Lab 3

RAVI SHANKAR MEENA

7 february 2023

The code in Q1 part_a code is written in x86-64 assembly language, and written in C++.

- firstly , I used the objdump -C -D a.out
- -C Alias for –demangle and -D Alias for –disassemble-all
- then I looked for "0000000000001369<part_ a(int)>:" "0000000000001443 <main>:"
- 1379: 81 7d dc 87 13 00 00 cmpl $ 4999, -36(%rbp)
- the above line comparing the value of the local variable with 4999. If the value is less than or equal to 4999, it jumps to a certain location.
- Enter the key to unlock this: 5000 Your secret number is: 362477779

Q1

The code in Q1 part_b code is written in x86-64 assembly language, and written in C++.

- firstly , I used the objdump -C -D -M intel a.out
- -C Alias for –demangle and -D Alias for –disassemble-all and The -M intel option in objdump specifies that the disassembly output should use the Intel syntax for x86 assembly language.
- then I looked for "0000000000001369 <part_b(int, int, int)>:" & "0000000000001443 <main>:"
- '8b 45 dc' : "mov eax, DWORD PTR [rbp-0x24]" moves the value from the memory location [RBP - 0x24] into the EAX register.
- '0f af c0': "imul eax, eax" performs an integer multiplication of EAX with itself and stores the result in EAX.
- 89 c2: "mov edx, eax" moves the value in EAX into EDX.
- 8b 45 d8: "mov eax, DWORD PTR [rbp-0x28]" moves the value from the memory location [RBP - 0x28] into the EAX register.

Q1

- 0f af c0: "imul eax, eax" performs an integer multiplication of EAX with itself and stores the result in EAX.
- 01 c2: "add edx, eax" adds the value in EAX to the value in EDX and stores the result in EDX.
- 8b 45 d4: "mov eax, DWORD PTR [rbp-0x2c]" moves the value from the memory location [RBP - 0x2c] into the EAX register.
- 0f af c0: "imul eax, eax" performs an integer multiplication of EAX with itself and stores the result in EAX.
- 39 c2: "cmp edx, eax" compares the values in EDX and EAX and sets the flags accordingly.
- 0f 85 a2 00 00 00: "jne 143f <part_b(int, int, int)+0xd6>" is a conditional jump instruction. If the result of the previous comparison was not equal (Z)
- So, We got to observer that the three integers must be in format of a*a + b*b = c*c.
- Enter the keys to unlock this: 3 4 5 iseasy!! Your secret number is: 1235850486

Q1

The code in Q1 part_c code is written in x86-64 assembly language, and written in C++.

- firstly , I used the objdump -C -D -M intel a.out
- -C Alias for –demangle and -D Alias for –disassemble-all and The -M intel option in objdump specifies that the disassembly output should use the Intel syntax for x86 assembly language.
- 1429: 83 7d dc 06 cmp DWORD PTR [rbp-0x24],0x6 and 142f: 83 7d dc 0a cmp DWORD PTR [rbp-0x24],0xa: This compares the length of the string stored in [rbp-0x24] with the values 6 and 10.
- 143d: 75 18 jne 1457 ¡part_c(char*)+0x4e¿: If the length of the string is not 6 or 10, the code jumps to 1457 and outputs an error message.
- 1462: e8 49 fd ff ff call 11b0 ¡operator new[](unsigned long)@plt¿: This calls the operator new[] function with the length of the new string + 1 as its argument to allocate memory for the new string. The result of the call is stored in rax.
- Enter the keys to unlock this: aaaaaaa R3v3rseEngine3ring Your secret number is: 1377820563

Q1

CS230 { iseasy!!} R3v3rseEngine3ring

Q2

This code is MIPS assembly code and it implements the extended Euclidean algorithm to find the greatest common divisor (GCD) of two integers, A and M, as well as the coefficients x and y that satisfy the equation Ax + My = GCD(A,M).
The code starts with a data segment, indicated by the ".data" directive. However, there is no data defined in this segment, only a newline string. The program starts in the ".text" section with a main function that first reads two integers from the user using the "syscall" instruction and stores them in registers v0 and a2. The value in a2 is also stored in register a3. The program then initializes some variables and jumps to the "loop" label to begin the extended Euclidean algorithm. In this loop, the program performs several operations to compute the GCD and the coefficients x and y.
At the end of the loop, the program checks whether A or x have become negative, which would indicate that the algorithm has finished. If they have not become negative, the program jumps to the "funct" label to update the value of x.

Q3

In the .data section, three variables are defined:
"array" is an array of n integers with a size of n bytes. "size" is a word (4 bytes) variable initialized to zero, which will hold the size of the array. "newline" is a null-terminated string of a single character, representing the newline character. In the .text section, the code starts with a label called "merge" and a main function.
The main function does the following steps:

Q3

It reads an integer from the user using the syscall function with the code 5, and stores it in the "size" variable. It reads integer values from the user and stores them in the "array" using a loop. The loop initializes three registers: t0, t1, and t2. t0 is set to the base address of the array, t1 is set to the size of the array, and t2 is set to zero, which will act as a loop counter. Inside the loop, the program reads an integer from the user, stores it in the current element of the array pointed to by t0, increments t0 to point to the next element of the array, increments t2 by one, and jumps back to the start of the loop. Once the loop is finished, the "size" and "array" are passed as arguments to the mergesort function using jal (jump and link) instruction.

It prints the sorted array using another loop. The loop initializes three registers: t0, t1, and t2. t0 is set to the base address of the array, t1 is set to the size of the array, and t2 is set to zero, which will act as a loop counter. Inside the loop, the program loads the current element of the array pointed to by t0 to a0, prints it using syscall function with code 1, prints a newline character using syscall function with code 4, increments t0 to point to the next element of the array, increments t2 by one, and jumps back to the start of the loop. Once the loop is finished, the program exits using the syscall function with code 10.

Q3

The mergesort function starts with a label called "mergesort" and has two variables (a1 and a3) initialized to 1 and size-1, respectively, which will hold the current size of the array and the last index of the array. Then, a loop starts with the condition that a0 (left) is less than or equal to a3 (right). Inside this loop, there is another loop that iterates from a2 (leftStart) to size-2, initializing a2 and a4 (a4 is size-2). Inside this loop, the program sets the mid-point of the array (a5) to the value of (leftStart + currSize - 1) if it is less than the last index of the array; otherwise, it sets it to the last index of the array. Then, it checks if the value of (leftStart + 2currSize - 1) is less than the last index of the array; if it is, it sets t3 to (leftStart + 2currSize - 1), otherwise it sets it to the last index of the array. Finally, it calls the merge function, passing it the address of the array (array), the left index (left).

Q4

- I implimented brk syscall in program
- The code first declares the necessary functions and sets up the main function. It then reads input for the sizes of the first and second matrices and allocates memory for them using the syscall function. The code then reads in the values for the first and second matrices, initializes the third matrix, and multiplies the first two matrices. The result is stored in the third matrix. The code outputs the resulting matrix to standard output.
- Then I wrote the ijk,ikj,kij,jik,jki and kji part of all so that we can get the muatrix multiplication by all possible ways then we plotted all of them using matplotlib and some python libraries. is the plot