ECE 30
Introduction to Computer Engineering
Programming Project: Fall 2023
In-place Merge Sort
October 15, 2023

Project TA: Yatharth Vyas (y1vyas@ucsd.edu)

1 Project Description

The goal of the project is to write a program that implements an in-place merge sort algorithm in the LEGv8 assembly language. Overall, the program will import an array of data and sort it's values using the merge sort algorithm without using any extra memory.

2 Merge Sort Algorithm Description

The overview of a merge sort algorithm is actually conceptually simple:

1. Split the input array in two sub-arrays.
2. Recursively call merge sort on the left sub-array and right sub-array, respectively.
3. Merge the sub-arrays together element-by-element in a sorted order.
4. Return the now-sorted array to the calling function.

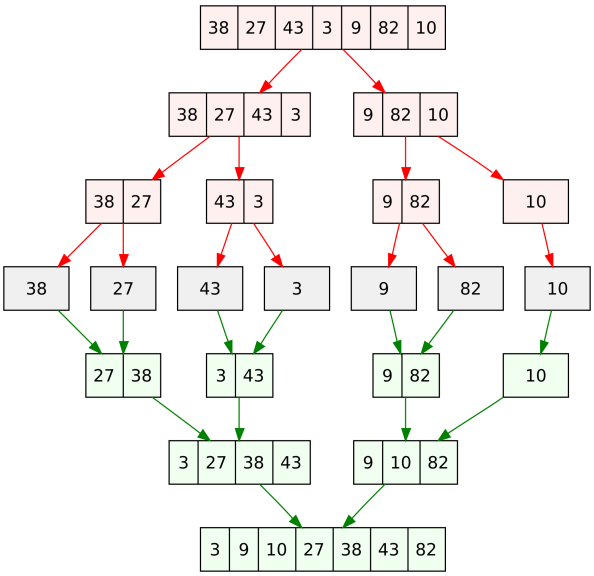

Figure 1: Merge Sort Visualization By Vineet Kumar at English Wikipedia - Transferred from en.wikipedia to Commons by Eric Bauman using CommonsHelper., Public Domain, https://commons.wikimedia.org/w/index.php?curid=8004317

The recursive merge sort will keep dividing the sub-array until it holds a single element. A single element sub-array (the base case) is inherently sorted. It is highly recommended that

you read over the description of a merge sort on Merge sort on Wikipedia (it's a very good overview), and graphical view of the sort at: www.sorting-algorithms.com/merge-sort.

Important:

- Your project **must use the provided template file**. While you are free to add any further helper functions, your code must implement the following functions according to the specifications given below.
- **Interoperability.** The grader should be able to remove any of these functions (and any non-essential helper functions it may use) from your code and use it in another person's programming assignment without issue (i.e. do not flip the variables passed to registers).

3 Implementation

In this project, you have to write two recursive functions, namely ***MergeSort*** and ***Inplace-Merge*** along with two helper functions: ***GetNextGap*** and ***Swap*** to implement the merge sort algorithm in the LEGv8 assembly language. Some details to note:

- Use the prototype of the procedure as mentioned below and given to you in the template. **Don't change** the registers or the arguments passed to the procedures or the values returned. **Changing input/output** registers results in loosing all the points.
- Follow the "Procedure Call Convention" for calling procedures, passing registers and managing the stack. The procedures should not make any assumptions about the implementation of other procedures, except for the name of input/output registers (as indicated below) and the conventions mentioned in the course. A code that works correctly but does not follow conventions will be penalized.
- We expect your code to be well-commented. Each instruction should be commented with a meaningful description of the operation. For example, this comment is bad as it tells you nothing:

```
// x1 gets x2 - 1
sub x1, x2, #1
```

A good comment should explain the meaning behind the instruction. A better example would be:

```
// Initialize loop counter x1 to n-1
sub x1, x2, #1
```

In this project we will be performing the merge-sort in place, ie. while performing the sorting operation, we will not be making use of any extra memory outside of the array. To ensure this, we will be making use of the **Shell Sorting method** to merge the subarrays. Read https://www.geeksforgeeks.org/efficiently-merging-two-sorted-arrays-with-o1-extra-space/ for an in-depth explanation on how this approach works.

To demonstrate this algorithm, we can visualize how this works in merging two arrays: $arr_1 = [1, 3, 10, 11]$ and $arr_2 = [2, 2, 7]$.

Please note that the *ceil* function in the below algorithm rounds off to the nearest greater integer.

1. First, we initialize $gap = ciel((len(arr_1) + len(arr_2))/2) = ceil(7/2) = ceil(3.5) = 4$
2. Now, we think of the two arrays as a continuous single array such that arr_2 is just after arr_1 to appear like $[1, 3, 10, 11, 2, 2, 7]$. Here we compare and swap all elements that are *gap* distance apart such that they end up being in an ascending order.
3. If $gap > 1$, We update *gap* to be $ceil(gap/2)$ and go to step 2; else, we stop.

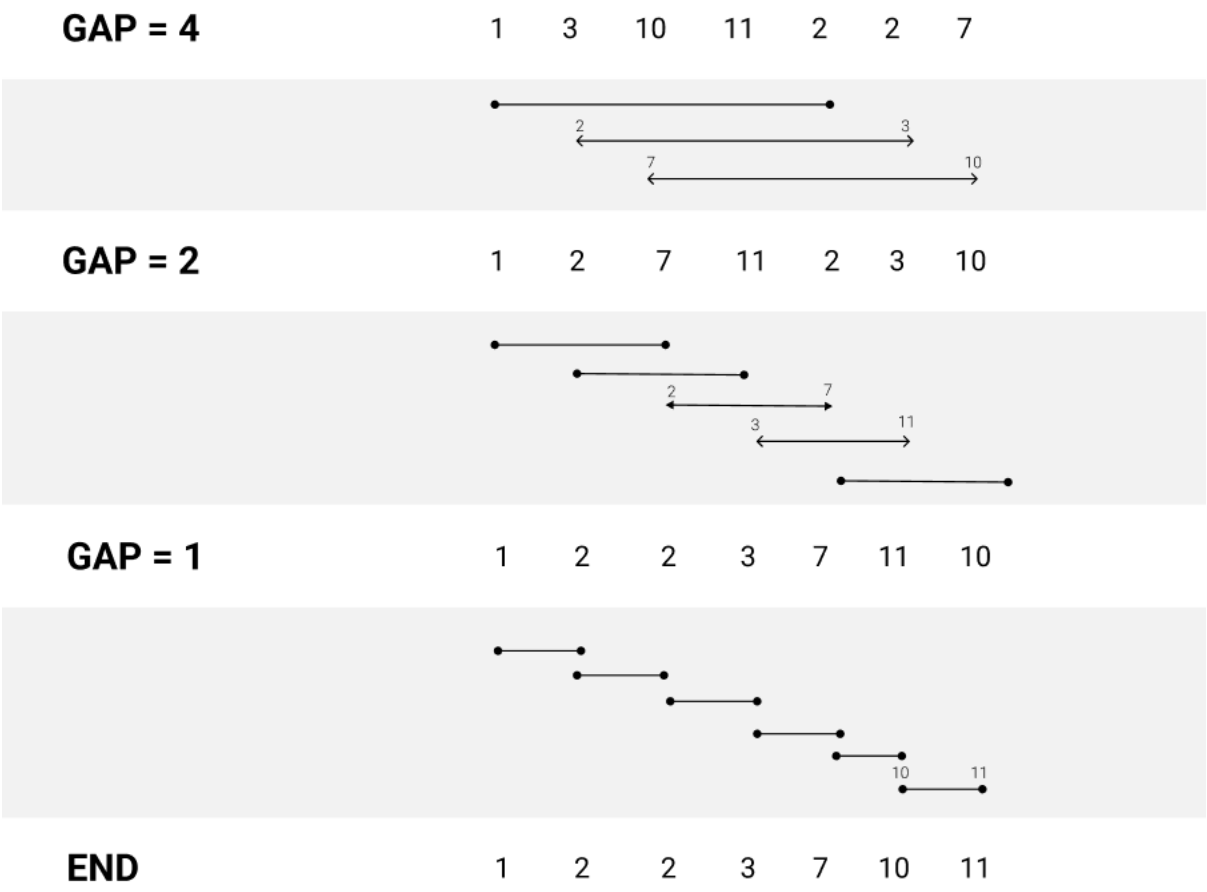

Figure 2: In-place Merge Example

3.1 Function 1: Swap(a, b)

Swaps two values pointed by a and b. It is a very generic swapping operation that you may have encountered in C language.

3.1.1 Parameters

- X0: the address of the first value
- X1: the address of the second value

3.1.2 Return Value

- This function does not return anything.

3.1.3 Pseudo-code

Swapping two values in the addresses X0 and X1. (note that you will need a temporary register).

3.2 C Procedure

```
// Function for swapping
void swap(int arr[], int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

3.2.1 Examples

Inputs:

- X0=100 (and suppose the value stored in address 100 is a)
- X1=108 (and suppose the value stored in address 108 is b)

When exiting:

- The value stored in address X0=100 is b
- The value stored in address X1=108 is a

3.3 Function 2: GetNextGap(gap)

Updates the value of gap and returns the update value of gap next iteration of shell sorting in the in-place merge step.

3.3.1 Parameters

- x0: The previous value for gap

3.3.2 Return Value

- x0: The new value for gap.

3.3.3 Pseudo-code

Algorithm 1 Function to get the Next Gap

```
1: function GetNextGap(gap)
2:     if gap ≤ 1 then
3:         return 0
4:     else
5:         return ceil(gap/2)
6:     end if
7: end function
```

Hint: $ceil(gap/2) = (gap/2) + (gap\&1) = floor((gap+1)/2)$
ie. The Ceil function can be implemented by dividing gap by 2 and adding 1 if gap was initially an odd number. The logical AND operation with 1 results in the output being 1 if the lowest bit of the number in it's binary format is 1 which is only true in case of odd numbers.

3.4 C Procedure

```
// Calculating next gap
int nextGap(int gap) {
    if (gap <= 1)
        return 0;

    return (gap+1)/2; // ceil(gap/2)
}
```

3.4.1 Example

Input:

- gap: 11

Output:

- gap: 6

3.5 Function 3: inPlaceMerge(start, end, gap)

This function merges two contiguous sorted sub-arrays into one sorted sub-array without using extra memory (in-place).

3.5.1 Parameters

- x0: The address of the starting element of the first sub-array.
- x1: The address of the last element of the second sub-array.
- x2: The gap used in comparisons for shell sorting

3.5.2 Return Value

- This function does not return anything.

3.5.3 Pseudo-code

Algorithm 2 In-place Merge

```
1: function InPlaceMerge(start, end, gap)
2:     if gap < 1 then
3:         return
4:     end if
5:     for left ← start to left + gap ≤ end do
6:         right ← left + gap
7:         if DATA[left] > DATA[right] then
8:             Swap(left, right)
9:         end if
10:    end for
11:    gap ← GetNextGap(gap)
12:    InPlaceMerge(start, end, gap)
13: end function
```

3.6 C Procedure

```
// Merging the subarrays using shell sorting in O(1) space
// Time Complexity: O(nlog n)
void inPlaceMerge(int arr[], int start, int end, int gap) {
    if (gap < 1)
        return;

    for(int left = start; left + gap <= end; left++) {
        int right = left + gap;

        if (arr[left] > arr[right])
            swap(arr, left, right);
    }

    gap = nextGap(gap);
```

```
    inPlaceMerge(arr, start, end, gap);
}
```

3.6.1 Example

Before start:

- first sub-array: $[1, 3, 10, 11]$
- second sub-array: $[2, 2, 7]$

When exiting:

- sorted combined array: $[1, 2, 2, 3, 7, 10, 11]$

3.7 Function 4: MergeSort(start, end)

This function sorts an array.

3.7.1 Parameters

- x0: The starting address of the array.
- x1: The ending address of the array.

3.7.2 Return Value

- This function does not return anything.

3.7.3 Pseudo-code

Algorithm 3 Merge Sort

```
1: function MergeSort(start, end)
2:     if start = end then
3:         return
4:     else
5:         mid = (start + end) / 2
6:         MergeSort(start, mid)
7:         MergeSort(mid + 1, end)
8:         gap ⟵ GetNextGap(end − start + 1)
9:         return InPlaceMerge(p, q, gap)
10:    end if
11: end function
```

Note: While finding mid and gap, *start* and *end* will be address locations. The above pseudocode treats them as integer indexes but for the project, you might have to divide them by 8 to convert those addresses (eg: Data[00008]: 00008 / 8 = 1) to indexes. Consequently, you will have to multiply them by 8 to get the addresses back.

3.8 C Procedure

```
// Merge Sort (Recursive)
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        inPlaceMerge(arr, left, right, nextGap(right - left + 1));     // recursive functio
    }
}
```

3.8.1 Example

Before start:

- input array: $[1, 4, 3, 5, 2]$

When exiting:

- sorted array: $[1, 2, 3, 4, 5]$

3.9 [Bonus Function] BinarySearch(start, end, value)

Binary Search is an algorithm that searches for a value in a sorted array in logarithmic time complexity. This function performs binary search to search for a target in the sorted array by repeatedly dividing it in half until the value is found. Wikipedia has a great explanation of how this algorithm works and it is recommended to give it a read.

3.9.1 Instructions for Bonus Question

- This question is an opportunity for all students to score 5 bonus marks for this course.
- You cannot ask any TAs for help with the bonus question.
- You will not lose any marks if you do not attempt or are unable to complete this function.

3.9.2 Parameters

- x0: The starting address of the array.
- x1: The ending address of the array.
- x2: The target value to be searched in the array.

3.9.3 Return Value

- x3: $True$ or 1 if the target value is present in the array otherwise $False$ or 0

3.9.4 Pseudo-code

Algorithm 4 Binary Search

```
1: function BinarySearch(start, end, target)
2:     while start ≤ end do
3:         mid ← ⌊(start+end)/2⌋
4:         if arr[mid] = target then
5:             return 1                                  ▷ True: Target Found
6:         else if arr[mid] < key then
7:             start ← mid + 1
8:         else
9:             end ← mid − 1
10:        end if
11:    end while
12:    return 0                                          ▷ False: Target Not Found
13: end function
```

Note: While finding mid and gap, *start* and *end* will be address locations. The above pseudocode treats them as integer indexes but for the project, you might have to divide them by 8 to convert those addresses (eg: Data[00008]: 00008 / 8 = 1) to indexes. Consequently, you will have to multiply them by 8 to get the addresses back.

3.10 C Procedure

```
// Bonus Extension: Binary Search
bool binarySearch(int arr[], int left, int right, int target) {
    if (right >= left) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) // if target is found at the middle
            return true;

        if (arr[mid] > target)  // if target is smaller than mid
            return binarySearch(arr, left, mid - 1, target);

        return binarySearch(arr, mid + 1, right, target);   // if target is greater than mi
    }

    return false;
}
```

3.10.1 Example

Example 1:
Before start:

- input array: $[1, 4, 3, 5, 2]$
- target: 2

When exiting:

- $x3$: 1 (True: Target found)

Example 2:
Before start:

- input array: $[1, 4, 3, 5, 2]$
- target: 6

When exiting:

- $x3$: 0 (False: Target not found)

4 Instructions

- Please use the template provided for the starter code and the test input data.
- You must submit your solution on Canvas in a completed file ABC_XYZ_2023_project.s, where ABC and XYZ are the email addresses exluding @ucsd.edu for each student. For example, if Sherlock Holmes and John Watson were working together on a submission, the file name would be: sholmes_jwatson_2023_project.s
- Fill the names and PID of each student in the file.

- Each project team contributes their own unique code – no copying, cheating, or hiring help. We will check the programs against each other using automated tools. These tools are VERY effective, and cheaters WILL get caught!
- Please start this project early!
- Try to test the behavior of each function independently, rather than trying to code all of them at once. It will make debugging far easier. Also, we will test functions individually.

5 Grading

- Swap (2 points): 2 test cases, 1 point each
- GetNextGap (2 points): 2 test cases, 1 point each
- InplaceMerge (5 points): 5 test cases, 1 point each
- MergeSort (4 points): 4 test cases, 1 point each
- Extensive testing (2 points): points deducted if your code does not compile, run, or terminate correctly
- [BONUS] BinarySearch (extra 5 points): 5 test cases, 1 point each

Note: Your comments will not be graded, but your TA may refuse to proofread your code if your code is not well-commented.

6 Miscellaneous (please read)

- You MUST show up in YOUR week 5 lab session. Otherwise, your partner has the right to request a new partner. If you cannot make it, notify the project TA (Yatharth) in advance.
- You may use ANY of week 5 and week 7 lab sessions to ask Yatharth any project-related questions as if the lab sessions are Yatharth's office hours. Please utilize Yatharth as a resource!
- Yatharth's office hour: 8:30-10am Wednesdays on week 6, 8, and 9 in Jacobs Hall (4506) and Week 10 on Zoom (Time: TBA).
- If you get stuck on any function, reviewing the following lecture video might help, especially S4E4-S5E5, S6E6 for recursion (also available on Canvas).

  `youtube.com/watch?v=myYu7DXsQs4&list=PLFxZj9wj5LQwxnB3hpi124YcAYpMDABnf`
- If encountering a syntactic error, Legiss will report the instruction that caused the error in the bottom of the left panel (see also the bottom-right panel). This information is often helpful.
- You are not allowed to use any LLMs or AI tools like ChatGPT to generate code snippets for the project.
- There is a BUG in LEGv8: You cannot have comments right after the instruction. You must use a SPACE to separate them!

```
LDUR x11, [x0, #0]// Causing a crash
LDUR x11, [x0, #0] // Okay
```

- Complaints about teammates will NOT be considered after the beginning of week 8.