Project Assignment 2
Interrupts

EEE212 - Microprocessors
2021-2022 Fall

Part A - Algorithm Implementation (60 pts)

Imagine that we have a simple function $y = f(x)$, such that, $y = a + bx$, and we do not know the values of a and b. Our goal is to estimate a and b, using a sequence of observations (y_i, x_i), where $i \in \{1, \dots\}$ are the observation indices. While there exists different methods to estimate a and b based on past observations, we will take a *learning* based approach and use a simplified version of the famous gradient descent algorithm in machine learning.

Let us have random initial guesses for a and b, which we denote by a_1 and b_1. Then, we make our first observation (y_1, x_1). Normally, by using a_1 and b_1, our model would guess $\hat{y}_1 = a_1 + b_1 x_1$, and its (squared) *error* would be $(y_1 - \hat{y}_1)^2$, where y_1 is the true value, and $\hat{y}_1$ is our model's guess. We can rewrite this squared loss as,

$$l_1 = (y_1 - a_1 - b_1 x_1)^2 \; . \tag{1}$$

Now, our aim is obviously to make the loss in (1) smaller. How can we do that? We can change our guesses for the parameters a and b, such that our model's guess is closer to the true value y_1! Normally, as the next, we can use simple ideas from calculus to do so. If we take the derivative of the loss function in (1) with respect to a_1 and b_1, we obtain the following (note: you will not need to calculate the exact derivative in the assignment, this is just an explanation/background information for the simpler solution you will follow),

$$\frac{\partial l_1}{\partial a_1} = -2(y_1 - \hat{y}_1) \tag{2}$$

$$\frac{\partial l_1}{\partial b_1} = -2x_1(y_1 - \hat{y}_1) \; , \tag{3}$$

Note that we have $(y_1 - \hat{y}_1) = (y_1 - a_1 - b_1 x_1)$. Now in (2) and (3), we have the derivatives (gradients) of our loss function in (1) with respect to our parameter estimates a_1 and b_1. These basically show the direction in which the loss is increasing. For instance, if the value in (2) is positive, it means that increasing a_1 will increase the loss. You can check that idea for simple functions such as $y = x^2$ and $y = x^3$, by taking the derivative with respect to x, and observe how the function changes with x by looking at the derivative at that point. If you understand how this works, you will have understood the basic idea behind most of the amazing things artificial intelligence is doing.

Going back to our problem. Since we want to *decrease* the loss, we will change our parameter estimates a_1 and b_1 by doing the opposite of what are suggested by (2) and (3). If (2) is positive, we will decrease a_1, and if it is negative, we will increase a_1. We will do the same for b_1. That is, after making our first observation, we will do the following parameter updates,

$$a_2 = \begin{cases} a_1 + 1 & \text{if } (y_1 - \hat{y}_1) > 0 \\ a_1 & \text{if } (y_1 - \hat{y}_1) = 0 \\ a_1 - 1 & \text{if } (y_1 - \hat{y}_1) < 0 \; , \end{cases} \tag{4}$$

and,

$$b_2 = \begin{cases} b_1 + 1 & \text{if } x_1(y_1 - \hat{y}_1) > 0 \\ b_1 & \text{if } x_1(y_1 - \hat{y}_1) = 0 \\ b_1 - 1 & \text{if } x_1(y_1 - \hat{y}_1) < 0 \; . \end{cases} \tag{5}$$

In (4) and (5), we are simply increasing or decreasing our parameter estimates by 1, depending on the signs of the derivatives in (2) and (3). After we update our guesses to a_2 and b_2, new observations will keep arriving. As they arrive, we will keep updating our parameters in the same way.

We will assume that the auxiliary variables x_i and the unknown parameters a and b are 4-bit signed integers. That is $x_i, a, b \in \{-8, -7, \ldots, 7\}$. You will do the following to sequentially *learn* a and b.

Algorithm 1 EEE212 Gradient Descent

```
1: Store the true values $a = -3$ and $b = 2$ in two registers (e.g., 40H and 41H).
2: Arbitrarily form your initial estimates $a_1$ and $b_1$ as integers between $-8$ and 7 and store them in
   another register pair (e.g., 42H and 43H)
3: for $i = 1, \ldots, 1000$ do
4:     Query the $i$th 4-bit signed integer from the dataset provided to you, this is your $x_i$.
5:     Calculate the true value $y_i = a + bx_i$
6:     Calculate your current estimate $\hat{y}_i = a_i + b_i x_i$
7:     Find $a_{i+1}$ according to (4)
8:     Find $b_{i+1}$ according to (5)
9:     Write your new parameter estimates ($a_{i+1}$ and $b_{i+1}$) in register 42H and 43H
```

Note1: In STEPS 1, 2, and 4 you should **make sure that if the D3 (most significant bit) of the 4-bit number (D3-D2-D1-D0) is 1, the register that will keep this number preserves the sign bit, i.e. D7-D6-D5-D4 should all be 1 as well. If D3 is zero, these upper bits should be 0 instead. Depending on D3, you can use `ORL source,#0F0H` or `ANL source,#0FH` instructions, respectively. Note that in the dataset provided to you (`random_4bit_signed.txt`), the value of x_i's are stored in the least 4 significant bits (D3-D2-D1-D0) originally.**

Part B - Using Timer Interrupt (40 pts)

Now, we are going to use LED to keep track of the updates in (a_i and b_i). The first line of the LED should show the original weights (a and b), and the second line should print the current value of the parameters (a_i and b_i) (which, of course, needs to be updated with a certain refresh rate to show the changing values).

Using one of the Timer interrupts, decide on its timer mode and initialize your offset to a value so that every 3 or 4 iterations, an interrupt will be triggered. (Your TA calculated this offset approximately as ~(-50) MCs based on his implementation.) When the interrupt is triggered, the LED will update its second line to show the current value of the parameters (a_i and b_i). However, this number will differ according to your own code. Therefore, after completing Part A, count the total number of machine cycles to finish one update (one iteration of the loop) in your implementation and multiply this by ~3 to find the timer offset value (i.e., how frequently an interrupt should occur). Then, if needed, monitoring the LED, tune this number so that you can follow the parameters by eye and see the trace produced by the above gradient-descent approach until they converge.

Note2: A python code (`grad_descent.ipynb`) is provided to you check the correctness of your implementation.

Note3: At all times, the first line of the LCD screen should display the original weights (a and b), and the second line of the screen should show the changing parameters (a_i and b_i). In other words, unlike the previous lab, the displayed values on the screen should never disappear unless they are being replaced by updated values. These four values should always be represented in signed decimal on the LCD screen. It is advised for you to reprint the entire screen (not merely the changing parameters) every time a timer interrupt is serviced. Reprinting the second line is harder to implement but also acceptable.