1 Bits and Bytes

1. Give a two’s complement binary representation (and corresponding hexadecimal representation) of the following decimal numbers:

   (a) 127
       0111 1111 (more leading zero bits acceptable), but not 111 1111
   (b) 241
       0 1111 0001 (more leading zero bits acceptable), but not 1111 0001
   (c) −30
       10 0010 (more leading one bits acceptable)

2. Give the decimal and binary representations of the following two’s complement numbers:

   (a) 0x81
       1000 0001, -127
   (b) 0x7F
       0111 1111, 127

3. Give the (unsigned) decimal and binary representations of 0xCC.
   11001100, 204

4. What is the range (in decimal) of a 32-bit two’s complement number?
   $-2^{31} \ldots 2^{31} - 1$
2 MIPS Assembly Language Programming

1. Write a MIPS procedure called *StringMap* that interprets the value in $1 as the address of a string (an array of characters that ends with -1), and the value in $2 as the address of a procedure which interprets the value in $1 as a character and returns a character in $3 (we call it the character transformation function). *StringMap* should apply the character transformation function to character of the string and place the address of the transformed string in $3. You may assume the character transformation function will restore any registers it modifies. However, you may not assume the caller of *StringMap* stores/restores any registers.

    StringMap:
    ; Callee saves registers
    sw $31, -4($30)
    sw $9, -8($30)
    sw $8, -12($30)
    sw $5, -16($30)
    sw $4, -20($30)
    lis $31
    .word 20
    sub $30, $30, $31

    ; Constants
    lis $5
    .word -1 ; $5 = EOF
    list $4
    .word 4

    add $9, $1, $0 ; $9 = current address
    add $8, $1, $0 ; save return value

    foreach: lw $1, 0($9)
    beq $1, $5, end
    jalr $2
    sw $3, 0($9)
    add $9, $9, $4
    beq $0, $0, foreach

    end:
    add $3, $8, $0
    ; Restore registers
    lis $31
    .word 20
    add $30, $30, $31
    lw $31, -4($30)
    lw $9, -8($30)
    lw $8, -12($30)
    lw $5, -16($30)
    lw $4, -20($30)
    jr $31

2. Write a MIPS program that interprets the value in $1 as the address of an array of MIPS instructions
and executes the program stored in the array. You may assume each element contains exactly one
instruction, and the MIPS program stored in the array is valid (ending with “jr $31”) and takes no
arguments.

jr $1

3. **Exercise.** Write a MIPS procedure called *gcd* which interprets the values in $1 and $2 as unsigned
decimal integers and computes their greatest common divisor using recursion, placing the result in $3.

Hint: [https://en.wikipedia.org/wiki/Greatest_common_divisor#Using_Euclid.27s_algorithm](https://en.wikipedia.org/wiki/Greatest_common_divisor#Using_Euclid.27s_algorithm)

```
; Calculate gcd(a,b)
; $1 - a
; $2 - b
; $3 - gcd(a,b)
; $

; Recall that gcd(a,b) = gcd(b, a mod b)
; and gcd(a,0) = a

gcd:
; Save all registers used
sw $1 , -4 ($30)
sw $2 , -8 ($30)
sw $4 , -12 ($30)
sw $31, -16($30)
lis $31
.word 16
sub $30, $30, $31

; Base Case
; if b==0 return a
bne $0, $2, recur
add $3, $0, $1
beq $0, $0, clean

recur:
; Recurrence Case
; $1 contains b
; $2 contains a mod b
; $4 contains label gcd
divu $1, $2
add $1, $2, $0
mfhi $2
lis $4
.word gcd
jalr $4

clean:
; Load all register used
lis $31
.word 16
add $30, $30, $31
```
lw $31, -16($30)
lw $4 , -12($30)
lw $2 , -8 ($30)
lw $1 , -4 ($30)

jr $31

4. **Exercise.** According to the MIPS assembly language reference sheet, if you load a word from address 0xffff0004, it will place the next byte from standard input in the destination register. Write a program which takes a number in $1 and reads in that many elements from standard input into an array, stores the address of the array in $1 and the number of elements in $2, and then jumps to a symbol called "start". This is essentially mips.array except that the array elements are provided as characters rather than as integers.

; Constants
lis $29
.word 0xffff0004
lis $4
.word 4

; Move number of elements to $2
add $2, $1, $0

; Allocate space for array on the stack
mult $2, $4
mflo $3
sub $30, $30, $3
add $1, $30, $0

; $5 points to current address
add $5, $30, $0

ilooop: beq $3, $0, iendl
    ; Save an item from stdin to the array
    lw $6, 0($29)
    sw $6, 0($5)
    ; Increase array* and decrease counter
    add $5, $5, $4
    sub $3, $3, $4
    beq $0, $0, ilooop

; We’re done here: save $31 and call start
ielooop:
lis $5
.word start
sw $31, -4($30)
sub $30, $30, $4
jalr $5

; Restore $31 and exit upon returning
add $30, $30, $4
lw $31, -4($30)
jr $31

; Test code follows
; Prints out the contents of the array to stdout as chars
; Does not preserve registers (irresponsible!)

start:
    lis $4
    .word 4
    lis $11
    .word 1
    lis $28
    .word 0xffff000c

sloop: beq $2, $0, sendl
    ; Move from array to stdout
    lw $5, 0($1)
    sw $5, 0($28)
    add $1, $1, $4
    sub $2, $2, $11
    beq $0, $0, sloop

sendl: jr $31

5. Exercise. Given the following hexdump of an assembled MIPS assembly language program, determine the original program without using .word.

Your program must use labels where possible. What does the program do?

lis $3
    .word 1
    lis $4
    .word 1
label1: mult $3, $3
    mflo $3
    sub $1, $1, $4
    bne $1, $0, label1
    jr $31

Given a in $2 and b in $1, this program calculates $a^b$ and stores the result in $3.
3 MIPS Assembler

1. Give the symbol table for following MIPS assembly language program:

    mult: mult $1, $2
    beq $0, $0, branchLocation
    frog:
        gorf: .word gorf
    branchLocation: ; branch here
                    ; now return

        jr $31
    end: ; end of the program

Symbol table:
    mult 0
    frog 8
    gorf 8
    branchLocation 12
    end 16

2. What is the purpose of the first and second pass in an assembler? Why does assemblers require two passes?
   First pass: create symbol table (with some error-checking). Second pass: syntax & error checking + machine code output
   Because labels can be used before they are defined

3. Ed tries to write his CS350 assignment in MIPS that requires to loop through 90,000 lines of code. Describe a potential problem that Ed could encounter if he uses branching instructions. How could Ed solve this problem?
   Branch instructions use 16-bit two’s complement integers, so they can only encode branches from $-2^{15}$ to $2^{15} - 1$ (-32768 to 32767). To encode a jump of 90 000 instructions, we can store the value of the target address in a register and jump to that address.

    lis $5
    .word jumptohere
    jr $5
    ...
    jumptohere: ...

4. What information can we not get back when we disassemble a program?
   Any comments and the names of labels will be lost when the program is assembled. Although we can retrieve the exact instructions used, it can be difficult to determine the context of those instructions.
4 Regular Languages

Note that for simplicity, you may chose to allow leading zeroes.

1. Prove that the language $L = \{A, B, C, D\}$ is regular.
   It suffices to provide the regular expression $(A|B|C|D)$ to show that $L$ is regular.

2. Give a DFA for each of the following languages:
   
   No solutions will be provided for this question
   
   (a) The language $L$ over $\Sigma = \Gamma \cup \{+\}$, where $\Gamma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, such that:
       
       $L = \{u + v : u, v \in \Gamma^*, u \equiv 0 \text{ (mod} 2) \text{ and } v \equiv 1 \text{ (mod} 2)\}$
   
   (b) Exercise. The language $L$ over $\Sigma = \{0, 1\}$ which contains strings with a binary value that is congruent to 3 modulo 8. (That is, the binary value of string $x$ divided by 8 gives a remainder of 3). Hint: Think about the 3 least-significant (rightmost) bits.

3. Give a regular expression for each of the following languages:
   
   (a) The language $L$ over $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ of positive decimal numbers divisible by 4.
       
       $(0|4|8)\left(\Sigma^* (0|2|4|6|8)(0|4|8)(1|3|5|7|9)(2|6))\right)$
   
   (b) Exercise. The language $L$ over $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$ of valid MIPS .word directives with positive decimal and all hexadecimal arguments. (Use _ to represent a space)
       
       "$\ast\text{.word}\ast\left(0|1|2|3|4|5|6|7|8|9|0x(0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f)^\ast\right)\ast$"

4. You are given a regular language $L$ and a DFA $M = \{\Sigma, Q, q_0, A, \delta\}$ for the language. In pseudocode, write a function that takes a string $s \in \Sigma^*$ as an argument and returns 1 if the string is in language $L$, 0 otherwise.
   
   assuming $s$ has length $n$
   
   state <- $q_0$
   for i from 1 to n
     state <- $\delta$(state, $s_i$)
   return (state \cap A) \neq \emptyset

5. Give an NFA for each of the following languages:
   
   No solutions will be provided for this question
   
   (a) The language $A$ over $\Sigma = \{0, 1\}$ of strings ends with either “100” or ”1”.
   
   (b) The language $B$ over $\Sigma = \{0, 1\}$ that is the complement of $A$. (For any language $A \subseteq \Sigma^*$, the complement of $A$ is $\Sigma^* \setminus A$)
   
   (c) The language over $\Sigma = \{c,a,r,t\}$ where words in the language must end with “car” or have ”cat” as a substring.
   
   (d) The language over $\Sigma = \{c,a,r,t\}$ where words must end with ”tar” and have “rat” as a substring.
   
   (e) Exercise. The language over $\Sigma = \{a,b,c,d\}$ where words must either include at least one a and no bs, or must include at least one b followed by at least one c.

6. For the 2 NFAs from the previous question, give an $\epsilon$-NFA for the language $B^* \cup A^*$. Describe the language accepted by this automaton.
7. Convert the following NFA into a DFA.

DFA:
5 Scanning

1. Using the above DFA, produce a tokenization of the following strings using the Simplified Maximal Munch algorithm. If the scanning produces an error, output all the tokens recognized up until the error and then the string ERROR.

(a) \text{0xc0x1-23--70xff}
\hspace{1cm} \text{HEXINT 0xc0, ID x1, INT -23, DECR --, INT 70, ID xff}

(b) \text{0xbeef--0xkcd}
\hspace{1cm} \text{HEXINT Oxbeef, DECR --, ERROR}