KTH Information and
Communication Technology

IS1200/IS1500

Lab1 – Assembly Programming
2017-09-04
v1.2

Introduction

Welcome to the first lab in the course! In this laboratory exercise you will learn the fundamentals of programming in an assembly language. After finishing this lab, you should be able to

1. analyze MIPS assembly code by using a simulator.
2. write short assembly code functions in MIPS assembler.
3. execute and test a compiled program on a PIC32 microcontroller.

Lab Environment

The first part of this lab is using MARS (MIPS Assembler and Runtime Simulator). You can download the software here http://courses.missouristate.edu/KenVollmar/MARS/.

In the second part of the lab, you will use a chipKIT development board, which includes a 32-bit MIPS microprocessor. The teaching assistants will hand out these development boards during the lab.

Preparations

You must do the lab work in advance, except for a few specific tasks. The teachers will examine your skills and knowledge at the lab session.

We recommend that you book a lab-session well in advance, and start preparing at least a week before the session.

Assignments 1 through 6 must be prepared completely before the start of your lab session.

Assignments 7 and 8 are performed during your lab session.

You can ask for help anytime before the lab session. There are several ways to get help; see the course website for details and alternatives.

During the lab session, the teachers work with examination, but can also offer help. Make sure to state clearly that you want to *ask questions*, rather than being examined.

Sometimes, our teachers may have to refer help-seekers to other sources of information, so that other students can be examined.

Resources

You will use the MARS simulator for Assignments 1 through 4. The simulator allows you ro run your programs without access to MIPS hardware.

The MCB32 Tools will be used for Assignments 5 through 7. These tools allow you to run your code on the Chipkit Uno32 board. Assignment 5 describes how to get started with the MCB32 Tools.

Get files `analyze.asm`, `hexmain.asm`, and `timetemplate.asm` from the course website. These files contain code to get you started with your first Assignments.

Also get the compressed archive `time4mips.zip` from the course website.

Reading Guidelines

Review the following course material while you are preparing your solutions.

- Lectures 1, 2, and 3.
- The course web page named *Reading Guidelines* that lists specific chapters in the text books. See information about module 1.
- The *MIPS reference sheet* that is available from the *Literature* page on the course web site.

General description of the files in the compressed archive `time4mips.zip`

- COPYING: Copyright information. Add lines with your name/names when you edit or add one or more files.
- Makefile: Standard file telling the system how to compile MIPS programs. Don't change this file.
- mipslab.h: Contains declarations of functions and variables. Included by C files. If you want to change this file, you are probably doing something wrong.
- mipslabdata.c: Contains declarations of some arrays and matrices used for the display. Don't change this file.
- mipslabfunc.c: Contains functions used for the labs. Don't change this file.
- mipslabmain.c: Contains start-up code, that initializes the display and shows the welcome message. Following that, this file calls your lab-specific functions. Don't change this file.
- mipslabwork.c: This file gets your own code started, in the first lab. You may need to change this file as part of a surprise assignment, and also for later labs and projects. Add lines with your name and the date whenever you change this file.
- stubs.c: Contains interrupt-related C code. For later labs and projects, you may need to change this file.
- vectors.S: Contains interrupt-related assembly code. For later labs and projects, you may need to change this file.

Examination

Examination is grouped into parts. Each part is usually a group of several assignments. Examining one part takes 5–15 minutes. Make sure to call the teacher immediately when you are done with an assignment.

Please state clearly that you want to *be examined*, rather than getting help.

The teacher checks that your program behaves correctly, and that your code follows all coding requirements and is easily readable. In particular, all assembly-language code **must** conform to the calling conventions for MIPS-32 systems. *The teacher will also ask you to compile your program. If there are any errors or warnings, the teacher will ask you to solve these problems before the actual examination can begin.*

The teacher will also ask questions, to check that your knowledge of the design and implementation of your program is deep, detailed, and complete. When you write code, make detailed notes on your choices of algorithms, data-structures, and implementation details. With these notes, you can quickly refresh your memory during examination, if some particular detail has slipped your mind.

You must be able to answer all questions. In the unlikely case that some of your answers are incorrect or incomplete, we follow a specific procedure for re-examination. This procedure is posted on the course website.

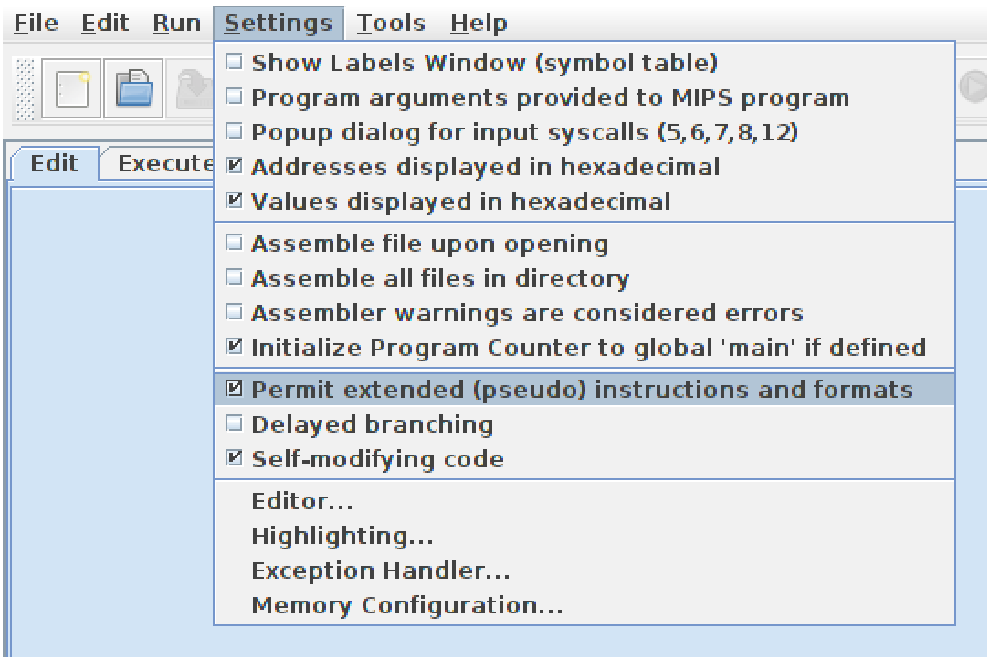
Part 1 – Assignments 1 and 2.

Part 2 – Assignments 3 through 7.

Part 3 – Assignment 8, the surprise assignment.

Assignments

When you run these assignments int the MARS simulator, make sure that **pseudo-instructions are turned on, and that delayed branching is turned off.** There are check-boxes for these items in the settings – see the screenshot:

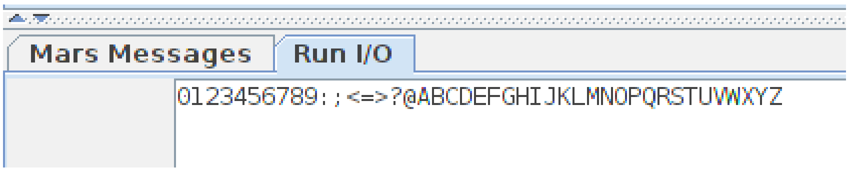


The recommended settings (on or off) for all options are shown in the screenshot.

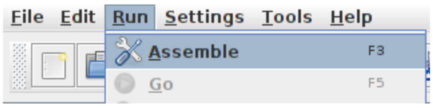
Assignment 1: Analyzing assembly code

Get file `analyze.asm` from the course website. Start the MARS simulator. Load file `analyze.asm`.

This file contains a small program, which prints the ASCII characters from 0 through Z in the "Run I/O" console window in Mars. After "Z", the program stops. See the screenshot:

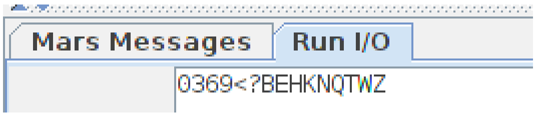


Assemble the code by clicking on the “screwdriver and wrench” icon:



Single-step through the program, then run it at full speed. While single-stepping, please note how the contents of registers `$s0` and `$a0` change during execution.

Now, change the program so that only every third character is printed. The program must still stop after "Z". The screenshot below shows the expected output:



Questions for assignment 1

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- Which lines of code had to be changed? Why?

Assignment 2: Writing your first assembly-language function

Get file `hexmain.asm` from the course website. Start the MARS simulator. Load file `hexmain.asm`.

You will now write a small subroutine, that converts numbers in the range 0 through 15 into a printable ASCII-coded character: '0' through '9', or 'A' through 'F', depending on the number.

For numbers not in the range 0 through 15, some bits will be ignored.

In file `hexmain.asm`, add an assembly-language subroutine with the following specification.

Name: The subroutine must be called `hexasc`.
Parameter: One, in register `$a0`. The 4 least significant bits specify a number, from 0 through 15. All other bits in register `$a0` can have any value and must be ignored.
Return value: The 7 least significant bits in register `$v0` must be an ASCII code as described below. All other bits must be zero when your function returns.
Required action: The function must convert input values 0 through 9 into the ASCII codes for digits '0' through '9', respectively. Input values 10 through 15 must be converted to the ASCII codes for letters 'A' through 'F', respectively. An ASCII code table is available at the last page in the document.

Important note for all subroutines: When a subroutine returns, registers \$s0–\$s7, \$gp, \$sp, \$fp, and \$ra *must* have the same contents as when the subroutine was called. If a subroutine uses any of these registers, the original contents must be saved, and restored before the subroutine returns. As a special case, the contents of registers \$k0 and \$k1 may not be modified *at all;* these registers are reserved for interrupt-handling code. All other register contents may be modified by any subroutine.

Run the file and explain the output. Change the constant on the line marked "input here" and re-run the program. Run the program for all values from 0 through 15 and check that the digits 0 through 9 and the letters A through F appear correctly in the "Run I/O" console in MARS.

Questions for assignment 2

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

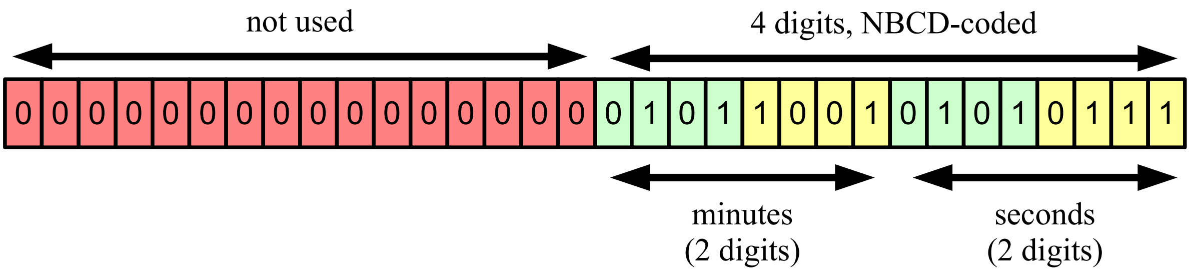
- Note to teachers and students: No s-registers may be used, and no registers should be saved.
- Your subroutine hexasc is called with an integer-value as an argument in register \$a0, and returns a return-value in register \$v0. If the argument is 17, what is the return-value? Why?
- If your solution contains a conditional-branch instruction: which input values cause the instruction to actually branch to another location? This is called a *taken* branch.

Assignment 3: Printing the time

a) Get file `timetemplate.asm` from the course website. The file `timetemplate.asm` contains most of the parts of a clock-program.

The variable `mytime` is initialized to 59:57, to be read as 59 minutes and 57 seconds. The subroutine `tick` increments the contents of the `mytime` variable, once for each iteration of the loop (from `main:` to `b main`).

The variable `mytime` stores time-info as four separate digits. Each digit uses four bits, allowing values from 0 through 9. The values 10 through 15 are also technically possible, but are not used.



You will complete the clock-program, by writing the subroutines `delay` and `time2string`; both subroutines are described below.

b) Copy your function `hexasc` from the previous assignment. Paste the copied code at the end of file `timetemplate.asm`.

c) Write a small function, like this, at the end of file `timetemplate.asm`.

```
delay:
  jr $ra
  nop
```

This is a temporary version of the `delay` function. You will soon write a more useful version.

d) You will now write a function that converts time-info into a string of printable characters, with a null-byte as an end-of-string-marker. At the end of file `timetemplate.asm`, add an assembly-language subroutine with the following specification.

Name: The subroutine must be called `time2string`.
Parameters (two): Register `$a0` contains the address of an area in memory, suitably large for the output from `time2string`. The 16 least significant bits of register `$a1` contains time-info, organized as four NBCD-coded digits of 4 bits each. All other bits in register `$a1` can have any value and must be ignored. *Example:* register `$a0` can contain the address `0x100100017`, and register `$a1` can contain the value `0x00001653`.
Return value: None.
Required action: The following sequence of six characters must be written to the area in memory pointed to by register `$a0`.

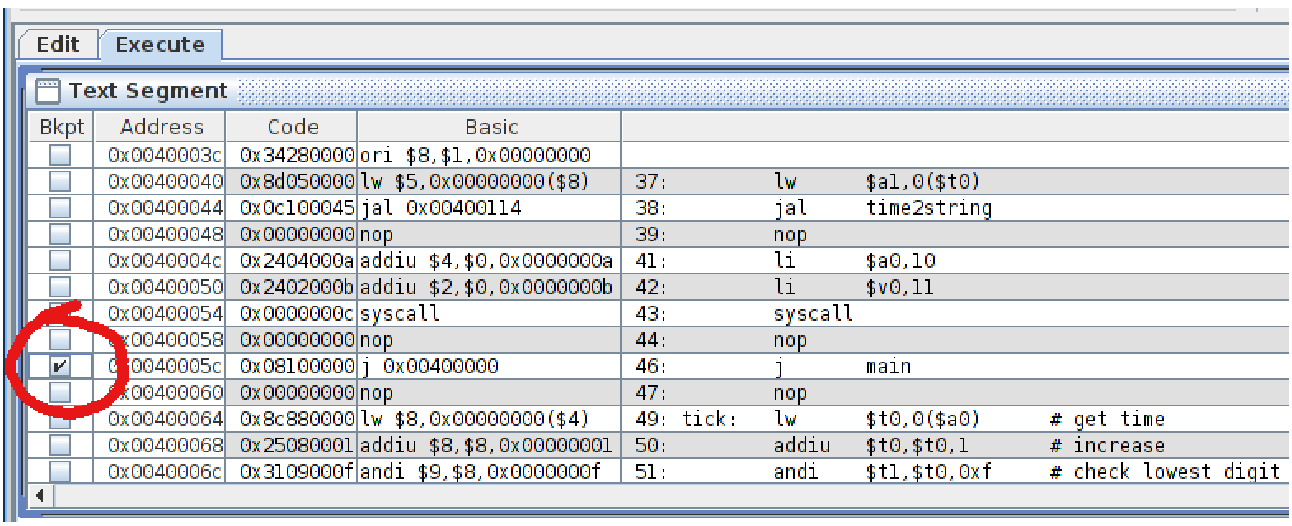
1. Two ASCII-coded digits showing the number of minutes, according to the two more significant NBCD-coded digits of the input parameter. *Example:* '1', '6' (ASCII 0x31, 0x36).
2. A colon character (ASCII **:**, code 0x3A).
3. Two ASCII-coded digits showing the number of seconds, according to the two less significant NBCD-coded digits of the input parameter. *Example:* '5', '3' (ASCII 0x35, 0x33).
4. A null byte (ASCII NUL, code 0x00).

*Notes:*You **must** use the function `hexasc` to convert each NBCD-coded digit into the corresponding ASCII code. Use the `sb` instruction to store each byte at the destination. The macros `PUSH` and `POP` are useful for saving and restoring registers.

Pitfall: Even when calling your own subroutine `hexasc`, you must save and restore registers just as if you didn't know anything about the internals of `hexasc`.

Important note for all subroutines (reprise): When a subroutine returns, registers $s0–$s7, $gp, $sp, $fp, and $ra *must* have the same contents as when the subroutine was called. If a subroutine uses any of these registers, the original contents must be saved, and restored before the subroutine returns. As a special case, the contents of registers $k0 and $k1 may not be modified *at all;* these registers are reserved for interrupt-handling code. All other register contents may be modified by any subroutine.

e) Test your program using the MARS simulator. When testing your function, insert a breakpoint at the instruction `j main`. The breakpoint will cause MARS to pause your program at the breakpoint, just before jumping back for the next iteration. To insert a breakpoint, check the box in the Bkpt column for the instruction `j main`. See the screenshow below.



Questions for assignment 3

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- Note to teachers and students: check register-usage and saving/restoring carefully.
- Which registers are saved and restored by your subroutine? Why?
- Which registers are used but not saved? Why are these not saved?
- Assume the time is 16:53. Which lines of your code handle the '5'?

Assignment 4: Programming a simple delay

You will now write a simple delay function. This kind of code can be used when a computer should wait a while between two different actions. However, using a program loop for delay is inefficient in many ways. In lab 3, we will use a timer-device to write a much-improved delay function.

a) Rewrite the following C function as an assembly-language subroutine with the same behavior.

```
void delay( int ms ) /* Wait a number of milliseconds, specified by the parameter value. */
{
  int i;
  while( ms > 0 )
  {
    ms = ms – 1;
    /* Executing the following for loop should take 1 ms */
    for( i = 0; i < 4711; i = i + 1 ) /* The constant 4711 must be easy to change! */
    {
      /* Do nothing. */
    }
  }
}
```

Important note for all subroutines (reprise): When a subroutine returns, registers \$s0–\$s7, \$gp, \$sp, \$fp, and \$ra *must* have the same contents as when the subroutine was called. If a subroutine uses any of these registers, the original contents must be saved, and restored before the subroutine returns. As a special case, the contents of registers \$k0 and \$k1 may not be modified *at all;* these registers are reserved for interrupt-handling code. All other register contents may be modified by any subroutine.

b) Replace the three-line `delay` subroutine from the previous Assignment, with your own assembly-language subroutine `delay` in file `timetemplate.asm`.

c) Test your program using MARS. Adjust the constant in the for loop to get a delay of 1000 ms when `delay` is called with a parameter value of 1000, and a delay of 3000 ms when `delay` is called with a parameter value of 3000.

Questions for assignment 4

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- Note to teachers and students: check that the assembly code matches the C code.
- If the argument value in register \$a0 is zero, which instructions in your subroutine are executed? How many times each? Why?
- Repeat the previous question for a negative number: -1.

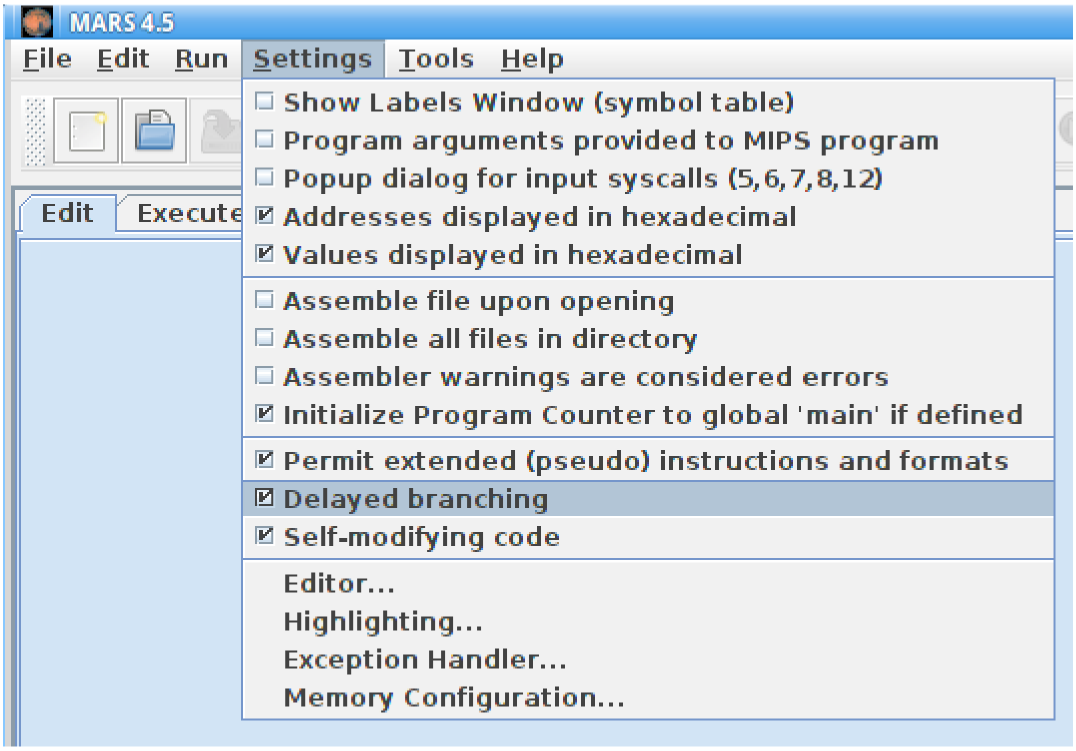
Assignment 5: Delayed branching

You will now prepare your program for actual Mips hardware, which always has delayed branching. Therefore, please change the MARS setting so that **delayed branching is turned on.**

The new recommended settings (on or off) for all options are shown in the screenshot.

Check that you have a `nop` instruction after every branch or jump. Run your program in Mars with the new settings, and verify that it still works.

Assignment 6: Move to the MCB32 environment



For this assignment, you need to have the MCB32 toolchain installed. You must also have an MCB32 Terminal open: "[mcb32]" should be shown as part of the promt-string in the Terminal window. Please see the course webpage “Software for Labs” for more information.

a) Get the zip file `time4mips.zip` from the course website. Unpack (unzip) the contents.

b) Copy your assembly-code for `hexasc`, `delay` and `time2string` from the previous assignment to the end of the file `labwork.S` (note the Capital S).

Please make sure that the directives `.global`, `.data`, and `.text` are placed correctly in the file. Since MARS doesn't use the `.global` directive, code that worked in MARS may need to be modified slightly when ported to the MCB32 environment.

c) In the Terminal window, at the "[mcb32]" prompt, change directory to the folder with the unzipped contents of file `time4mips.zip`. Then type

```
make
```

You will see a few lines of messages. Look for warnings and errors, and correct them all.

Note: The assembly-language dialects are slightly different for MARS and MCB32. Macros, in particular, must be rewritten (by you) when you move your code from one to the other. In particular, you should not use quotation marks in MCB32 macros.

Questions for assignment 6

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions which are not on this list.

- What is the effect of the assembler directive `.global`? Why is the directive particularly important in this assignment? The teachers will help you with this if necessary.

Assignment 7: At the lab session

For this assignment, you must have an MCB32 Terminal open: "[mcb32]" should be shown as part of the promt-string in the Terminal window. Please see the course webpage “Software for Labs” for more information. In the Terminal window, at the "[mcb32]" prompt, change directory to the folder with your code.

a) Connect the Uno32 + Basic IO Shield combo to your computer with an USB cable. Wait a few seconds, then type

```
make
```

followed by

```
make install
```

in order to compile and run your project on the board.

If this doesn't work right away, you need to figure out the name of the USB serial port.

- Linux: This is normally `/dev/ttyUSB0`
- Windows: In MSYS2, this is normally `/dev/ttyS2`
- Mac: This is normally `/dev/cu.usbserial-*` (replace * with something)

Ask the laboratory teachers for help if necessary.

When you know the path to your serial device, issue the command
`make install TTYDEV=/dev/ttyUSB0`, substituting the correct path of the device.

During the flashing process, the LEDs LD4 and LD5 on the Uno32 board should start blinking. When the process in done, your code should be running on your Uno32!

Mac users may have to press the Reset button on the Uno32 board, and then directly run "make install".

Note: The command `make` will only compile the code, and the command `make install` will only download into the board whatever is already compiled. Therefore, you need to use both commands in order to compile and run an update of your code.

b) Update the constant in the for loop of the delay subroutine, so that the time is updated correctly. Going from 00:00 to 01:59 should take two minutes. An error of less than +/- 10% is acceptable – that's 6 seconds per minute. Test and run your project on the board to adjust and demonstrate this.

c) Disconnect the USB connection from the board, count to three, and plug it back in. Check that the lab-board stores your latest program.

Questions for assignment 7

The following questions aim to help you check that you understand the assignment well. At the examination, the teacher may choose to ask questions which are not on this list.

- When you move your code from the simulator to the lab-board, you have to change the value of the constant in the `delay` subroutine to get correct timing. Why?

Assignment 8: Surprise assignment

You will get a surprise assignment at the lab session.

The surprise assignment must be completed during the session.

ASCII table – hexadecimal codes and corresponding ASCII characters

In this table, "0x" before a number indicates that the number is hexadecimal (in base 16).

0x00..NUL	0x10. .DLE	0x20.….SP	0x30........0	0x40......@	0x50.......P	0x60.......`	0x70........p
0x01..SOH	0x11. .DC1	0x21........!	0x31........1	0x41.......A	0x51.......Q	0x61........a	0x71........q
0x02. .STX	0x12. .DC2	0x22........"	0x32........2	0x42.......B	0x52.......R	0x62........b	0x72........r
0x03. .ETX	0x13. .DC3	0x23........#	0x33........3	0x43.......C	0x53.......S	0x63........c	0x73........s
0x04. .EOT	0x14. .DC4	0x24........$	0x34........4	0x44.......D	0x54.......T	0x64........d	0x74.........t
0x05..ENQ	0x15. NAK	0x25......%	0x35........5	0x45.......E	0x55.......U	0x65........e	0x75........u
0x06. ACK	0x16..SYN	0x26.......&	0x36........6	0x46.......F	0x56.......V	0x66........f	0x76........v
0x07. .BEL	0x17. .ETB	0x27.........'	0x37........7	0x47.......G	0x57......W	0x67........g	0x77.......w
0x08.....BS	0x18. CAN	0x28........(	0x38........8	0x48.......H	0x58.......X	0x68........h	0x78........x
0x09….HT	0x19....EM	0x29........)	0x39........9	0x49........I	0x59.......Y	0x69.........i	0x79........y
0x0A….LF	0x1a. .SUB	0x2a........*	0x3a.........:	0x4a........J	0x5a.......Z	0x6a.........j	0x7a........z
0x0B....VT	0x1b. .ESC	0x2b.......+	0x3b........;	0x4b.......K	0x5b........[	0x6b........k	0x7b........{
0x0C.....FF	0x1c.….FS	0x2c.........,	0x3c........<	0x4c.......L	0x5c.........\	0x6c.........l	0x7c.........|
0x0D. .CR	0x1d.....GS	0x2d........-	0x3d.......=	0x4d......M	0x5d........]	0x6d.......m	0x7d........}
0x0E….SO	0x1e.....RS	0x2e........ .	0x3e........>	0x4e.......N	0x5e........^	0x6e........n	0x7e........~
0x0F......SI	0x1f.….US	0x2F......../	0x3f.........?	0x4f.......O	0x5f........_	0x6f........o	0x7f...DEL

NUL = null character, used as filler.
BEL = bell, makes your computer beep when printed.
BS = backspace, moves the cursor left one step.
LF = line feed, moves the cursor down one line (often without moving to the leftmost column).
FF = form feed, starts a new page on a hard-copy printout (and sometimes also on screen).
CR = carriage return, moves the cursor to the beginning of the current line.
ESC = escape, starts a sequence of control characters.
DEL = delete, sometimes used to delete one character.
SP = space between words.

SOH, STX, ETX, EOT, ENQ, ACK, HT, VT, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, FS, GS, RS, US are control characters; we don't use them in this lab.

Version history

1.0 – First published version. 2015-09-04.

1.1 – refined and condensed questions, updated lab code. 2016-01-15.

1.2 – added a short note that all programs need to be error and warning free, before examination.