بسمه تعالی

گزارش تمرین ۲ معماری کامپیوتر

۲۹ اسفند ۱۴۰۱

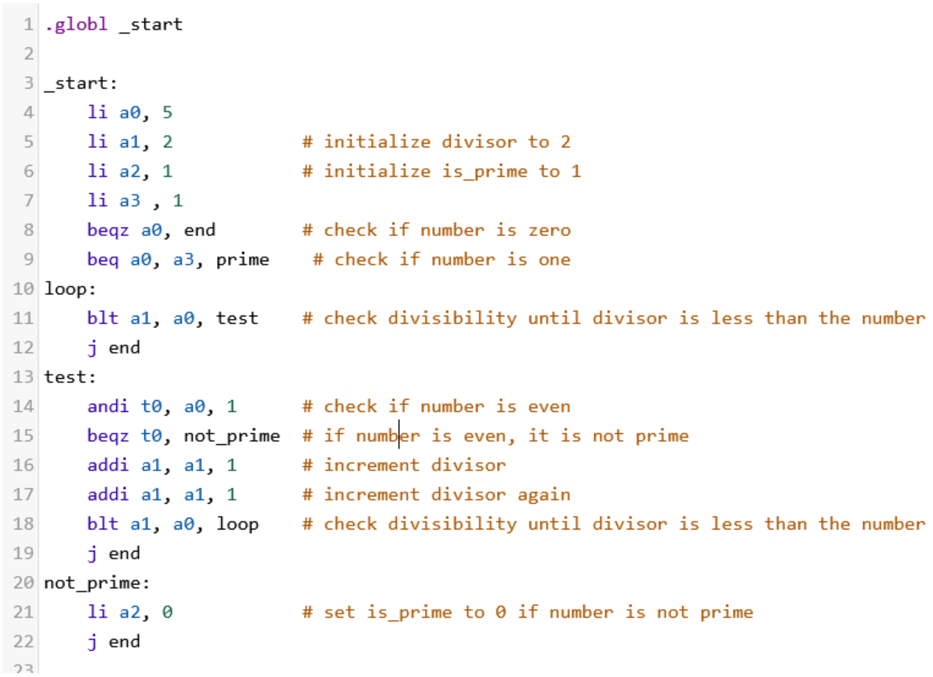
سوال ۱:

توضیحات:

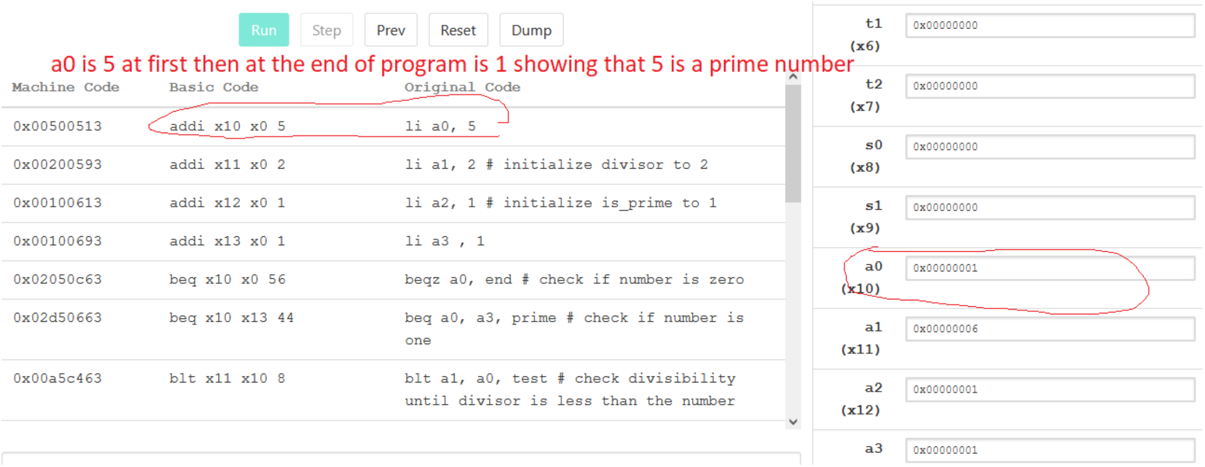
این برنامه یک مقسوم علیه *a*۱ به ۲ و یک پرچم *a*۲ را به ۱ (که عدد را اول فرض می کند) مقداردهی اولیه می کند. اگر عدد *a*۰ صفر باشد، برنامه بلافاصله پایان می یابد و ۰ را در *a*۰ ذخیره می کند.

اگر عدد *a*۰ یک باشد، برنامه بلافاصله پرچم را روی ۱ قرار می دهد (از آنجایی که ۱ اول اســت) و آن را در *a*۰ ذخیره می کند. این برنامه تقسیم پذیری را با افزایش ۲ تقسیم کننده بررسی می کند تا زمانی که به عدد برسد یا از آن عبور کند. اگر عدد بر هر مقسوم‌کننده‌ای قابل بخش باشد، پرچم روی ۰ تنظیم می‌شود (زیرا عـدد اول نیست) .هنگامی که بررسی تقسیم پذیری کامل شد، مقدار نهایی پرچم در *a*۰ ذخیره می شود.

کد و نتیجه:



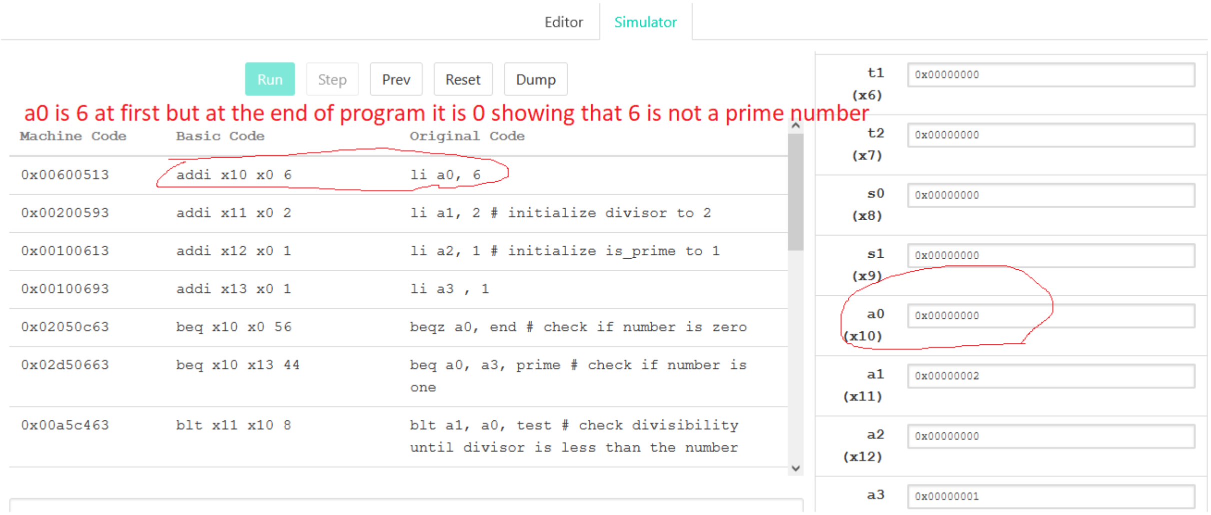
```
.globl _start

_start:
    li a0, 5
    li a1, 2            # initialize divisor to 2
    li a2, 1            # initialize is_prime to 1
    li a3 , 1
    beqz a0, end        # check if number is zero
    beq a0, a3, prime    # check if number is one
loop:
    blt a1, a0, test    # check divisibility until divisor is less than the number
    j end
test:
    andi t0, a0, 1      # check if number is even
    beqz t0, not_prime  # if number is even, it is not prime
    addi a1, a1, 1      # increment divisor
    addi a1, a1, 1      # increment divisor again
    blt a1, a0, loop    # check divisibility until divisor is less than the number
    j end
not_prime:
    li a2, 0            # set is_prime to 0 if number is not prime
    j end
```



a0 is 5 at first then at the end of program is 1 showing that 5 is a prime number

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	li a0, 5
0x00200593	addi x11 x0 2	li a1, 2 # initialize divisor to 2
0x00100613	addi x12 x0 1	li a2, 1 # initialize is_prime to 1
0x00100693	addi x13 x0 1	li a3 , 1
0x02050c63	beq x10 x0 56	beqz a0, end # check if number is zero
0x02d50663	beq x10 x13 44	beq a0, a3, prime # check if number is one
0x00a5c463	blt x11 x10 8	blt a1, a0, test # check divisibility until divisor is less than the number

t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x00000001
a1 (x11)	0x00000006
a2 (x12)	0x00000001
a3	0x00000001



a0 is 6 at first but at the end of program it is 0 showing that 6 is not a prime number

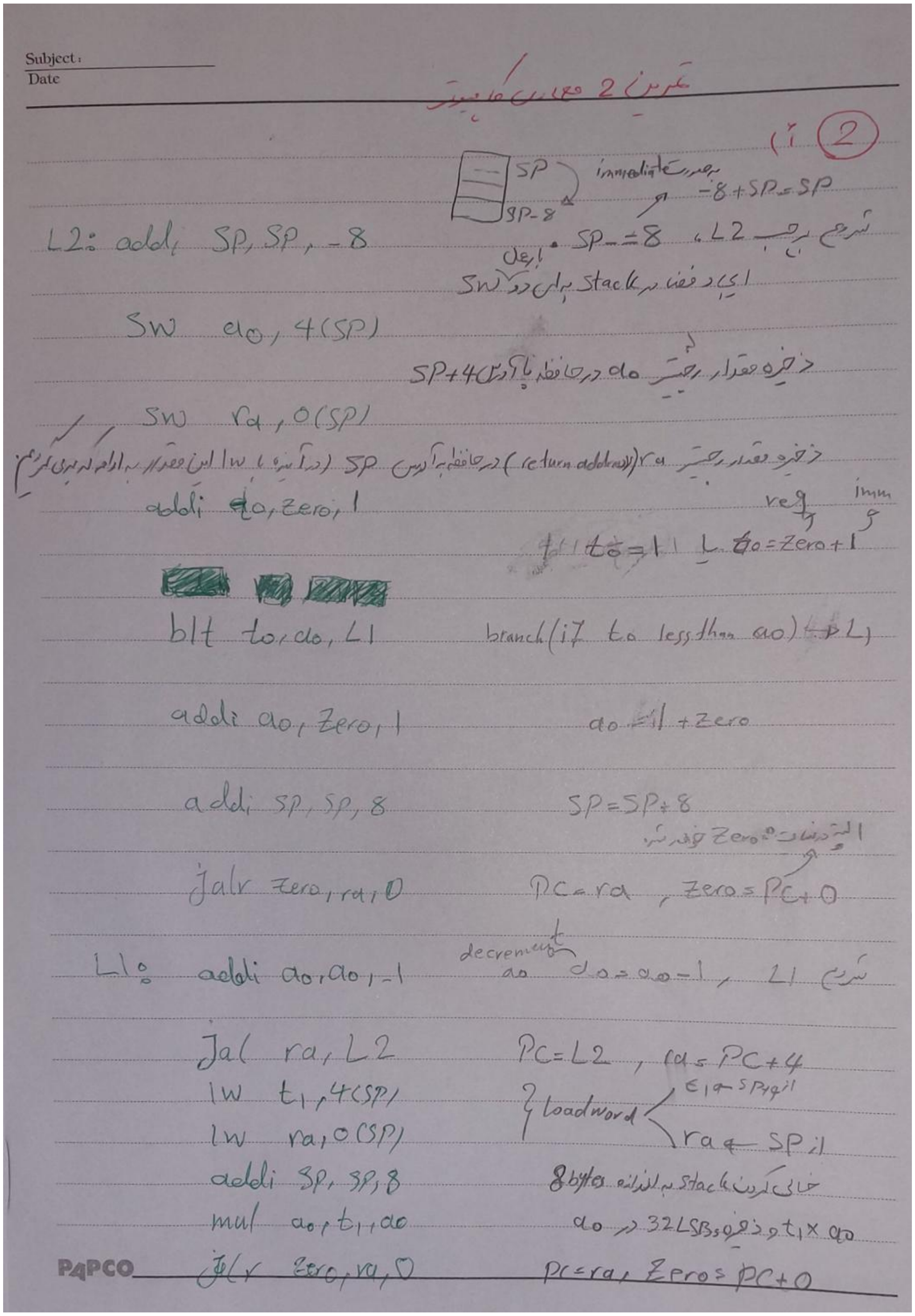
Machine Code	Basic Code	Original Code
0x00600513	addi x10 x0 6	li a0, 6
0x00200593	addi x11 x0 2	li a1, 2 # initialize divisor to 2
0x00100613	addi x12 x0 1	li a2, 1 # initialize is_prime to 1
0x00100693	addi x13 x0 1	li a3 , 1
0x02050c63	beq x10 x0 56	beqz a0, end # check if number is zero
0x02d50663	beq x10 x13 44	beq a0, a3, prime # check if number is one
0x00a5c463	blt x11 x10 8	blt a1, a0, test # check divisibility until divisor is less than the number

t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x00000000
a1 (x11)	0x00000002
a2 (x12)	0x00000000
a3	0x00000001

سوال ۲:

توضیحات:

الف)



Subject:
Date

تمرین 2 معماری کامپیوتر

2 آ)

بصورت immediate
-8+SP=SP

L2: addi SP, SP, -8

شروع برچسب L2 ، SP-=8 .
با عمل
ایجاد فضا در stack برای دو sw

sw a0, 4(SP)

ذخیره مقدار رجیستر a0 در حافظه با آدرس SP+4

sw ra, 0(SP)

ذخیره مقدار رجیستر ra (return address) در حافظه با آدرس SP (در آینده با lw این مقدار را به ادامه برمی‌گردیم)

addi ~~t0~~ a0, zero, 1

reg و imm

~~t0=1~~ یا ~~t0~~ a0=zero+1

[redacted]

blt t0, a0, L1

branch(if t0 less than a0) → L1

addi a0, Zero, 1

a0 = 1 + zero

addi SP, SP, 8

SP=SP+8

البته در نهایت Zero صفر شد

jalr Zero, ra, 0

PC=ra , Zero=PC+0

L1: addi a0, a0, -1

شروع L1 ، a0=a0-1 ، decrement a0

jal ra, L2

PC=L2 , ra=PC+4

lw t1, 4(SP)

lw ra, 0(SP)

loadword: از SP+4 ← t1 ، از SP ← ra

addi SP, SP, 8

خالی کردن stack به اندازه 8 bytes

mul a0, t1, a0

a0 × t1 و ذخیره 32LSBs در a0

jalr Zero, ra, 0

pc=ra, Zero=PC+0

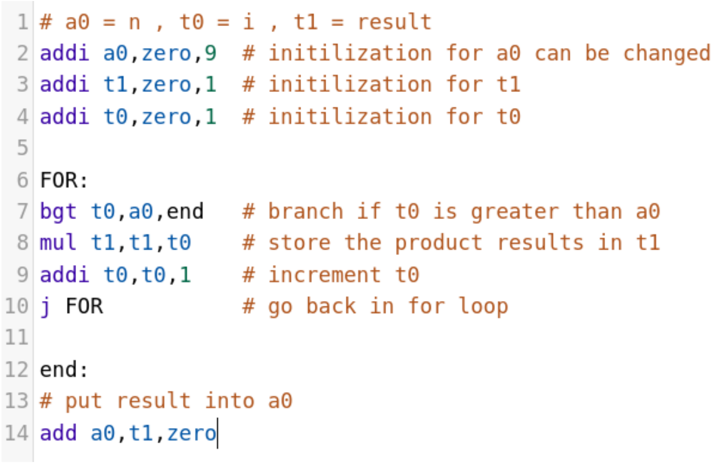
PAPCO

ب)

عملکرد برنامه به دست آوردن و ذخیره *a*۰ فاکتوریل در *a*۰ می‌باشد. به این صورت که *a*۰ در ابتدا در *stack* ذخیره و سپس *decrement* می‌شود و این رفتار تکرار می‌شود تا *a*۰ برابر ۱ شود. (مقایسه با *t*۰)
سپس با کمک رجیستر *t*۱ مقادیر ذخیره شده *a*۰ در *stack* به ترتیب صعودی فراخوانی و در مقدار حاصل‌ضرب قبلی، ضرب و ذخیره می‌شوند.

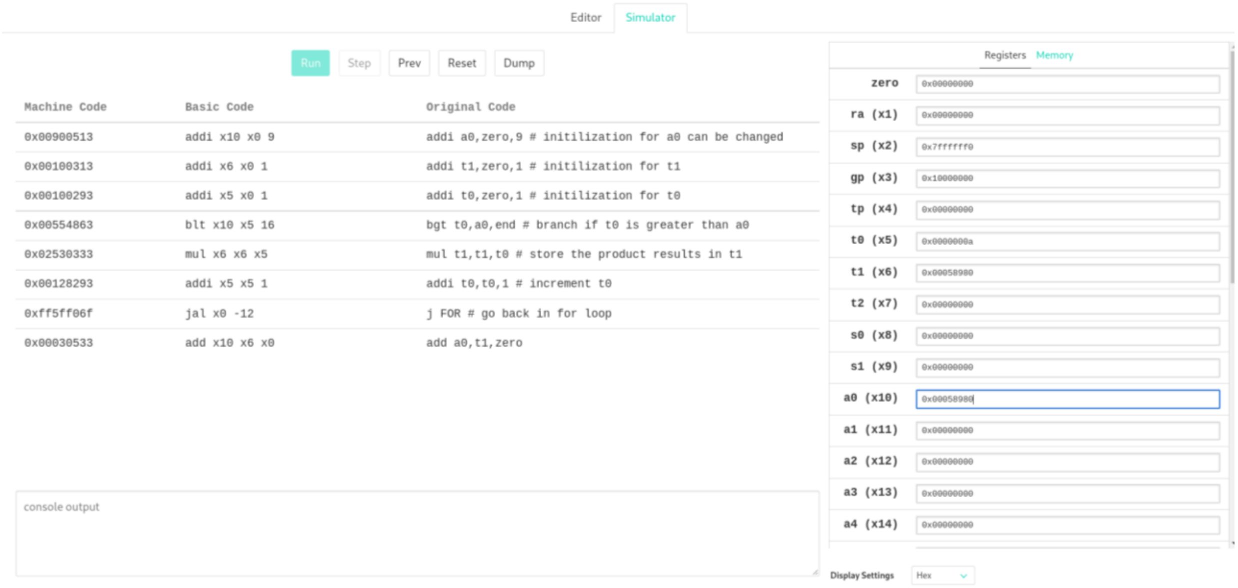
ج)

کد و نتیجه:



```
# a0 = n , t0 = i , t1 = result
addi a0,zero,9  # initilization for a0 can be changed
addi t1,zero,1  # initilization for t1
addi t0,zero,1  # initilization for t0

FOR:
bgt t0,a0,end   # branch if t0 is greater than a0
mul t1,t1,t0    # store the product results in t1
addi t0,t0,1    # increment t0
j FOR           # go back in for loop

end:
# put result into a0
add a0,t1,zero
```

۹ فاکتوریل:



Machine Code	Basic Code	Original Code
0x00900513	addi x10 x0 9	addi a0,zero,9 # initilization for a0 can be changed
0x00100313	addi x6 x0 1	addi t1,zero,1 # initilization for t1
0x00100293	addi x5 x0 1	addi t0,zero,1 # initilization for t0
0x00554863	blt x10 x5 16	bgt t0,a0,end # branch if t0 is greater than a0
0x02530333	mul x6 x6 x5	mul t1,t1,t0 # store the product results in t1
0x00128293	addi x5 x5 1	addi t0,t0,1 # increment t0
0xff5ff06f	jal x0 -12	j FOR # go back in for loop
0x00030533	add x10 x6 x0	add a0,t1,zero

zero	0x00000000
ra (x1)	0x00000000
sp (x2)	0x7ffffff0
gp (x3)	0x10000000
tp (x4)	0x00000000
t0 (x5)	0x0000000a
t1 (x6)	0x00058980
t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x00058980
a1 (x11)	0x00000000
a2 (x12)	0x00000000
a3 (x13)	0x00000000
a4 (x14)	0x00000000

Display Settings: Hex

سوال ۳:

توضیحات:

این برنامه به صورت بازگشتی تابع خیام پاسکال را محاسبه میکند. بدین منظور ابتدا برنامه را با C را نوشـتیم و سپس آن را با استفاده از دستورات موجود تبدیل به اسمبلی کردیم.

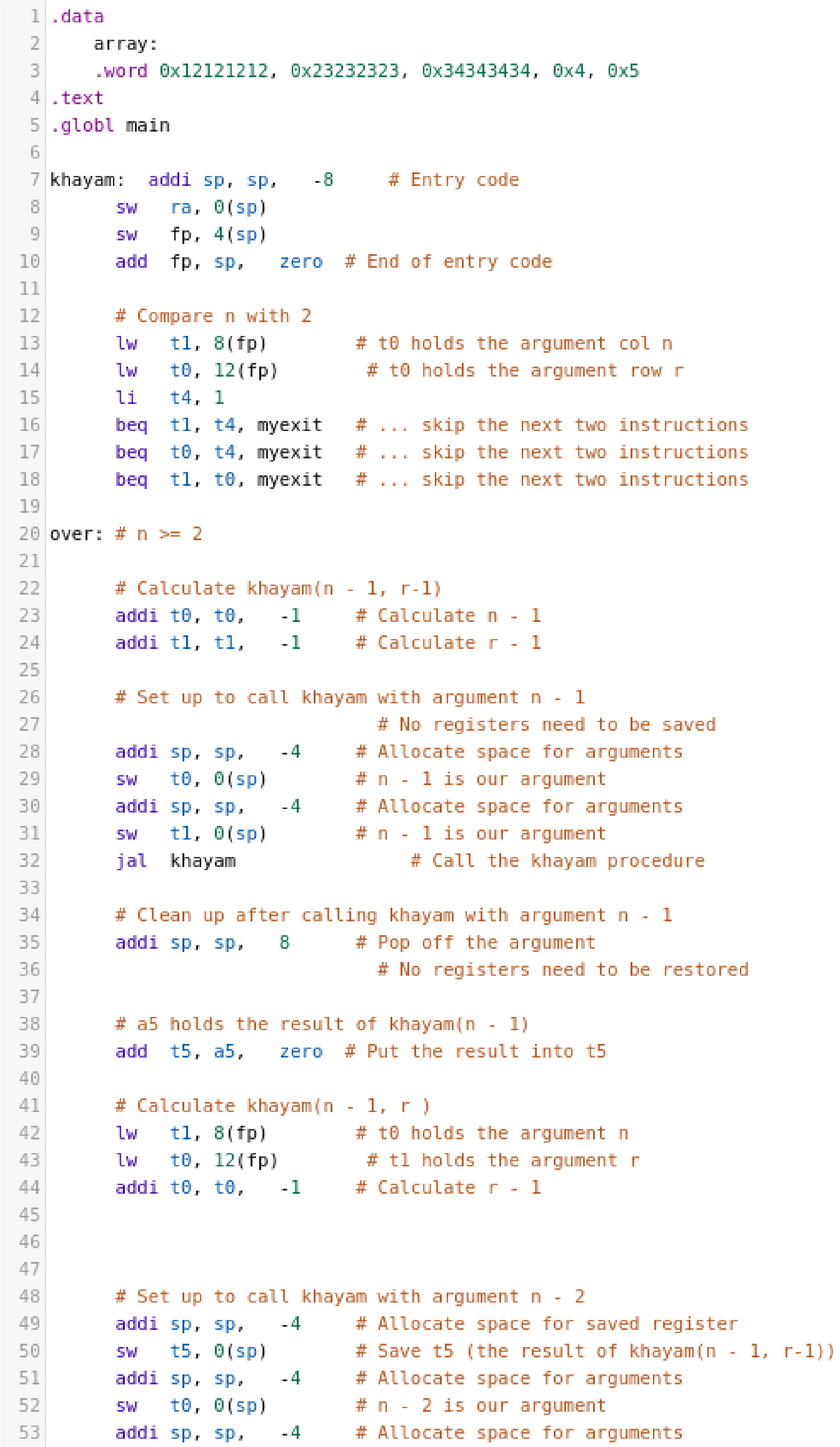
برنامه به صورت C به صورت زیر است.

```
calc(int n, int r)
 {
        if ((n == ۰) || (r == ۰) || (n == r))
                return ۱;
        else
                 return (calc(n-۱,r-۱) + calc(n-۱,r)); }
```

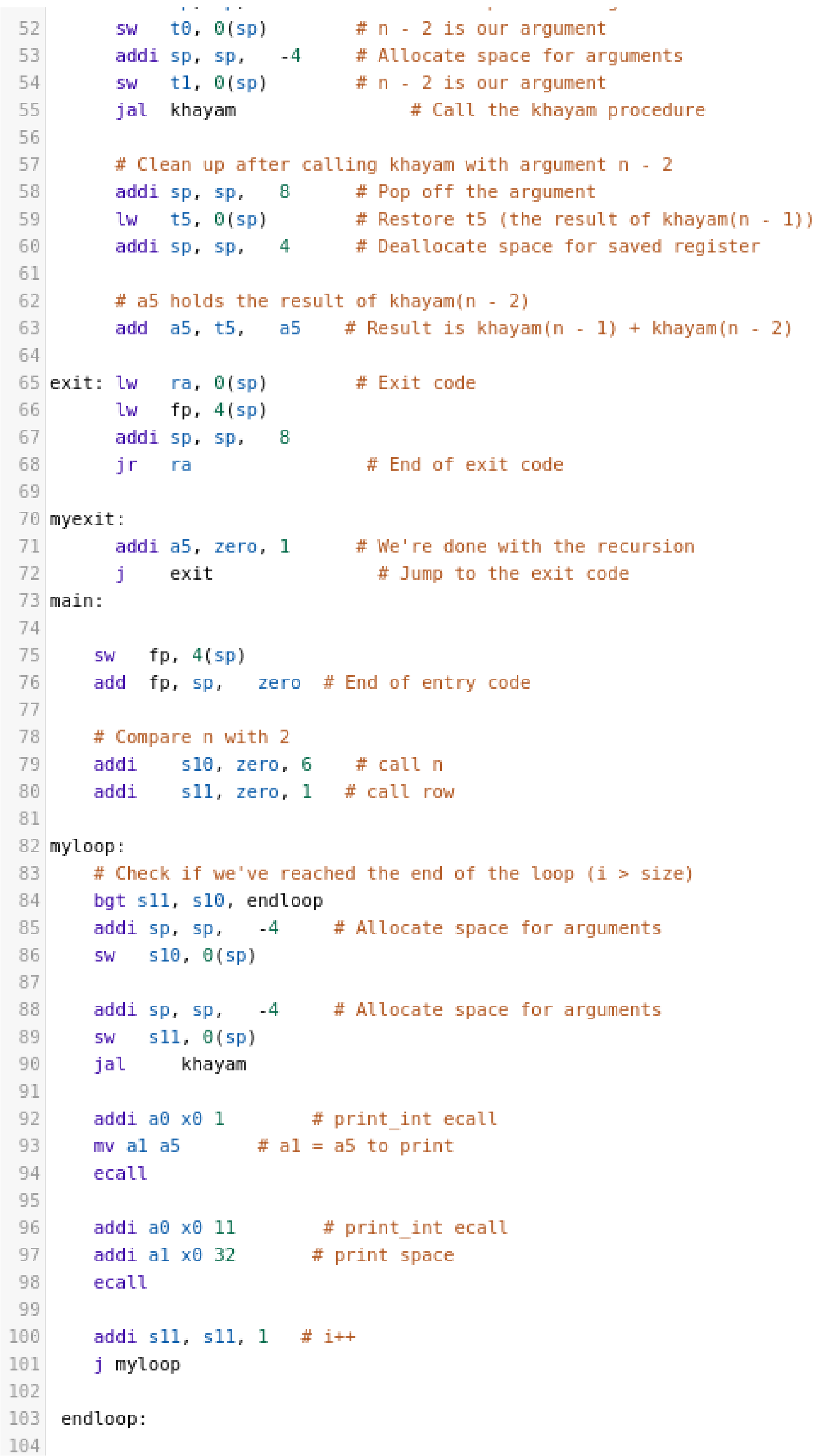
در برنامه اسمبلی به صورت خط به خط کامنت گذاری شده است.

ادامه در صفحه بعد ....

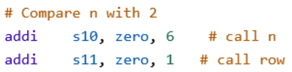
کد و نتیجه:



```
.data
    array:
    .word 0x12121212, 0x23232323, 0x34343434, 0x4, 0x5
.text
.globl main

khayam:  addi sp, sp,   -8      # Entry code
      sw   ra, 0(sp)
      sw   fp, 4(sp)
      add  fp, sp,   zero  # End of entry code

      # Compare n with 2
      lw   t1, 8(fp)        # t0 holds the argument col n
      lw   t0, 12(fp)        # t0 holds the argument row r
      li   t4, 1
      beq  t1, t4, myexit   # ... skip the next two instructions
      beq  t0, t4, myexit   # ... skip the next two instructions
      beq  t1, t0, myexit   # ... skip the next two instructions

over: # n >= 2

      # Calculate khayam(n - 1, r-1)
      addi t0, t0,   -1     # Calculate n - 1
      addi t1, t1,   -1     # Calculate r - 1

      # Set up to call khayam with argument n - 1
                              # No registers need to be saved
      addi sp, sp,   -4     # Allocate space for arguments
      sw   t0, 0(sp)        # n - 1 is our argument
      addi sp, sp,   -4     # Allocate space for arguments
      sw   t1, 0(sp)        # n - 1 is our argument
      jal  khayam                # Call the khayam procedure

      # Clean up after calling khayam with argument n - 1
      addi sp, sp,   8      # Pop off the argument
                              # No registers need to be restored

      # a5 holds the result of khayam(n - 1)
      add  t5, a5,   zero  # Put the result into t5

      # Calculate khayam(n - 1, r )
      lw   t1, 8(fp)        # t0 holds the argument n
      lw   t0, 12(fp)        # t1 holds the argument r
      addi t0, t0,   -1     # Calculate r - 1



      # Set up to call khayam with argument n - 2
      addi sp, sp,   -4     # Allocate space for saved register
      sw   t5, 0(sp)        # Save t5 (the result of khayam(n - 1, r-1))
      addi sp, sp,   -4     # Allocate space for arguments
      sw   t0, 0(sp)        # n - 2 is our argument
      addi sp, sp,   -4     # Allocate space for arguments
```

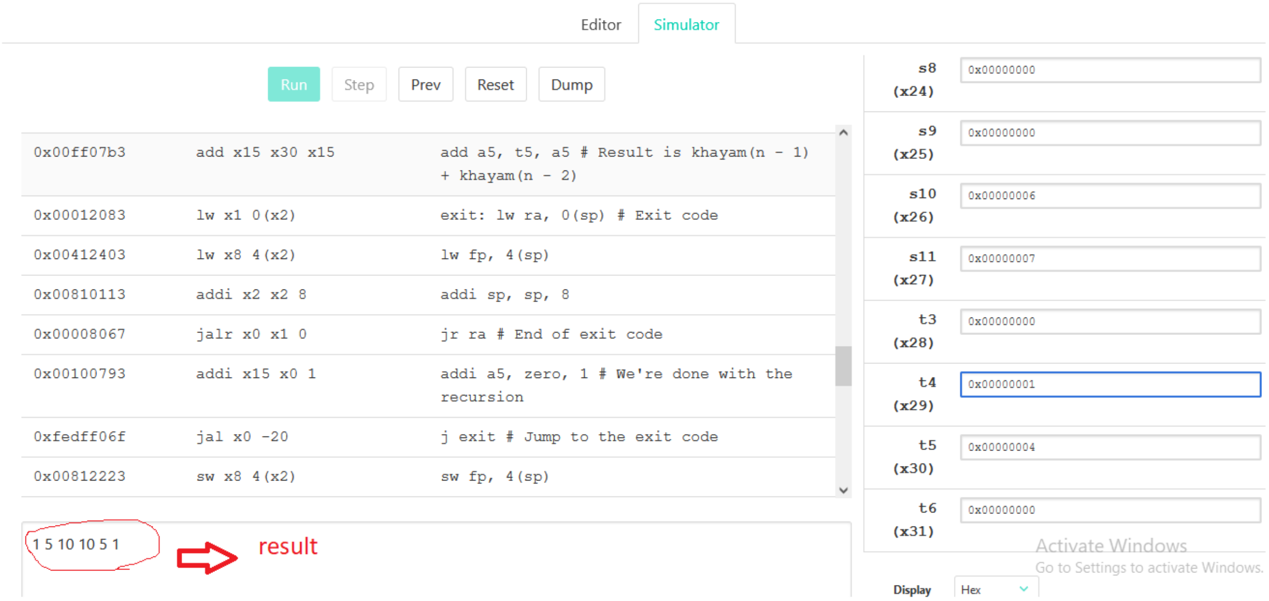


```
      sw   t0, 0(sp)        # n - 2 is our argument
      addi sp, sp,   -4     # Allocate space for arguments
      sw   t1, 0(sp)        # n - 2 is our argument
      jal  khayam                # Call the khayam procedure

      # Clean up after calling khayam with argument n - 2
      addi sp, sp,   8      # Pop off the argument
      lw   t5, 0(sp)        # Restore t5 (the result of khayam(n - 1))
      addi sp, sp,   4      # Deallocate space for saved register

      # a5 holds the result of khayam(n - 2)
      add  a5, t5,   a5    # Result is khayam(n - 1) + khayam(n - 2)

exit: lw   ra, 0(sp)        # Exit code
      lw   fp, 4(sp)
      addi sp, sp,   8
      jr   ra                # End of exit code

myexit:
      addi a5, zero, 1      # We're done with the recursion
      j    exit               # Jump to the exit code
main:

    sw   fp, 4(sp)
    add  fp, sp,   zero  # End of entry code

    # Compare n with 2
    addi    s10, zero, 6    # call n
    addi    s11, zero, 1   # call row

myloop:
    # Check if we've reached the end of the loop (i > size)
    bgt s11, s10, endloop
    addi sp, sp,   -4     # Allocate space for arguments
    sw   s10, 0(sp)

    addi sp, sp,   -4     # Allocate space for arguments
    sw   s11, 0(sp)
    jal     khayam

    addi a0 x0 1        # print_int ecall
    mv a1 a5       # a1 = a5 to print
    ecall

    addi a0 x0 11        # print_int ecall
    addi a1 x0 32       # print space
    ecall

    addi s11, s11, 1   # i++
    j myloop

 endloop:
```



```
# Compare n with 2
addi    s10, zero, 6    # call n
addi    s11, zero, 1   # call row
```

initializing n = 6



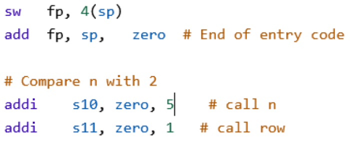
0x00ff07b3	add x15 x30 x15	add a5, t5, a5 # Result is khayam(n - 1) + khayam(n - 2)
0x00012083	lw x1 0(x2)	exit: lw ra, 0(sp) # Exit code
0x00412403	lw x8 4(x2)	lw fp, 4(sp)
0x00810113	addi x2 x2 8	addi sp, sp, 8
0x00008067	jalr x0 x1 0	jr ra # End of exit code
0x00100793	addi x15 x0 1	addi a5, zero, 1 # We're done with the recursion
0xfedff06f	jal x0 -20	j exit # Jump to the exit code
0x00812223	sw x8 4(x2)	sw fp, 4(sp)

s8 (x24)	0x00000000
s9 (x25)	0x00000000
s10 (x26)	0x00000006
s11 (x27)	0x00000007
t3 (x28)	0x00000000
t4 (x29)	0x00000001
t5 (x30)	0x00000004
t6 (x31)	0x00000000

Display Hex

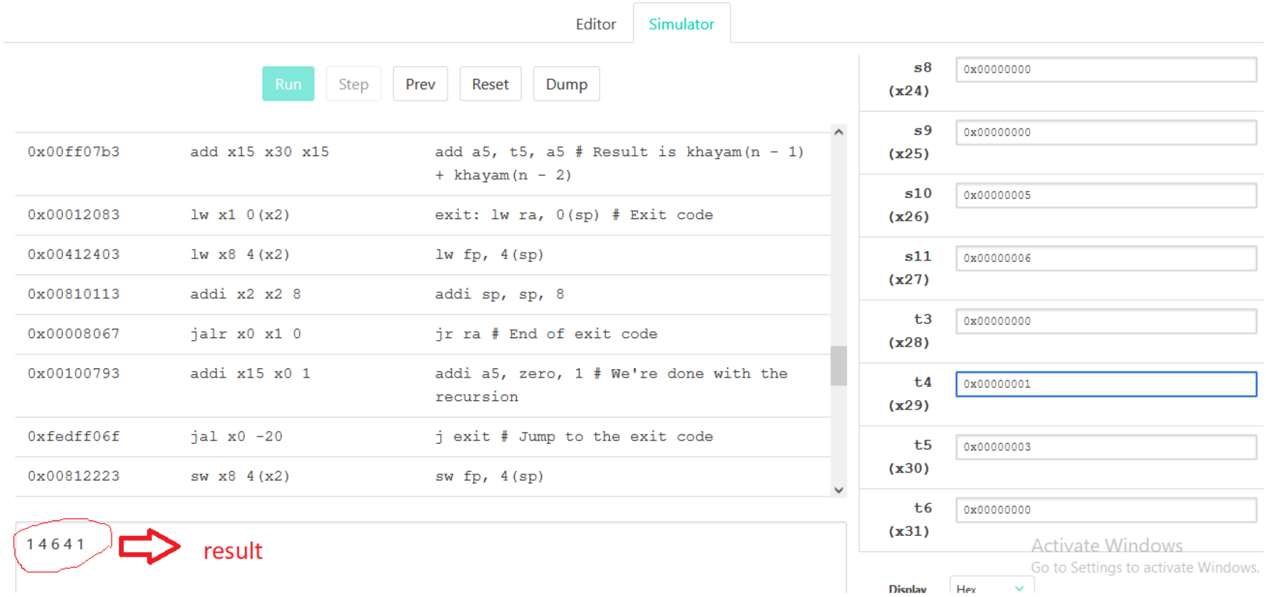
```
1 5 10 10 5 1
```

result



```
sw   fp, 4(sp)
add  fp, sp,   zero  # End of entry code

# Compare n with 2
addi    s10, zero, 5    # call n
addi    s11, zero, 1   # call row
```

initializing n = 5



0x00ff07b3	add x15 x30 x15	add a5, t5, a5 # Result is khayam(n - 1) + khayam(n - 2)
0x00012083	lw x1 0(x2)	exit: lw ra, 0(sp) # Exit code
0x00412403	lw x8 4(x2)	lw fp, 4(sp)
0x00810113	addi x2 x2 8	addi sp, sp, 8
0x00008067	jalr x0 x1 0	jr ra # End of exit code
0x00100793	addi x15 x0 1	addi a5, zero, 1 # We're done with the recursion
0xfedff06f	jal x0 -20	j exit # Jump to the exit code
0x00812223	sw x8 4(x2)	sw fp, 4(sp)

s8 (x24)	0x00000000
s9 (x25)	0x00000000
s10 (x26)	0x00000005
s11 (x27)	0x00000006
t3 (x28)	0x00000000
t4 (x29)	0x00000001
t5 (x30)	0x00000003
t6 (x31)	0x00000000

Display Hex

```
1 4 6 4 1
```

result