Computer Architecture and system programming laboratory - 2019/Spring

Search Go

- Main
- Syllabus
- Course info: Arch&SPlab, Splab, Arch only.
- Announcements
- Assignments
- Class material
- Practical sessions
- Lab sessions
- Lab Completions
- Labs schedule
- Previous exams
- Useful links
- Forum
- Labs Forum
- Submission system

- recent changes
- login

• Assignments » Assignment 1 printable version

• Assignmen 1: Binary to Decimal, C calling convention

Contents (hide)

Note the fixes in the assignment:

- 10010001101000101011001111000 is changed to 11110010001101000101011001111000 in the example to be 32-bit number
- fs is changed to format_string in assembly file provided

Submissions which deviate from the instructions will not be graded!

Please make your output exactly as in the examples below.
Otherwise automatic scripts would fail on your assignment and you will lose your points for this.

Assignment Description

This assignment contains two tasks:

Task 1 - Converting signed number in binary base to signed number in decimal base

We provide you a simple program in C that inputs a string (as a line) from the user in loop, and calls convertor(…) function **that you need to implemlent in assembly language**. The main loop is terminated when user enters "q" input string.

The input string may contain up to 32 digits, means, the input number is signed integer. Signed integer is positive if its MSB (bit 31) is 0; otherwise, signed integer is negative. If the input string contains less than 32 digits, it is positive (completed by '0' leading bits). For negative numbers a user should provide all 32 digits. For example, in order to provide "-5", the input string should be "11111111111111111111111111111011". The input "11" is indeed "00000000000000000000000000000011".

- convertor(…) receives a pointer to the beginning of a null terminated string
- assume that the input string contains only binary digit characters (i.e. '0' and '1'), or only 'q' in a case that a user wants to quit the program
- output string should also be null terminated
- do not copy '\n' character to the output string

- convertor(…) should print the result as a **string**, means, in your assembly code you should call printf function with '%s' format
- you should not print leading zeros. For example, print "57" and not "0057"

Examples:

You may use this online convertor to check correctness of your answers.

```
> task1Assignment1.out
10111011
187
10111111100
1532
11110010001101000101011001111000
-231451016
q
```

Character conversion will **not** be in place this time.
You should write the output string into another buffer (provided in the code skeletone) before printing.

Note: you **may not** use any available function that automatically does the conversion, such as printf. You need to compute the output digits yourself.
Note: for more efficient calculation you are strongly recommended to use a small table of precalculated powers of 2: 2^0, 2^1, …, 2^31. Then you just need to retrieve and summarize appropriate cells from this table to calculate the needed result.

Task 2 - C calling convention

You will implement entirely on your own (there is no provided code files for this task). Your program should be composed from two files, one written in C and one written in assembly language, according to the instructions below.

Write a C program that contains 'main(…)' function and performs the following steps:

1. reads two integer (32 bits) numbers x and y in decimal base from a user
2. calls 'void assFunc(int x, int y)' written in assembly language with the above integers as arguments

'void assFunc(int x, int y)' performs the following steps:

1. calls 'char c_checkValidity(int x, int y)' (as defined below) to check if the numbers are legal
2. if x and y are valid, calculates z=x+y, and prints z in decimal base
3. otherwise, prints "illegal input"

'char c_checkValidity(int x, int y)' performs the following steps:

1. returns false if x is negative
2. returns false if y is non-positive or greater than 2^15
3. returns true otherwise

Examples:

```
> task2Assignment1.out
5
1
6


> task2Assignment1.out
-5
1
illegal input
```

What We Provide

The attached files:

- makefile
- main_task1Assignment1.c
- asm_task1Assignment1.s

For task 1, you need to edit both assembly and C provided files.
Note that the provided C file does not contain the needed loop. Ensure you change the code appropriately.

For task 2, you are required to create 2 new files: main_task2Assignment1.c and asm_task2Assignment1.s. These files will contain the code for task 2 as described above. You are also required to modify the supplied Makefile so that main_task2Assignment1.c and asm_task2Assignment1.s will be compiled in a manner similar to task 1.

In order to compile the files for the first task , you only need to use the 'make' utility. In order to compile the files for the second task , you will have to modify the 'Makefile'. You can take example from the way task 1 is compiled.

Submission Instructions

You have to submit a single zip file, **ID1_ID2.zip** , includes 4 files: **asm_task1Assignment1.s**, **asm_task2Assignment1.s**, **main_task1Assignment1.c** and **main_task2Assignment1.c** . Do not add directory structure to the zip file! Make sure you follow the coding and submission instructions correctly (print exactly as requested).

Good Luck!