Lab 1

In this lab, we will learn how to work with an ARM processor and the basics of ARM assembly by programming some common routines. The following lecture content will be required for conducting this lab: compare instructions, branch instructions, shift instructions, load and store instructions, subroutines and function calls.

Part 1

In this part, we will implement a well-known optimization technique, namely stochastic gradient descent (SGD), which is the backbone of a wide range of machine learning algorithms. We will use the SGD technique to solve a simple problem: finding the square root of an integer number.

Let $x^2 = a$, where $a \in Z^+$, the task is to calculate $x = \text{round}(\sqrt{a})$. Let $L = \frac{(x^2-a)^2}{4}$ be a loss function. The square root of a can be found (approximated) by finding x^* that minimizes the loss function L, where $x^* = \text{argmin}_{\forall x} L$.

To find (estimate) x^* using the SGD technique, at the first time step, $i = 0$, we start with a (random) value of x_0. The algorithm updates the estimate of x^* by performing the following computation at each iteration

$$x_{i+1} = x_i - \gamma * L'(x_i) = x_i - \gamma * (x_i^2 - a)x_i,$$

where γ $(0 < \gamma < 1)$ is the learning rate, i is the time step. To simplify the update equation, we choose γ to be in the form of 2^{-k} and $k \in Z^+$. In practice, the value of $\gamma * (x^2 - a)x$ is often constrained to be within an interval $(-t, t),\ t > 0$, to enable numerical stability.

In C, finding the square root of an integer number a using the SGD technique with 100 iterations (`cnt=100`) can be implemented as:

```
int sqrtIter(int a, int xi, int cnt, int k, int t)
{
   for (int i=0; i<cnt; i++)
   {
        int step = ((xi*xi-a)*xi)>>k;
        if (step > t)
           step = t;
        else if(step< -t)
           step = -t;
        xi = xi - step;
   }
   return xi;
}

int x = sqrtIter(168, 1, 100, 10, 2); // Output: 13
```

In addition, the iterative approach used in the `sqrtIter` function can be alternatively transformed to a recursive manner as follows.

```
int sqrtRecur(int a, int xi, int cnt, int k, int t)
{
    if (cnt == 0)
        return xi;
    else
    {
        int grad = ((xi * xi - a) * xi) >> k;
        if (grad > t)
            grad = t;
        else if (grad < -t)
            grad = -t;
        xi = xi - grad;
        return sqrtRecur(a, xi, cnt - 1, k, t);
    }
}

int x = sqrtRecur(168, 1, 100, 10, 2); // Output: 13
```

In this part, your tasks consit of writing assembly programs to find the square root of an integer number a using different approaches. Note that in all cases, `xi` is stored in `r0`, `a` is stored in `r1`, and `cnt` is stored in `r2`. The values of `k`, and `t` are treated as immediate values and are fixed to 10, and 2, respectively. *In addition, it is recommended that you perform the second exercise of Part 1 only once you will have covered function calls in the lecture.*

Part 1 Exercises

1. Write an assembly program that implements the `sqrtIter` function.
2. Write an assembly program that implements the `sqrtRecur` function using subroutines and function calls.

Note

If you are experiencing the warning **Function calls nested too many levels (> 32)**, e.g., when `cnt>32` for the `sqrtRecur` function, you can turn off the warning message by navigating to the **Settings/Debugging Check** menu of the simulator, then please untick the following option: **Function nesting too deep**.

Part 2

In this part, your task is to calculate the norm of a vector (array). Let $\boldsymbol{x} = \{x_0, x_1, \ldots\ x_{n-1}\}$ be a vector of size n. The norm of $\boldsymbol{x}$ is given as $||\boldsymbol{x}|| = \sqrt{\frac{x_0^2+x_1^2+\cdots+x_{n-1}^2}{n}}$.

Note that you can reuse one of the assembly programs in Part 1 to calculate the square root in the above equation. In addition, we only consider the length of $\boldsymbol{x}$ to be a power of 2, thus the division can be implemented by a right-shift operation.

Below is a C program that calculates the norm of a given vector (array) of size 4.

```c
// Initialization
int array[] = {5, 6, 7, 8};
size_t n = sizeof(array) / sizeof(array[0]);
int log2_n = 0;
int *ptr;
int tmp = 0;
int norm = 1;
int cnt = 100;
int k = 10;
int t = 2;

// Calculate log_2(n)
while ((1 << log2_n) < n)
    log2_n++;

// Calculate the norm of the given array
ptr = &array[0];
for (int i = 0; i < n; i++)
{
    tmp += (*ptr) * (*ptr);
    ptr++;
}
tmp = tmp >> log2_n;
norm = sqrtIter(tmp, norm, cnt, k, t); // Output: 7
```

Part 2 Exercises

1. Understand the C program that calculates the norm of a vector (array), add comments to the C program if necessary.

2. Write an assembly that calculates the norm of a vector (array). Note that the length of the array is also given as a parameter of the program. If you did not manage to implement the `sqrtIter` function in part 1, simply call a dummy function that returns its argument and you will not loose any point as long as you can demonstrate you can call a function correctly (you may want to wait before implementing the function call until this has been covered in the lectures).

Part 3

In this part, your task is to implement an algorithm that "centers" a vector (array). It is often necessary to ensure that a signal is "centered" (that is, its average is 0). For example, DC signals can damage a loudspeaker, so it is important to center an audio signal to remove DC components before sending the signal to the speaker.

You can center a signal by calculating the mean (average value) of the signal and subtracting the average from every sample of the signal. Write an ARM assembly program to center a signal.

The program should be able to accept the signal length as an input parameter. In order to simplify calculations, work with the assumption that only signal lengths that are powers of two can be passed to the program.

The following C program implements the above array centering algorithm.

```c
// Initialization
int array[] = {3, 4, 5, 4};
size_t n = 4;
int mean = 0;
int *ptr;

// Calculate log_2(n)
int log2_n = 0;
while ((1 << log2_n) < n)  log2_n++;

// Calculate the mean of the given array
ptr = &array[0];
for (int i = 0; i < n; i++)
{
   mean += *ptr;
   ptr++;
}
mean = mean >> log2_n;

// Center the given array
ptr = &array[0];
for (int i = 0; i < n; i++)
{
   *ptr -= mean;
   ptr++;
}

// Output: array={-1,0,1,0}
```

Part 3 Exercises

1. Understand the C program of the array centering algorithm, add comments to the C program if necessary.
2. Write an assembly program that performs the array centering algorithm. Note that the length of the array is also given as an input parameter of the program. Store the resulting centered signal 'in place' - i.e. in the same memory location that the input signal is passed in.

Part 4

In this part, your task is to implement the selection sort algorithm. Below is an example of the selection sort algorithm implemented in a C program.

```c
int array[] = {4, 2, 1, 4, -1};
size_t n = sizeof(array) / sizeof(array[0]);
int *ptr = &array[0];
for (int i = 0; i < n-1; i++)
{
    int tmp = *(ptr + i);
    int cur_min_idx = i;
    for (int j = i + 1; j < n; j++)
    {
        if (tmp > *(ptr + j))
        {
            tmp = *(ptr + j);
            cur_min_idx = j;
        }
    }
    // Swap
    tmp = *(ptr + i);
    *(ptr + i) = *(ptr + cur_min_idx);
    *(ptr + cur_min_idx) = tmp;
}

// Output: array={-1, 1, 2, 4, 4}
```

Part 4 Exercises

1. Understand the C program of the selection sort algorithm, add comments to the C program if necessary.
2. Write an assembly program that implements the selection sort algorithm to sort a given array in ascending order. Note that the length of the array is also given as an input parameter of the program.

Grading and Report

Your grade will be evaluated through the deliverables of your work during the demo (**70%**) (basically showing us the working programs), your answers to the questions raised by the TA's during the demo (**10%**), and your lab report (**20%**).

Grade distribution of the demo:

- Part 1.1: assembly implementation of `sqrtIter` (**5%**).
- Part 1.2: assembly implementation of `sqrtRecur` (**10%**).
- Part 2: assembly program that calculates the norm of an array (**15%**).
- Part 3: assembly program that centers an array (**15%**).
- Part 4: assembly program that implements selection sort algorithm (**25%**).

Write up a short report (~1 page per part) that should include the following information.

- A brief description of each part completed (do not include the entire code in the body of the report).
- The approach taken (e.g., using subroutines, stack, etc.).
- The challenges faced, if any, and your solutions.
- Possible improvevement to the programs.

Your final submission should be sumitted on myCourses. The deadline for the submission and the report is **Friday, 16 October 2020**. A **single compressed folder** should be submitted in the **.zip** format, that contains the following files:

- Your lab report in **pdf** format: **StudentID_FullName_Lab1_report.pdf**
- The assembly program for Part 1.1: **part1_1.s**
- The assembly program for Part 1.2: **part1_2.s**
- The assembly program for Part 2: **part2.s**
- The assembly program for Part 3: **part3.s**
- The assembly program for Part 4: **part4.s**

Important

Note that we will check every submission (code and report) for possible plagiarism. All suspected cases will be reported to the faculty. Please make sure to familiarize yourself with the course policies regarding Academic Integrity and remember that all the labs **are to be done individually**. Also please note that you are not allowed to use a compiler to generate the assembly code and must write it by hand.

The demo will take place via Zoom during the week of 5-9 October 2020 on the day of you **assigned lab session day**. We will provide a registration system for the demo the week before. You will need to answer live questions during the demo with your screen shared to onstrate the working program.