IST-ASM Retake Exam — Fall 2023

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 For each acronym below, give the full unabbreviated expression.

INSA	Institut National des Sciences Appliquées
CPU	
LR	
PC	
SP	

Central Processing Unit, Link Register, Program Counter, Stack Pointer

Question 2 Fill in the following table by converting each value to all notations.

Decimal	Binary	Hexadecimal
99		
		F1
		42
	1 1 1 1 1 0 1 0	

99	110 0011	63
241	1111 0001	F1
66	100 0010	42
250	1111 1010	FA

Question 3 In the boxes below, give the full machine language encoding for intruction `blt r6, r7, -8`.

31	28	24	20	16	12	8	4	0

32 67 FF F8 == 0011 0010 0110 0111 1111 1111 1111 1000

Question 4 Write a program which sets register 4 to the value `0xCAFE`. You cannot use the `leti` instruction.

leti is forbidden because it would be too confusing: `leti R5, 0xCAFE` would leave the immediate untouched, which triggers a sext at runtime. this is probably not what you want.
(Our assembler assumes that 0xFFFF really is a -1, the programmer has to write `leti Rn, 0x0FFFF` if they really want a positive number. yes this is confusing)

```
$ python
>>> hex(0xcafe >> 1)
'0x657f'
```

```
addi R4, R0, 0x657F
add  R4, R4, R4
```

Question 5 The Hamming weight of an integer is defined as the number of bits equal to one in its binary representation. For instance, the Hamming weight of 42 = 0b101010 is three, and the Hamming weight of 0xFFFFFFFF is 32. Write a program which computes the Hamming weight of any number (initially stored in R1) then halts. In a comment, indicate which register holds the result.

;

```
        leti R1, 0x00FF00AA ; weight 4+4+2+2 = 12
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        leti R2, 0 ; initial weight
loop:
        beqz R1, done ; are we finished yet ?

        andi R3, R1, 1
        beqz R3, shift ; skip the increment
        addi R2, R2, 1 ; increment weight
shift:
        lsri r1, r1, 1
        jmp loop

done:
        halt ;;comment: result is in R2
```

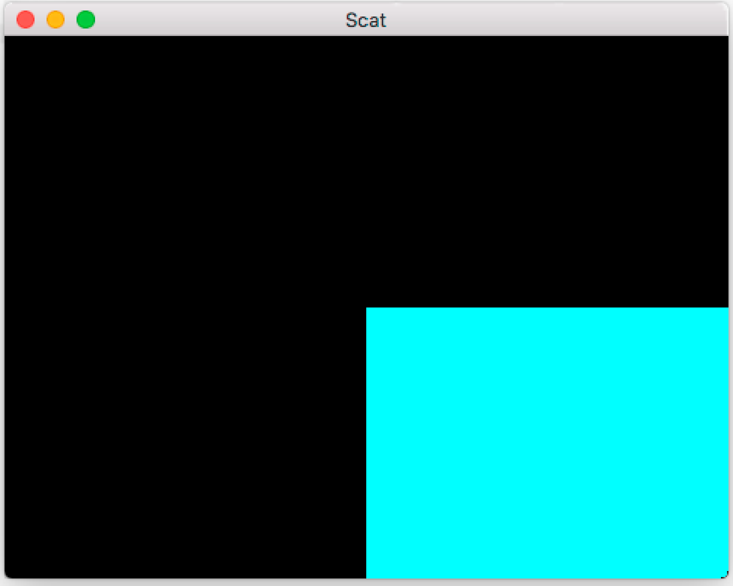
Question 6 Given two arrays A and B of the same (known) length, we define their *element-wise distance* as the array C such that for all *n*, $\mathrm{C}[n]=\left|\mathrm{A}[n]-\mathrm{B}[n]\right|$. In other words, each element of C is defined as the absolute value of the difference between corresponding elements of A and B.

The program below allocates two arrays A and B of length 10. Complete the code so that it computes their element-wise distance in array C.

```
start:
    jmp main

A:      .word 13, 50,  2, 42, 27, 12,  1, 8, 37, 19
B:      .word  1,  5, 24, 42, 51, 21, 36, 2, 71,  7
C:      .word  0,  0,  0,  0,  0,  0,  0, 0,  0,  0

main:
```

```
start:
    jmp main

T1:     .word 13, 50,  2, 43, 27, 12,  1, 8, 37, 19
T2:     .word 1,   5, 24,  4, 72, 21, 36, 2, 71,  7
T3:     .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    ;; should be
    ;; 12, 45, 22, 39, 45, 9, 35, 6, 34, 12
    ;;  c, 2d, 16, 27, 2d, 9, 23, 6, 22, c
main:
    leti r1, T1                         ; pointer to T1
    leti r2, T2                         ; pointer to T2
    leti r3, T3                         ; pointer to T3

    leti r9, 0                          ; i
    leti r10, 10

loop:
    load r4, [r1]                       ; r4 contains T1[i]
    load r5, [r2]                       ; r5 contains T2[i]
    ble r4, r5, first                   ; if T1[i] <= T2[i] ... goto first ...
    sub r6, r4, r5                      ; otherwise compute T1[i]-T2[i]
    jmp store_result
first:
    sub r6, r5, r4                      ; ... and compute T2[i]-T1[i]

store_result:
    store [r3], r6

advance:
    addi r1, r1, 4
    addi r2, r2, 4
    addi r3, r3, 4
    addi r9, r9, 1
    blt r9, r10, loop

    halt
```

Question 7 In 25 lines or less, write a program which fills the bottom right quarter of the screen in cyan, like illustrated below.



```
        leti R9, 0xB0000000  ; framebuffer address
        leti R8, 0x00FFFF00  ; cyan color

        leti R7, 30          ; line number in pixels (we start halfway)
vloop:
        muli R6, R7, 320     ; vert offset in bytes
        leti R5, 160         ; horz offset in bytes (we start halfway)
hloop:
        add  R4, R5, R6      ; add vert and horz offsets
        add  R4, R4, R9      ; add framebuffer base address
        store [R4], R8       ; draw pixel

        addi R5, R5, 4       ; move right by one pixel (4 bytes)
        leti R1, 320
        blt  R5, R1, hloop   ; end-of-line test

        addi R7, R7, 1       ; move down by one pixel
        leti R1, 60
        blt  R7, R1, vloop   ; bottom-of-screen test

        halt
```