**CS39001-COMPUTER ORGANISATION LABORATORY**

**IIT KHARAGPUR**

**GROUP 53**

**21CS30017** – DEEPRAJ DAS

**21CS30026** – JAYKUMAR D PRIYADARSHI

**Bubble Sort Processor Code**

* The set of integers is being loaded into the stack:
* The **ldsp 1000(R0):** The memory address 1000 is loaded into the stack pointer (SP) by this instruction. This establishes the stack's starting location.
* **ld R1, 4(R0):** This command loads the value into register R1 at memory location 4.
* **push R1:** This command places the value of R1 at the current location (1000 in this example) on the stack. The value is efficiently stored in the stack.
* Every number is handled by repeating the first two instructions, increasing the memory location by 4.

**Setting up loop variables:**

* **move R1, R0:** This command duplicates R0's value, which is taken to be 0, into register R1.
* **move R2, R0:** This instruction copies the value of R0 into register R2. It also initializes the loop variable i. It sets up the loop variable called j.

**(out\_loop) Outer Loop:**

* **br out\_loop:** The outer loop is initiated when this instruction branches to the out\_loop label.
* **subi R3, R1, 10:** This instruction puts the result in register R3 after subtracting 10 from the value in register R1 (i). It determines whether i - n == 0 (where n is the set's integer count). The outer loop should stop and transition to the output section if the result is zero.
* **bz R3, output:** In the event that the value in register R3 is zero, this instruction branches to the output label.
* **move R2, R0:** This command duplicates R0's value in register R2. For the inner loop, it initializes the loop variable j.

**Inner Loop (in\_loop):**

* **subi R4, R2, 9:** This instruction subtracts 9 from the value in register R2 (j) and stores the result in register R4. It checks if j - (n-1) == 0. If the result is zero, it means the inner loop should exit and move back to the outer loop.
* **bz R4, out\_loop:** This instruction branches to the out\_loop label if the result in register R4 is zero.
* **srai R5, R2, 1:** This instruction right-shifts the value in register R2 by 1 bit (equivalent to dividing it by 2). It stores the result in register R5.
* **srai R5, R5, 1:** This instruction right-shifts the value in register R5 by 1 bit again (equivalent to dividing it by 2). This effectively divides the value in register R2 by 4, which is needed to calculate the correct offset in the stack.
* **add R6, SP, R5:** This instruction adds the value in register R5 to the stack pointer (SP) and stores the result in register R6. It calculates the address of the current element a[j] in the stack.
* **addi R7, R6, 4:** This instruction adds 4 to the value in register R6 and stores the result in register R7. It calculates the address of the next element a[j+1] in the stack.
* **ld R8, 0(R6):** This instruction loads the value at memory address 0 relative to register R6 into register R8. It loads the value of a[j] from the stack.
* **ld R9, 0(R7):** This instruction loads the value at memory address 0 relative to register R7 into register R9. It loads the value of a[j+1] from the stack.
* **sub R10, R9, R8:** This instruction subtracts the value in register R8 from the value in register R9 and stores the result in register R10. It calculates the difference between a[j+1] and a[j].
* **bmi R10, swap:** This instruction branches to the swap label if the result in register R10 is negative. It checks if a[j+1] < a[j] and performs a swap if true.
* **addi R2, R2, 1:** This instruction increments the value in register R2 (j) by 1. It effectively increments the inner loop counter.

**Swap (swap):**

* **st 0(R6), R9:** This instruction stores the value in register R9 (a[j+1]) at memory address 0 relative to register R6. It updates a[j] with a[j+1].
* **st 0(R7), R8:** This instruction stores the value in register R8 (a[j]) at memory address 0 relative to register R7. It updates a[j+1] with the original value of a[j].
* **addi R2, R2, 1:** This instruction increments the value in register R2 (j) by 1. It effectively increments the inner loop counter.

**Output (output):**

* **halt:** This instruction halts the execution of the processor.

**GCD MIPS Code Explanation**

The given MIPS code is a program to calculate the greatest common divisor (GCD) of two numbers using Euclid's algorithm. Let's go through the code step by step to understand its functionality.

**Initializing the Numbers**

addi R1, R0, 10 # load the first immediate number a

addi R2, R0, 25 # load the second immediate number b

In this part, the program initializes two numbers a and b with the values 10 and 25, respectively. These values can be changed to calculate the GCD of any two numbers.

**Comparing the Numbers and Swapping**

sub R3, R1, R2

bmi R3, swap

br loop

In this part, the program compares the values of a and b by subtracting b from a and storing the result in register R3. If the result is negative, it means that a is less than b, and the program branches to the swap section to swap the values of a and b. Otherwise, it continues to the loop section.

**Swapping a and b**

swap:

move R3, R2

move R2, R1

move R1, R3

br loop

In this part, the program swaps the values of a and b using a temporary register R3. The values of a and b are moved to R3 and then swapped using the move instruction. After the swap, the program branches back to the loop section to continue the calculation.

**Euclid's Algorithm Calculation**

loop:

sub R4, R1, R2

move R1, R2

move R2, R4

bz R2, output

sub R5, R1, R2

bmi R5, swap

br loop

* This is the main loop of the program where Euclid's algorithm is applied to calculate the GCD. The program subtracts b from a and stores the result in register R4. Then, it updates the values of a and b by moving R2 to R1 and R4 to R2.
* After that, the program checks if b (stored in R2) is zero using the bz instruction. If b is zero, it means that the GCD has been found and the program branches to the output section to halt the program and display the GCD.
* If b is not zero, the program checks if a is greater than b by subtracting b from a and storing the result in R5. If the result is negative, it means that a is less than b, and the program branches to the swap section to swap the values of a and b. Otherwise, it continues to the next iteration of the loop.

**Outputting the GCD**

**output:**

**halt**

In this part, the program halts after reaching the output section. This section is reached when b becomes zero, indicating that the GCD has been found. At this point, the value of a is the GCD. The program can be modified to display the GCD instead of halting if desired.