RA FS 23 Series 5

Abdelhak Lemkhenter, Sameni Sepehr, Viktor Shipitsin

The fifth series has to be solved until Tuesday, 2. May 2023 at 3 p.m. and has to be uploaded to ILIAS. If questions arise, you can use the ILIAS forum at any time. Any problems should be communicated as soon as possible, we will be happy to help.
Have fun!

Theoretical Questions

Total score: 13 points

1 Static Branch Structure (3 points)

Which problems arise in the following assembly loop, if *Predict Not Taken* is chosen by default?

N.B. : Refer to slide 26 in 005e-Der-Prozessor - Pipeline Konflikte.pdf for the difference between jump and branch instructions in terms of required stalls.

```
1       addi $s1, $zero, 1024 // s1 := 1024
2 loop: addi $s1, $s1, -1 // s1--
3       jal subroutine // call subroutine()
4       bne $s1, $zero, loop // if (s1 != 0) jump loop
```

Optimize the example to *Predict not Taken*.

2 Antidependencies (2 point)

Explain when antidepencies happend and provide a short code example.

3 Control Hazards (3 points)

What are the three cases when control hazards occur? Given an example for each case.

4 Stalls (1 points)

What the values of the control bits when the a *noop* instruction is used.

5 Branch Predictor (3 points)

State how many times does a 2-bit branch predict need to be wrong before it changes its prediction. List all possible states of such a predictor and the transitions between them.

6 Storage Conflict (1 point)

Explain what register renaming is.

Programming Part

The programming exercises with the Raspberry Pi must be solved in groups of two or three students. You and your group members work together such that everyone contributes equally. In order to ensure that all participants understand the whole code they turn in, we require each student to hand in a slightly different version of the exercise/code. As described below, these versions differ only in a very specific functionality for which each individual student is responsible.

Paddle Ball

In this programming assignment, we ask you to implement a digital version of the game *Paddle Ball*[1]. There are three states the game can be in:

Start The player starts the game with `BUTTON2`. This will reset the game score to zero.

Playing The active LED ("the ball") starts to move in one direction. Change of direction is at `LED0` and `LED7` respectively, where the player has to press `BUTTON1` in the right moment to send the ball in the opposite direction. Two cases apply here:

Hit If the player presses `BUTTON1` within the allowed time window, he is awarded one point.

Miss If the player misses the time window and does not interact with the button, the buzzer makes an acoustic signal to indicate a miss. In this case the score does *not* change and the game continues.

In any case, the speed of the running light increases every round (next "level").

Game Over The game ends as soon as the maximum speed is reached (highest level). The LED panel shows the score in binary format.

Details

The player starts the game with `BUTTON2`. This will initiate a running light animation, that is, the active LED moves from left to right, then from right to left and so on. You are free to choose the initial position and direction of the light. We recommend that you reuse the code from Serie 4 for this part.

At the time where `LEDR0` or `LEDR7` is active, the player has to interact with `BUTTON1`. The score does not change if the button is pressed to early or to late (or not at all). Instead, the buzzer outputs a warning signal (a short beep). In the appendix you find instructions regarding the usage of the buzzer.

With every change of direction, the speed of the running light is increased. It is up to you to choose how much faster the light gets and how difficult the game should be overall. You should also define the maximum speed at which the game terminates. At this point, the game stops and the LED panel shows the score in binary format. With `BUTTON2`, the player is able to restart the game.

This is the basic functionality you need to implement. We encourage you to add your own ideas to the game and its rules. If you do so, please clearly describe all changes as a comment at the top of the source code or in a separate text file.

Exercises

(a) Extend the running light from Serie 4 to "Paddle Ball" as described above.

Student 1: You hand in the standard version as described.

Student 2: You modify the standard version in such a way that the penalty signal for a miss is inverted. This means that the buzzer is activated when the player hits the ball, and no sound is played when the ball is missed.

Student 3: In your version, the score display at the end of the game must be inverted, that is, the LEDs that are turned off show the binary value of the score.

(b) Make sure that your source code contains useful and detailed comments. This is an essential requirement to pass the programming exercise!

[1] https://en.wikipedia.org/wiki/Paddle_ball

(c) Make sure that your program compiles without any errors or warnings. This is also an essential requirement to pass the programming exercise!

(d) Create a Zip file named `<firstname>-<lastname>.zip` with *all the files*, where `<firstname>` and `<lastname>` are to be replaced with your first- and last name respectively.

(e) Hand in your solution electronically by uploading the Zip file to ILIAS.

(f) Remember that you have to return the Raspberry Pi and accessories on Tuesday, 16. May 2023. Bring all the equipment to the exercise hour and we will return your deposit of 120 francs. If it is not possible for you to be there at the return date, please contact one of the teaching assistants to make an appointment.

Bonus Task: Timer

This exercise is **optional**. Implement a simple kitchen timer which has the following functionality.

Start In the initial state, the timer is off and the LED of the first segment (bottom) is turned on.

Set Pressing `BUTTON1` will move the light up by one segment. The position of the light indicates the number of minutes the timer will run. For example, pressing the button twice will result in the third LED being turned on. This sets the timer to three minutes.

Wait The timer can be started by pressing `BUTTON2`. The active LED should start to blink in order to signalize that the timer is running. Additionally, every minute that passes, the active (blinking) light moves one segment back (down). This allows the user to see how much time is remaining.

Alarm When the timer runs out, the active light should be back in the starting position. An alarm is played through the buzzer and pressing `BUTTON1` will stop the alarm and turn off the timer (starting state).

You are of course allowed to extend the described functionality with your own ideas. Have fun!

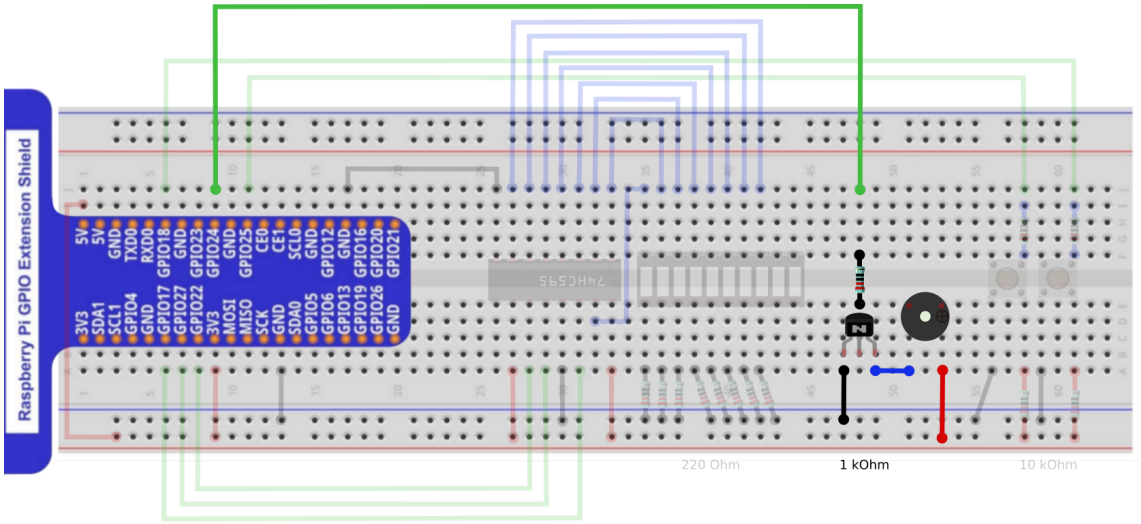

Figure 1: Circuit diagram for the buzzer with an NPN-transistor. Red: power connection to positive electrode (+). Black: power connection to negative electrode (−). Green: data connections for input- and output signals. Blue: other internal data connections.

A Buzzer

The buzzer can be accessed exactly the same way as the LED. As shown in figure 1 it is connected to pin 24. You can use `digitalWrite` to set the output signal to HIGH (sound) or LOW (no sound).

The buzzer will play a constant sound when turned on. If you are in the process of implementing and debugging your code, having the buzzer turned on might be annoying at times. We recommend that you temporarily reroute the buzzer signal to the LED. This can be done physically by unplugging the green cable in figure 1 and moving the LED cable from pin 21 to pin 24. Alternatively you can also redefine the `LED_PIN` field directly in the code.

If you wish you can connect the buzzer to 5V power (instead of 3V). This will increase the volume. For this, simply move the red cable in figure 1 to the pin labeled with “5V”.