA      R6,1(R6)
    C       R6,N_INPUT
    BL      LOOP_2_XY
    C       R11,XY_BEST
    BL      LOOSE_XY
    ST      R11,XY_BEST
    ST      R7,XY_X_ANS
    ST      R8,XY_Y_ANS
LOOSE_XY:
    LA      R5,1(R5)
    C       R5,N_INPUT
    BL      LOOP_1_XY
```

```
L  R13,SAVEAREA_4
LM R14,R12,12(R13)
BR R14

N_INPUT         DC F'5'
Y_BEST      DC F'-100'
Y_ANS       DS F
X_BEST      DC F'-100'
X_ANS       DS F
XY_BEST     DC F'-100'
XY_X_ANS        DS F
XY_Y_ANS        DS F
arr     DC F'-1',F'0',F'0',F'-1',F'-1',F'0',F'-1',F'1',F'0',F'1',F'1',F'1',F'2',F'-1',F'1'
REG_SAVE        DS 10F
SAVEAREA    DS F
SAVEAREA_4  DS 17F

end


GET START       0
    STM             R14,R12,12(R13)
    BALR            R12,R0
    USING           *,R12
    ST      R13,SAVEAREA_4(R8)
    LA      R13,SAVEAREA(R8)


    LA      R7,3
    MR      R6,R1
    SLL     R7,2
    L       R2,0(R7,R10)
    L       R3,4(R7,R10)
    L       R4,8(R7,R10)
```

۱۱) در این سوال میخواهیم مکان بزرگترین بزرگترین زیر رشته پالیندروم را در یک رشته ورودی پیدا کنیم.

در دو حلقه تو در تو روی کلمه مورد نظر حرکت می کنیم. مثلا اگر شمارنده های این دو حلقه را i و j در نظر بگیریم هر بار بررسی می کنیم کلمه ای که از اندیس i تا اندیس j کلمه اصلی است، رشته آینه ای هست یا خیر.

اگر رشته آینه ای بود طول آن را با ماکسیموم طول رشته های آینه ای مقایسه می کینم. اگر بیشتر بود، اندیس ابتدایی آن را در مکان مورد نظر در حافظه ذخیره می کنیم و طول ماکسیموم رشته آینه ای را به روزرسانی می کنیم.

اگر رشته آینه ای نبود هم متغیر حلقه درونی را یک واحد به رسیدن به انتهای حلقه نزدیک می کنیم.

```
mirror PROC
    push bp
    push cx
    push dx
    push ax
    push si
    push di
    mov bp,sp
    mov si,word ptr[bp+14] ;start
    mov di,word ptr[bp+16] ;end
    mov cx,word ptr[bp+18] ;len
    mov ax,0
    ;isMirror offset
lop:
    cld
    lodsb
    mov dl,byte ptr[di]
    dec di
    cmp ax,dx
    jnz notequal
    loop lop
    mov si,word ptr[bp+20]
    mov word ptr[si],1
    pop di
    pop si
    pop ax
    pop dx
    pop cx
    pop bp
    ret
notequal:
    mov si,word ptr[bp+20]
    mov word ptr[si],0
    pop di
    pop si
    pop ax
    pop dx
    pop cx
    pop bp
    ret
mirror ENDP

END start
```

در نهایت در خروجی اندیس ابتدایی بزرگترین زیررشته آینه ای را چاپ می کنیم.

در زیرروال mirror صرفا چک می کنیم زیررشته ای ورودی داده شده است آینه ای است یا خیر.

از اول و آخر زیررشته شروع می کنیم ودر یک حلقه هر بار دو اندیس زیررشته را با یکدیگر مقایسه می کنیم و اگر یکی نبودند زیررشته آینه ای نبوده است و خروجی مورد نظر را برمی گردانیم.

اگر هم یکی بودند هر کدام از این دو را یک واحد به یکدیگر نزدیک می کنیم و دوباره مقایسه را انجام می دهیم تا این دو اندیس به یکدیگر برسند. اگر به هم رسیدند یعنی زیررشته آینه ای است و خروجی مورد نظر را برمی گردانیم.

۱۲) در این سوال n عدد را دور یک میچینیم. یک در میان اعداد را حذف میکنیم تا در نهایت یک عدد باقی بماند. ما باید برنامه ایی پیاده سازی کنیم که این عدد را پیدا کند. برای حل این سوال از الگوریتم ژوزفوس استفاده میکنیم که به صورت بازگشتی در هر مرحله این از الگوریتم به این شکل دوباره برنامه را صدا میزنیم

```
(Josephus (n - 1, 1)) % n + 1;
```

```
JOSEPHUS PROC
    push bp
    push dx
    push ax
    push si
    push di
    mov bp,sp
    mov si,word ptr[bp + 12]
    mov di,word ptr[si]
    cmp di,1
    jnz recrusive
    pop di
    pop si
    pop ax
    pop dx
    pop bp
    ret
recrusive:
    dec word ptr[si]
    push si
    CALL JOSEPHUS
    mov ax,word ptr [si]
    inc ax
    cwd
    div di
    inc dx
    mov word ptr[si],dx
    pop trash
    pop di
    pop si
    pop ax
    pop dx
    pop bp
    ret

JOSEPHUS ENDP
END start
```