KTH Information and Communication Technology

IS1200/IS1500

Lab1 – Assembly Programming
2024-07-30
v1.3

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 RISC-V assembly code by using a simulator.
2. Write short assembly code functions in RISC-V assembler.
3. Execute and test a compiled program on a DTEK-V board

Lab Environment

The first part of this lab uses RARS (RISC-V Assembler and Runtime Simulator). Link to the project's repo: https://github.com/TheThirdOne/rars.

In the second part of the lab, you will use a DTEK-V development board, which includes a RISC-V/32 processor. These development boards will be loaned out during one of the early lectures.

To access the lab environment, either using your own computer or through KTH computers, please visit the "Software Resources" page on Canvas.

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 any time 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 will perform examinations, 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 RARS simulator for Assignments 1 through 4. The simulator allows you to run your programs without access to RISC-V hardware and allows the user to look up the register values, execute programs step-by-step and check the memory state.

The DTEK-V tools will be used for Assignments 5 through 7. These tools allow you to run your code on the DTEK-V board with riscv32. Assignment 5 describes how to get started with the DTEKV Tools.

All files required to perform the laboratory exercise can be downloaded from the Canvas course page under "Laboratory Exercises". There are four assembly files: `analyze.S`, `hex2asc.S`, `timetemplate.S and labmain.S.`

Reading Guidelines

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

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

General description of the files in the compressed archive `time4riscv.zip`

These files contain important functions that are needed to run on the DTEK-V board, which include how to boot up the board, how to manage interrupts, etc.

> **NOTE***: These files do not contain a main (entry) function and thus are insufficient to make a working program. Instead, a user must insert lab-specific files. Please complement these with lab-specific files to make a working program.*

- **`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 RISC-V programs. Don't change this file.
- **`dtekv-lib.c`**: Contains functions for printing characters. If you want to change this file, you are probably doing something wrong.
- **dtekv-lib.h:** The header containing the function prototypes in dtekv-lib.c
- **`boot.S`**: Contains start-up code and interrupt-handling code. After initialization, this code calls the `main` function. Don't change this file.
- **`dtekv-script.lds`**: Describes how to assemble the compiled object files into a binary executable ready to be executed on the DTEK-V board.
- **softfloat.a:** This is a library that emulates floating-point operations, and is based on the high-performance Rvfplib (https://github.com/pulp-platform/RVfplib/)

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, that your code follows all coding requirements, and that it is easily readable. All assembly-language code **must** conform to the calling conventions for RISC-V/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 the examination, if some 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.

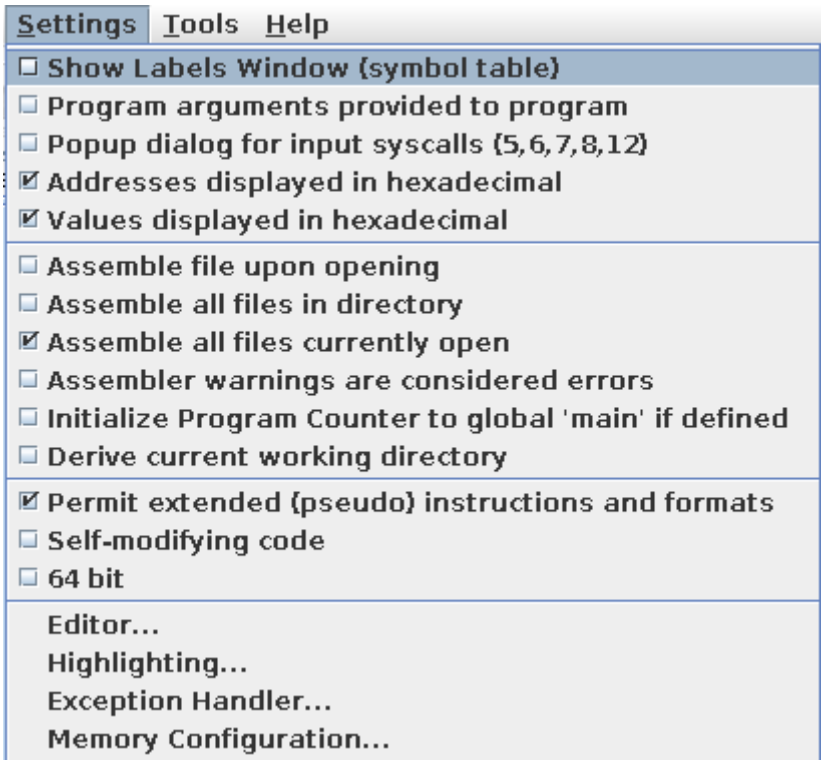
Part 1 – Assignments 1 and 2.

Part 2 – Assignments 3 through 7.

Part 3 – Assignment 8, the surprise assignment.

RARS setup

When you run the following Assignments in the RARS simulator, make sure that **pseudo-instructions are turned on** and that the option for **Assembling all files currently open is selected.** To do that, go into *Settings* and make sure that the following checkboxes are selected:



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

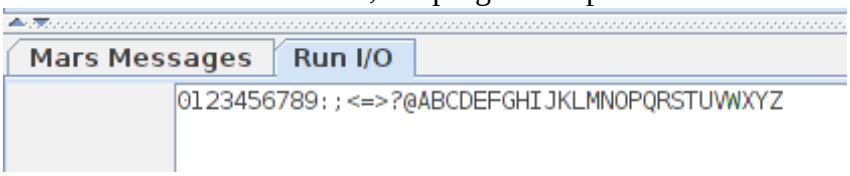
Assignments

NOTE: It is expected that the student will call different functions residing in files analyze.S, timetemplate.S and hex2asc.S from the main file labmain.S. To achieve this, the labmain.S file and one of the files containing the function you want to run (e.g., analyze from analyze.S) need to be assembled at the same time. RARS has an option for assembling the files currently opened during assembly (previous screenshot) that will allow us do that.
Example: if you want to run analyze, make sure that the line for calling analyze is uncommented from labmain.S and make sure that labmain.S and analyze.S are opened in RARS.

Assignment 1: Analyzing assembly code

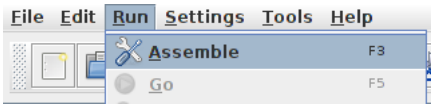
Start the RARS simulator. Load the files `analyze.S and labmain.S.`

File analyze.S contains a small function, which prints the ASCII characters from 0 through Z in the "Run I/O" console window in RARS. After "Z", the program stops. See the screenshot:



Expected output of the code in labmain.S

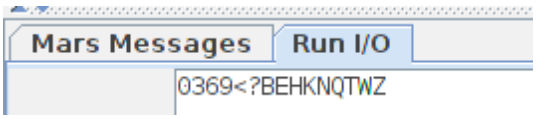
Assemble the code by clicking on the *Assemble* icon (or by clicking F3). **Make sure that you're assembling with** labmain.S **selected in Editor:**



Assembling the code.

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:



Expected output of the modified code in analyze.S

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 the file `hex2asc.S` from the course website. Start the RARS simulator. Load file `hex2asc.S and labmain.S.`

You will now write a small subroutine, that converts numbers 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 `hex2asc.S`, 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 `a0` 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 on the last page of this document.

Important note for all subroutines*:* When a subroutine returns, registers s1-s11, sp, s0 (or fp), ra, gp and tp **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. All other register contents may be modified by any subroutine.

Run the file and explain the output. Change the constant on the line marked "test number" 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 RARS.

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.

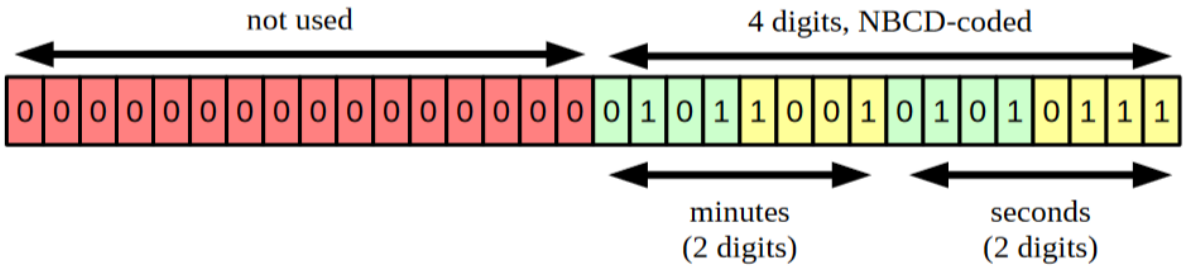
- Note to teachers and students: No s-registers may be used, and no registers should be saved.
- Your subroutine hex2asc is called with an integer-value as an argument in register a0 and returns a return-value in register a0. If the argument is 17, what is the return-value? Why?
- If your solution contains a conditional-branch instruction: which input values cause the instruction to branch to another location? This is called a *taken* branch.

Assignment 3: Printing the time

a) Get file `timetemplate.S` from the course website. The file `timetemplate.S` contains most of the parts of a clock function.

The variable `mytime` is initialized to 59:57, 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 `timetemplate:` to `j timetemplate`).

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.



Time representation in a word.

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

b) Copy your function `hex2asc` from the previous assignment. Paste the copied code at the end of the file `timetemplate.S`.

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

```
delay:
  jr ra
```

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.S`, add an assembly-language subroutine with the following specification:

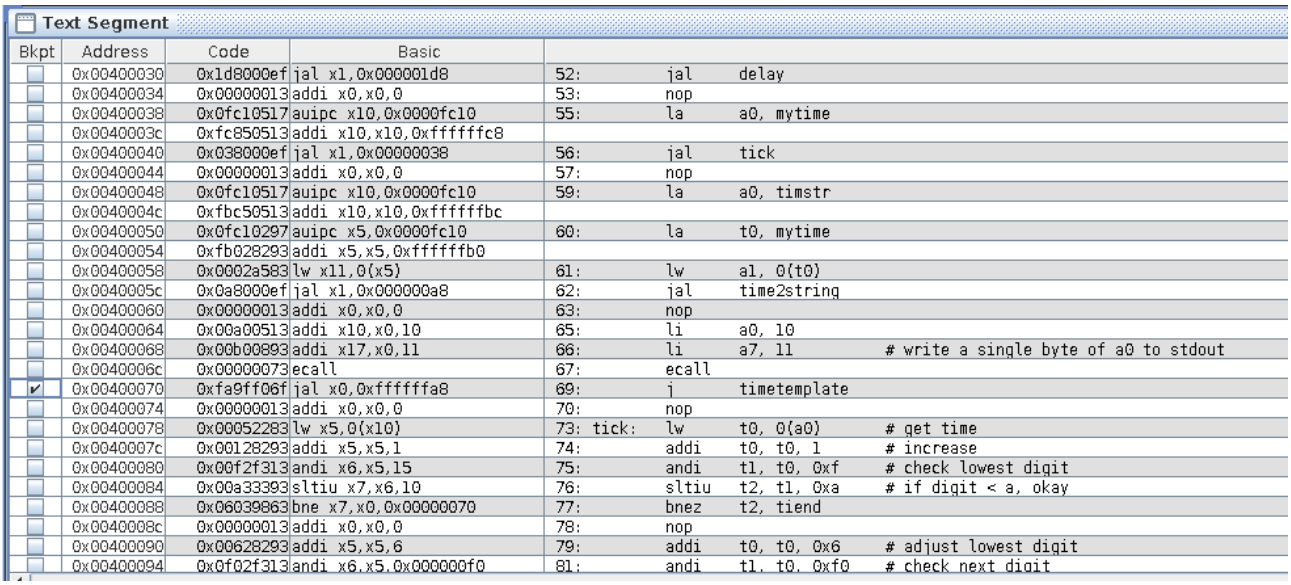
- *Name*: The subroutine must be called `time2string`.
- *Parameters (two):* Register `a0` contains the address of an area in memory, suitably large for the output from `time2string`. The 16 least significant bits of register `a1` contains time-info, organized as four NBCD-coded digits of 4 bits each. All other bits in register `a1` can have any value and must be ignored. *For 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`.
    - 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).
    - A colon character (ASCII `:`, code 0x3A).
    - 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).
    - A null byte (ASCII NUL, code 0x00).

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`.

NOTE: 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.

Important note for all subroutines (reminder): When a subroutine returns, registers s1-s11, sp, s0 (or fp), ra, gp and tp **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. All other register contents may be modified by any subroutine.

e) Test your program using the RARS simulator (do not forget to run it from `labmain.S` and uncomment the jump to `timetemplate`). When testing your function, insert a breakpoint at the instruction `j timetemplate`. The breakpoint will cause RARS 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 timetemplate`. See the screenshot below:



Setting up a breakpoint.

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 the 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 (reminder): When a subroutine returns, registers s1-s11, sp, s0 (or fp), ra, gp and tp **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. All other register contents may be modified by any subroutine.

Note: the constant 4711 is too large to fit into the `addi` instruction; use `li` or equivalent instead.

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

c) Test your program using RARS. 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 that 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: Move to the development board

For this assignment, you need to have the DTEK-V toolchain installed, which includes three binaries:

- **dtekv-run:** a file used to upload a binary executable file to the DTEK-V board and execute it there.
- **dtekv-upload:** a file used to upload data to the DTEK-V board. Unused in this lab.
- **dtekv-download:** a file that can download data from the DTEK-V board. Unused in this lab.

Please see the course webpage “Software Resources” for more information.

a) Get the zip file `time4riscv.zip` from the course website. Unpack (unzip) the contents.

b) Copy all your assembly-code for `hexasc`, `delay` and `time2string` from the previous assignment, as well as the `labmain.S` file into the directory where `time4riscv.zip` was unpacked.

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

c) In the Terminal window, 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 RARS and GCC. Macros must be rewritten (by you) when you move your code from one to the other. Uncomment the correct macros in the `timetemplate.S` file for use in RARS and DTEK-V respectively.

Questions for assignment 5

The following questions aim to help you check that you understand the code well. At the examination, the teacher may choose to ask questions that 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 6: At the lab session

For this assignment, you must have access to the DTEK-V toolchain and its binary `dtekv-run`. Please see the course webpage “Software Resources” for more information. In the Terminal window, at the prompt, change directory to the folder with your code.

a) Connect the DTEK-V board to your computer using the provided USB cable. Wait a few seconds and ensure that the board is active and lit up. After that, compile your program by typing:

```
make
```

followed by (assuming there were no errors in above compilation)

```
dtekv-run main.bin
```

to run your project on the board.

If this does not work immediately, you need to figure out why. Some possible candidates are:

- The jtagd daemon stopped working or timed-out. Type: “`killall jtagd && jtagd –user-start`” to restart it
- Your compilation failed with errors. Please look through your code and fix the errors and try again.
- You have an existing application running on the DTEK-V board. Press the reset button (see picture) and try again.

Ask the laboratory teachers for help if necessary and see the course Canvas page for further troubleshooting options.

During the time when the binary is being downloaded, information is being shown on your terminal and the LED called “Load” is blinking. When the process in done, your code should be running on your DTEK-V board.

Note: The command `make` will only compile the code, and the command `dtekv-run main.bin` 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.

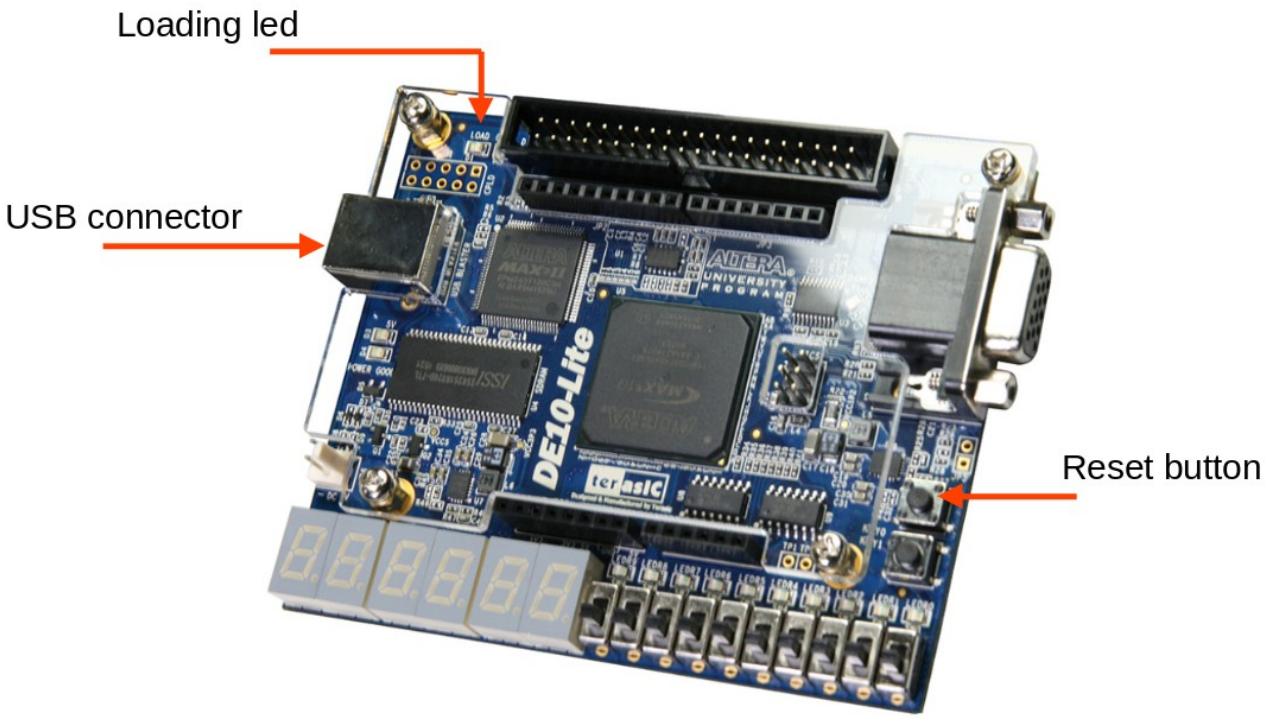
> ***NOTE:*** *The command `make` will only compile the code, and the command `dtekv-run main.bin` will only download into the board whatever is already compiled. Therefore, you need to use both commands in order to compile and run an update of your code.*
>
> ***NOTE:*** *Remember to Reset the device when you want to program it by pressing the reset button (KEY0).*

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.

Questions for Assignment 6

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

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



Assignment 7: 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.

1.3 – Ported to RISC-V/32 platform. 2024-07