Assignment #1:
MIPS Assembly Programming

CAS3102-01 (Architecture of Computers), Spring 2025

Welcome to the first programming assignment of CAS3102-01 @ Yonsei University in Spring 2025! In this programming assignment, you will write various functions using the MIPS assembly language (in particular, MIPS R2000 assembly language). The purposes of this assignment are (1) to make you get familiar with the MIPS assembly language, and (2) to give you hands-on experience with SPIM, a MIPS processor simulator designed to run the MIPS assembly code. Note that this assignment deals with the MIPS assembly language, not the MIPS machine language, which consists of 0s and 1s.

Before working on the assignment, we strongly recommend you to study the textbook's Chapter A.10 for more details on the assembly language and the SPIM simulator.

Prerequisites

As the first step of this assignment, you should bring up a Virtual Machine (VM) that you will use throughout the semester.

1-a. Intel/AMD CPU Users:

First, install VMware Workstation Player by visiting this site. Then, download a compressed image from https://drive.google.com/file/d/1f0PIurypF6gGC449AUGHx2wwby5ta1Ml/view?usp=share_link. After that, decompress the image and import the image into VMware Workstation Player.

1-b. Apple Silicon Users:

Apple Silicon implements ARM64 ISA, not x86-64 ISA, so you should use a different virtual machine image. First, install UTM, an alternative to VMware Workstation Player, by referring to this site. Then, get the image from https://drive.google.com/file/d/1XLgQMF0EW02v8k7sVOOYC44HfHzPI2D7/view?usp=share_link. After that, decompress the image and import it to UTM.

2. Both Intel/AMD CPU and Apple Silicon Users:

Boot the image and you will encounter a login screen. Both the ID and password are `csi3102`. Log into the account by typing in the password. You will use the VMware image throughout this semester, so make sure you properly set up the VMware image and can successfully log in.

Preparing a MIPS Simulation Environment

To make sure that you have a proper version of the SPIM simulator, log in to the VM, launch a terminal, and type `which spim` in the terminal. You should see an output similar to the following:

```
$ which spim
/usr/bin/spim
```

Now, try executing the SPIM simulator by typing `spim`. You should see the following output:

```
$ spim
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
(spim)
```

Make sure that you are using SPIM version 8.0. To inspect many useful built-in commands provided by SPIM, type `help` followed by the enter key in SPIM's command-line. Type `quit` and press the Enter key to exit SPIM's command-line environment.

If the above commands do not work, try executing the following command in the terminal: `sudo apt update && sudo apt -y install spim`. When the VM asks for a root password, type `csi3102`.

Downloading the Assignment

Please download this assignment and perform a sanity check by typing the following commands:

```
~$ wget https://hpcp.yonsei.ac.kr/cas3102-2025sp/assn1-20250402.tar.gz
~$ tar zvxf ./assn1-20250402.tar.gz
~$ cd ./assn1
~/assn1$ ls
examples     Q1_FourWayAddAndSub.asm     Q2_GCD.asm     Q3_CompareStrings.asm
Q4_SortArray.asm  Q5_SortedArrayInsert.asm  Q6_BinaryHeap.asm
```

Example MIPS Assembly Code

As a small guide on how to write MIPS assembly code for the SPIM simulator, we provide four example code snippets under the `examples/` directory. The examples are named `0_helloWorld.asm`, `1_addTwoInts.asm`, `2_add1To10.asm`, and `3_updateMemory.asm`. The four code snippets demonstrate four different code patterns.

1_helloWorld.asm

This code demonstrates a simple "Hello World!" example. The code consists of a single function named `main`. The `main` label is the start point of any SPIM simulator invocation; the SPIM simulator always jumps to the `main` label after it sets up some preparation assembly code (e.g., system calls).

Try executing the file with `spim -file ./examples/0_helloWorld.asm`. You should see the following output:

```
~/assn1$ spim -file ./examples/1_helloWorld.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
```

```
Hello world
```

When you invoke this file with the above command, the SPIM simulator does the following operations:

1) Load the file named `./examples/0_helloWorld.asm`, which contains MIPS assembly code
2) Initialize a MIPS simulation environment
3) Invoke the function named `main` by executing a `jal` instruction with `main` as the target label
4) Print a string "`Hello world\n`" to the terminal
5) Function `main` returns control by invoking `jr $ra` instruction.
6) The simulation gets terminated.

The code consists of two `.`-prefixed regions: `.data` and `.text`. `.data` and `.text` correspond to the data segment and the text segment of the MIPS ISA, respectively. You can think of the two segments as: `.data` is where the heap and stack get located at, and `.text` is where the assembly instructions should be.

You will notice `greet: .asciiz "Hello world\n"` under the data segment. `.asciiz` is an assembler directive for storing a null-terminated string in memory, and `"Hello world\n"` is the string being stored in the memory. Once the string is stored in memory, we need to know the starting memory address of the string. This is done by using identifiers; the starting memory address is associated with the identifier `greet`. The identifiers can later be used by MIPS assembly instructions to access the string. More details on assembler directives can be found in the textbook's chapter A.10.

Now, let's look at the instructions placed in the text segment. You will see that there are three instructions (i.e., `li`, `la`, and `syscall`) followed by `jr $ra` under the `main` label. The first three instructions are needed to invoke a system call that prints a string stored in memory to the terminal. The system call we are currently interested in is `print_string` (see Figure A.9.1 in the textbook), and invoking `print_string` requires `$v0` to have 4, the system call code for `print_str`, and `$a0` to contain the starting memory address of the string to be printed. The `li` and `la` instructions are used to prepare `$v0` and `$a0`. `li` is a pseudoinstruction (i.e., an instruction not defined by the machine language, but defined by the assembly language for easier programming) which loads an immediate into a register (see page A-57 of the textbook). `la` is also a pseudo-instruction which loads a computed address, not the contents, of a specified memory location into a destination register (see page A-66 in the textbook). In our example, the memory location is `greet`, the identifier for the string to be printed to the terminal, and the destination register is `$a0`. The code then invokes `print_string` system call by executing the syscall instruction. After printing the string to the terminal output, the main function returns by executing the `jr $ra` instruction. This is a classic function-terminating instruction pattern in the MIPS ISA.

2_addTwoInts.asm

This example demonstrates how to perform an arithmetic operation on temporary registers (e.g., `$t0`, `$t1`, `$t2`) and utilize other system calls (e.g., `print_int`). In particular, the example performs `$t2` = `$t0` + `$t1` and prints the values of `$t0`, `$t1`, and `$t2` to the terminal.

```
~/assn1$ spim -file ./examples/2_addTwoInts.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
```

```
value1 ($t0): 3
value2 ($t1): 1
sum ($t2): 4
```

3_add1To10.asm

This example demonstrates a loop that adds integers from one to ten and prints the result to the terminal. To implement the loop, note that there are two labels in the code: `main` and `loop`. `$t0` stores the sum and `$t1` serves as the iterator for the loop.

```
~/assn1$ spim -file ./examples/3_add1To10.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
55
```

4_printMemory.asm

This example demonstrates more complex scenarios: (1) invoking a callee function `print_array` from the caller function `main`, (2) loading and printing an array of integers stored in memory, (3) spilling `$ra` to `$s0` before calling `print_array` and recovering `$ra` after the callee function completes.

```
~/assn1$ spim -file ./examples/4_printMemory.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
1
2
3
4
5
6
7
8
```

First, invoking a callee function demands the use of `jal` instruction. You can see this behavior through `jal print_array` instruction under the main label. Note that `$ra` gets spilled to a callee-saved register `$s0` before executing the `jal` instruction, as the `jal` instruction overwrites `$ra` upon execution. Second, under the `print_array_loop` label, the array's elements are accessed with `lw $a0, ($t1)` instruction. This instruction shows the register-relative addressing mechanism of the MIPS ISA; the instruction performs `$a0 = memory[$t1]`.

Problems

By analyzing the examples, we believe you will get a sense of how to write MIPS assembly instructions and execute the assembly instructions on the SPIM simulator.

Now, it is time to write some MIPS assembly code! Please carefully read the instructions below. One thing you should keep in mind is that your implementation of the functions should always follow the MIPS register convention (e.g., backup/restore the saved registers, increment/decrement the stack pointer).

Q1: Adding and Subtracting Four 32-Bit Integers

Implement a function fourWayAddAndSub which takes as input four 32-bit signed integers through `$a0`, `$a1`, `$a2`, and `$a3`, and returns two values `$v0=($a0+$a1+$a2+$a3)` and `$v1=($a0-$a1-$a2-$a3)`. You may assume that no overflow occurs for both the addition and the subtraction.

The four 32-bit signed integers are given through the terminal as decimal numbers.

Example 1-1: 1 + 2 + (-3) + 4 & 1 - 2 - (-3) - 4

```
~/assn1$ spim -file ./Q1_FourWayAddAndSub.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit signed integer (in decimal): 1
Enter a 32-bit signed integer (in decimal): 2
Enter a 32-bit signed integer (in decimal): -3
Enter a 32-bit signed integer (in decimal): 4
INFO: fourWayAddAndSub returned:
INFO:   $v0 = 4
INFO:   $v1 = -2
```

Example 1-2: 1 + 2 + 4 + 8 & 1 - 2 - 4 - 8

```
~/assn1$ spim -file ./Q1_FourWayAddAndSub.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit signed integer (in decimal): 1
Enter a 32-bit signed integer (in decimal): 2
Enter a 32-bit signed integer (in decimal): 4
Enter a 32-bit signed integer (in decimal): 8
INFO: fourWayAddAndSub returned:
INFO:   $v0 = 15
INFO:   $v1 = -13
```

Example 1-3: (-1) + 2 + (-4) + 8 & (-1) - 2 - (-4) - 8

```
~/assn1$ spim -file ./Q1_FourWayAddAndSub.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit signed integer (in decimal): -1
Enter a 32-bit signed integer (in decimal): 2
```

```
Enter a 32-bit signed integer (in decimal): -4
Enter a 32-bit signed integer (in decimal): 8
INFO: fourWayAddAndSub returned:
INFO:   $v0 = 5
INFO:   $v1 = -7
```

Q2: Greatest Common Divisor

Implement a function `calculateGCD` which identifies the greatest common divisor (GCD) of the two 32-bit unsigned integers stored in `$a0` and `$a1`. You may assume that the values of `$a0` and `$a1` are always greater than 0. The function should return the GCD of `$a0` and `$a1` by storing the value of the GCD in `$v0`.

There can be more than one way to implement the function. For example, you may implement the function as a recursive function or by embedding a loop. You are free to implement the function using any method as long as the function correctly calculates and returns the GCD.

Example 2-1: GCD(10, 5) = 5

```
~/assn1$ spim -file ./Q2_GCD.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit unsigned integer (in decimal): 10
Enter a 32-bit unsigned integer (in decimal): 5
INFO: calculateGCD returned:
INFO:   $v0 = 5
```

Example 2-2: GCD(5, 10) = 5

```
~/assn1$ spim -file ./Q2_GCD.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit unsigned integer (in decimal): 5
Enter a 32-bit unsigned integer (in decimal): 10
INFO: calculateGCD returned:
INFO:   $v0 = 5
```

Example 2-3: GCD(128, 100) = 4

```
~/assn1$ spim -file ./Q2_GCD.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit unsigned integer (in decimal): 128
Enter a 32-bit unsigned integer (in decimal): 100
INFO: calculateGCD returned:
INFO:   $v0 = 4
```

Example 2-4: GCD(129, 311) = 1

```
~/assn1$ spim -file ./Q2_GCD.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter a 32-bit unsigned integer (in decimal): 129
Enter a 32-bit unsigned integer (in decimal): 311
INFO: calculateGCD returned:
INFO:   $v0 = 1
```

Q3: String Comparison

Implement a function `compareStrings` that identifies whether an ASCII string is larger than another ASCII string. The function takes the starting memory addresses of one string and the other string as input through `$a0` and `$a1`. If the first string (whose starting memory address is stored in `$a0`) is larger than the second string (whose starting memory address is stored in `$a1`), the function should return a value of 1 through `$v0`. If the second string is larger than the first string, the value of `$v0` should be set to -1. Otherwise (i.e., when the first and the second strings are the same), the values of `$v0` should be 0.

An ASCII string consists of ASCII characters (each ASCII character is one-byte long and its value falls within the range of 0 to 127 inclusive) and ends with a null character whose byte value is zero. You may assume that all the ASCII characters of a string are sequentially stored in a contiguous memory region. For example, if `$a0` = 0x1000, then the first, second, and third ASCII characters of the first string are stored in 0x1000, 0x1001, 0x1002, respectively.

One string A is “larger” than the other string B if one of the following conditions holds:

- If the length of A is greater than the length of B, then A > B.
- If the length of A and the length of B are the same, then use the following C/C++-style pseudocode:

```
// $a0 = &A[0]; $a1 = &B[0];
int $v0 = 0;
for (unsigned int i = 0; i < strlen(A); i++) {
    if (A[i] > B[i]) then { $v0 =  1; break; }
    if (B[i] > A[i]) then { $v0 = -1; break; }
}
return $v0;
```

Example 3-1: $a0 = “This is One String!”, $a1 = “This is Another String!”

```
~/assn1$ spim -file ./Q3_CompareStrings.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the first string: This is One String!
Enter the second string: This is Another String!
INFO: compareStrings returned:
INFO:   $v0 = -1
```

Example 3-2: $a0 = “This is Another String!”, $a1 = “This is One String!”

```
~/assn1$ spim -file ./Q3_CompareStrings.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the first string: This is Another String!
Enter the second string: This is One String!
INFO: compareStrings returned:
INFO:   $v0 = 1
```

Example 3-3: $a0 = “This is One String!”, $a1 = “This is One String!”

```
~/assn1$ spim -file ./Q3_CompareStrings.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the first string: This is One String!
Enter the second string: This is One String!
INFO: compareStrings returned:
INFO:   $v0 = 0
```

Example 3-4: $a0 = “ABCDEFGHIJKMNLOP”, $a1 = “ABCDEFGHIJKLMNOP”

```
~/assn1$ spim -file ./Q3_CompareStrings.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the first string: ABCDEFGHIJKMNLOP
Enter the second string: ABCDEFGHIJKLMNOP
INFO: compareStrings returned:
INFO:   $v0 = 1
```

Example 3-5: $a0 = “ABCDEFGHIJKLMNOP”, $a1 = “abcdefghijklmnop”

```
~/assn1$ spim -file ./Q3_CompareStrings.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the first string: ABCDEFGHIJKLMNOP
Enter the second string: abcdefghijklmnop
INFO: compareStrings returned:
INFO:   $v0 = -1
```

Q4: Sorting an Array

Implement a function `sort_array` which sorts an array of 32-bit signed integers stored in memory. The function takes as input the number of integers in the array through `$a0` and the starting memory address of the array through `$a1`. You may assume that the number of elements is smaller than or equal to 1024.

Example 4-1: $a0 = 10, $a1 = &array[0] (see Q4_SortArray.asm for details)

```
~/assn1$ spim -file ./Q4_SortArray.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to sort: 10
Enter a 32-bit integer (in decimal): 1
Enter a 32-bit integer (in decimal): -2
Enter a 32-bit integer (in decimal): 3
Enter a 32-bit integer (in decimal): -4
Enter a 32-bit integer (in decimal): 5
Enter a 32-bit integer (in decimal): -6
Enter a 32-bit integer (in decimal): 7
Enter a 32-bit integer (in decimal): -8
Enter a 32-bit integer (in decimal): 9
Enter a 32-bit integer (in decimal): -10
1 -2 3 -4 5 -6 7 -8 9 -10
-10 -8 -6 -4 -2 1 3 5 7 9
```

Example 4-2: $a0 = 8, $a1 = &array[0] (see Q4_SortArray.asm for details)

```
~/assn1$ spim -file ./Q4_SortArray.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to sort: 8
Enter a 32-bit integer (in decimal): 93
Enter a 32-bit integer (in decimal): 14
Enter a 32-bit integer (in decimal): 23
Enter a 32-bit integer (in decimal): 53
Enter a 32-bit integer (in decimal): 98
Enter a 32-bit integer (in decimal): 10
Enter a 32-bit integer (in decimal): 9
Enter a 32-bit integer (in decimal): 6
93 14 23 53 98 10 9 6
6 9 10 14 23 53 93 98
```

Example 4-3: $a0 = 1, $a1 = &array[0] (see Q4_SortArray.asm for details)

```
~/assn1$ spim -file ./Q4_SortArray.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
```

```
Enter the number of integers to sort: 1
Enter a 32-bit integer (in decimal): 1234567890
1234567890
1234567890
```

Q5: Inserting an Integer into a Sorted Array

Implement a function `sorted_array_insert` which inserts a 32-bit signed integer into an already-sorted array of 32-bit signed integers stored in memory. The function takes as input a new integer through `$a0`, the current number of integers in the array through `$a1`, and the memory address to the first integer in the array through `$a2`. You may assume that the number of integers to insert into the array is always a non-negative integer, and no out-of-memory error occurs when inserting a new integer into the array.

Example 5-1: Inserting 4 integers into the array

```
~/assn1$ spim -file ./Q5_SortedArrayInsert.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to insert into the array: 4
Enter a 32-bit integer to insert into the array (in decimal): 4
4
Enter a 32-bit integer to insert into the array (in decimal): 3
3 4
Enter a 32-bit integer to insert into the array (in decimal): 2
2 3 4
Enter a 32-bit integer to insert into the array (in decimal): 1
1 2 3 4
```

Example 5-2: Inserting 4 integers into the array

```
~/assn1$ spim -file ./Q5_SortedArrayInsert.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to insert into the array: 4
Enter a 32-bit integer to insert into the array (in decimal): 1
1
Enter a 32-bit integer to insert into the array (in decimal): -2
-2 1
Enter a 32-bit integer to insert into the array (in decimal): 3
-2 1 3
Enter a 32-bit integer to insert into the array (in decimal): -4
-4 -2 1 3
```

Example 5-3: Inserting 8 integers into the array

```
~/assn1$ spim -file ./Q5_SortedArrayInsert.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
```

```
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to insert into the array: 8
Enter a 32-bit integer to insert into the array (in decimal): 123
123
Enter a 32-bit integer to insert into the array (in decimal): -53
-53 123
Enter a 32-bit integer to insert into the array (in decimal): 52
-53 52 123
Enter a 32-bit integer to insert into the array (in decimal): 34
-53 34 52 123
Enter a 32-bit integer to insert into the array (in decimal): -123
-123 -53 34 52 123
Enter a 32-bit integer to insert into the array (in decimal): 98
-123 -53 34 52 98 123
Enter a 32-bit integer to insert into the array (in decimal): 0
-123 -53 0 34 52 98 123
Enter a 32-bit integer to insert into the array (in decimal): 75
-123 -53 0 34 52 75 98 123
```

Q6: Binary Heap

Implement two functions, `min_heap_push` and `min_heap_pop`, which implement the push and pop operations on a binary min heap consisting of 32-bit signed integers. The elements of the binary min heap are stored sequentially in memory: the root element, followed by the left and right childs of the root element, and then the left and right childs of the root element's left child, etc. Accordingly, assuming that the starting memory address of the binary min heap is `addr0`, the `i`-th element is stored at `addr0+(4*i)`.

Your implementation of the two binary min heap functions should comply with the following requirements:

- The `min_heap_push` function takes as input three arguments: the current number of elements in a binary min heap through `$a0`, the starting memory address of the binary min heap through `$a1`, and a new 32-bit signed integer to push to the binary min heap via `$a2`. Your implementation of the `min_heap_push` function should first store the new signed integer as the last element of the binary min heap, and then repeat replacing the new signed integer with its parent until the min-heap property gets restored. You may assume that no out-of-memory occurs during an invocation of the `min_heap_push` function (i.e., there is always enough space in memory when adding a new signed integer to the binary min heap).

- The `min_heap_pop` function takes as input two arguments: the current number of elements in a binary min heap through `$a0`, and the starting memory address of the binary min heap through `$a1`. Given the input arguments, the `min_heap_pop` function pops/deletes the root from the binary min heap, moves the last element of the binary min heap as the new root, and then repeats replacing the new root with its childs. When swapping an element with its childs (i.e., when the element is greater than one or two of the childs), the element should be swapped with the smaller child (e.g., when the element is 15 and its childs are 5 and 10, 15 should be replaced with 5, not with 10). When the two childs' values are the same, replace the parent with the right child. After restoring the min-heap property, the `min_heap_pop` function should return the popped/deleted 32-bit signed integer (i.e., the value of the popped/deleted root) through `$v0`.

Example 6-1: Pushing/popping 4 integers to/from the binary min heap

```
~/assn1$ spim -file ./Q6_BinaryHeap.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to push into the min heap: 4
Enter a 32-bit integer to push into the min heap (in decimal): 1
1
Enter a 32-bit integer to push into the min heap (in decimal): 4
1 4
Enter a 32-bit integer to push into the min heap (in decimal): 2
1 4 2
Enter a 32-bit integer to push into the min heap (in decimal): 3
1 3 2 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 1
2 3 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 2
3 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 3
4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 4
```

Example 6-2: Pushing/popping 6 integers to/from the binary min heap

```
~/assn1$ spim -file ./Q6_BinaryHeap.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to push into the min heap: 6
Enter a 32-bit integer to push into the min heap (in decimal): 6
6
Enter a 32-bit integer to push into the min heap (in decimal): 5
5 6
Enter a 32-bit integer to push into the min heap (in decimal): 4
4 6 5
Enter a 32-bit integer to push into the min heap (in decimal): 3
3 4 5 6
Enter a 32-bit integer to push into the min heap (in decimal): 2
2 3 5 6 4
Enter a 32-bit integer to push into the min heap (in decimal): 1
1 3 2 6 4 5
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 1
```

```
2 3 5 6 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 2
3 4 5 6
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 3
4 6 5
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 4
5 6
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 5
6
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 6
```

Example 6-3: Pushing/popping 8 integers to/from the binary min heap

```
~/assn1$ spim -file ./Q6_BinaryHeap.asm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Enter the number of integers to push into the min heap: 8
Enter a 32-bit integer to push into the min heap (in decimal): 4
4
Enter a 32-bit integer to push into the min heap (in decimal): 1
1 4
Enter a 32-bit integer to push into the min heap (in decimal): 3
1 4 3
Enter a 32-bit integer to push into the min heap (in decimal): 2
1 2 3 4
Enter a 32-bit integer to push into the min heap (in decimal): 2
1 2 3 4 2
Enter a 32-bit integer to push into the min heap (in decimal): 3
1 2 3 4 2 3
Enter a 32-bit integer to push into the min heap (in decimal): 1
1 2 1 4 2 3 3
Enter a 32-bit integer to push into the min heap (in decimal): 4
1 2 1 4 2 3 3 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 1
1 2 3 4 2 3 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 1
2 2 3 4 4 3
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 2
```

```
2 3 3 4 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 2
3 3 4 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 3
3 4 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 3
4 4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 4
4
INFO: Popping a 32-bit integer from the min heap:
INFO: min_heap_pop returned:
INFO:   $v0 = 4
```

Submitting Your Code

You should upload **only the six `.asm` files** to the LearnUs system. Please do not compress the .asm files before uploading them to the LearnUs system.

The deadline for uploading the .asm files to the LearnUs system is **11:59 pm on April 15, 2025 (Tue)**. If you need more time, you may upload the .asm files to the LearnUs system by **11:59 pm on April 17, 2025 (Thu)**; however, uploading the file after April 15, 2025 will result in **a 50% reduction in your score**. For example, if your file gets 80 points, your score will be (1) 80 points if you uploaded your file by April 15, 2025, or (2) 40 points if you uploaded your file by April 17, 2025.

Please be careful that if your file results in a compilation error, you will receive no score (i.e., 0 points). Therefore, please double check that your file successfully gets compiled and run before submitting it.

<<<<<<<<<<<<<<<<<<<<<<<<< END OF ASSIGNMENT #1 >>>>>>>>>>>>>>>>>>>>>>>>>