IST-ASM Final Exam — Fall 2024

Name:

- First, write your name in the box above. Then, have a quick read through all 7 questions.
- In the end, you will write up your answers on this paper.
  - But please make a draft elsewhere first. Only hand in something readable. Really.
- This is an open-book open-laptop exam: you may work on scrap paper and/or on your screen.
- Each question is independent from others, except stated otherwise.

Question 1 Perform the addition 100 + 50 in binary notation: convert both numbers to binary, then compute the sum entirely in binary. Show the details of your work.

decimal 100 as binary:

decimal 50 as binary:

addition:

```
  1100100
+  110010
 11
---------
 10010110 = 128 + 16 + 4 + 2 = 150
```

Question 2 Write number -37 in two's complement on eight bits. Show the details of your work.

```
 37 = 32 + 4 + 1
    = 0010 0101

~37 = 1101 1010

 +1 = 1101 1011
```

Question 3 Disassemble word `3120fff8` into ASM syntax.

`3` type=3 : conditional jump
`1` comparison code=3 : Branch if not equal
`2` rd=2 : destination register is R2
`0` rs=0 : source register is R0
`fff8` jump offset= -8

final answer: `bnez R2, -8`

Question 4 Write a program which uses a loop to compute the sum of all positive integers up to N. For instance with $N = 5$ you should find $1 + 2 + 3 + 4 + 5 = 15$. Initially N is stored in R1, and at the end the result should be in R2.

```
leti R1, 5
```

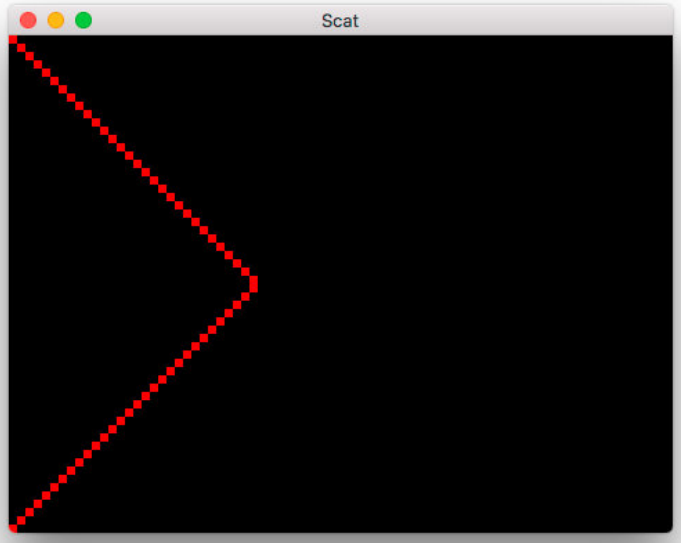
```
    leti R1, 5
    leti R2, 0
loop:
    add R2, R2, R1
    dec R1
    bnez r1, loop
    halt
```

Question 5 Write a program which loops over an array of integers and computes their average value, rounded down to the nearest integer (e.g. the "average" of 9, 15, 10 and 17 is 12). The length of the array is also given in memory, as illustrated below. At the end the result should be in R1.

```
bra main

A:    .word 9, 15, 10, 17
len:  .word 4

main:
```

```
bra main

A:      .word 9, 15, 10, 17
len:    .word 4

main:   leti R1, 0     ; partial sum
        leti R2, A     ; data ptr
        load R3, [len] ; remaining values counter

loop:   load R4, [R2]
        add  R1, R1, R4 ; accumulate
        addi R2, R2, 4  ; advance ptr
        dec R3
        bnez R3, loop

        load R3, [len]
        div R1, R1, R3
        halt
```

Question 6 In 30 lines or less, write a program that draws two diagonal red lines like illustrated on the right. The lines should start from the top left and bottom left corners of the screen, respectively, and they should end at mid-height i.e. where they cross each other.




```
        leti R1, 0xB0000000 ; VRAM address
```

```
        leti R2, 0xFF000000 ; color RED

        mov R3, R1        ; R1 is top-line ptr, R3 is bottom-line ptr
        leti R4, 320
        muli R5, r4, 59   ; advance R3 to bottom-left of screen
        add  R3, R3, R5

loop:
        store [R1], R2  ; draw one top-line pixel
        store [R3], R2  ; draw one bottom-line pixel

        add R1, R1, R4  ; move 1px down
        addi R1, R1, 4  ; move 1px right

        sub R3, R3, R4  ; move 1px up
        addi R3, r3, 4  ; move 1px right

        blt R1, R3, loop ; are we there yet ?

        halt
```

Question 7 Exponentiation is defined as repeated multiplication. In other words, expression a^n denotes "a multiplied by itself n times" i.e. $a \times a \times ... \times a$. Obviously, we could compute all these multiplications iteratively. But to optimize execution time, we can leverage the fact $a^k \times a^k = a^{2k}$, which leads to this recursive formulation (with base case $a^0 = 1$):

$$a^n = \begin{cases} (a^2)^{n/2}, & \text{if } n \text{ is even} \\ a\,(a^2)^{(n-1)/2}, & \text{if } n \text{ is odd} \end{cases}$$

In other words, exponentiation can be implemented by the following algorithm:

```
integer fast_exp(a: non-negative integer,
                 n: non-negative integer)
{
    if ( n == 0 )    return 1
    if ( n is even ) return fast_exp(a*a, n/2)
    if ( n is odd )  return a * fast_exp(a*a, (n-1)/2)
}
```

Your task is to translate this pseudo-code to assembly language. Please add comments to help us understand your answer. You can test your implementation with obvious examples like $10^7 = 10000000$ or $2^{20} = 1048576$.

```
    leti SP, 0x1000000
    leti R1, 10 ; base A
    leti R2, 7  ; exponent N
    call fast_exp
    ; expected result: R1 == A**N
    halt

fast_exp:
```

```
        leti SP, 0x1000000

        leti R1, 10 ; base X
        leti R2, 7  ; exponent N

        ;; leti R1, 2   ; base X
        ;; leti R2, 20  ; exponent N

        call fast_exp

        halt

fast_exp:
        push LR
        push R4

        beqz R2, ret_one

        modi R3, R2, 2   ; parity of N ?
        beqz R3, even    ; note: R3 is scratch
        bra odd
even:
        mul R1, R1, R1 ; X squared
        divi R2, R2, 2  ; N/2
        call fast_exp
        bra end
odd:
        mov  R4, R1 ; save X
        mul R1, R1, R1 ; X squared
        dec R2 ; N-1
        divi R2, R2, 2  ; (N-1)/2
        call fast_exp ; result comes back in R1
        mul R1, R1, R4
        bra end
ret_one:
        leti R1, 1 ; fall-through to epilogue for simplicity
end:
        pop R4
        pop LR
        ret
```