Semin Bae

Tony Mione

CSE320 / Fall 2024

Oct/10/2024

hw3_report:

This assignment includes three tasks which are hw3_bits.c, hw3_matmul.c, and hw3_struct_union_func.c. For each task, I will provide a summary of my coding efforts, challenges faced, and an analysis comparing the code with its assembly version. And Finally I will share what I learned and what I find really interesting at the end.

Task 1: hw3_bits.c

Summarizing my efforts to code the function that I have:

In this task, I have to work on setting, clearing, and toggling specific bits. Through this, I know a deeper understanding of bitwise operators like |, &, ^, and ~. I had previously experienced manipulating these operations only in the paper, but implementing them directly in C allowed me to see how they work at a lower level.

Challenges faced:

At first, using bitwise operators felt overwhelming because I didn't know how to change specific bits as I wanted. There are so many choices that I make ways to change bits. But as I practiced, I realized there were specific methods for each operation: using | for setting bits, & with ~ for clearing, and ^ for toggling bits.

Comparing with assembly code and analyze:

In the assembly code, the initial data is set with mov w8, #236, which matches the value 0b11101100 in binary. Setting a bit using | in C corresponds to the orr instruction like orr w8, w8, #0x8 in assembly. For clearing a bit, C's & and ~ translate to and w8, w8, #0xffffffdf in assembly, where the high hexadecimal number results from inverting bits. The toggle operation uses the eor instruction, shown as eor w8, w8, #0x80. The condition checking for the seventh bit in C with if and & appears as ands w8, w8, #0x80 in assembly, followed by branching and printing the result.

Task 2: hw3_matmul

Summarizing my efforts to code the function that I have:

This task is matrix multiplication using pointers and printing the result. I created a matrix_multiply(MatA, MatB, MatOut) function that multiplies two matrices MatA and MatB, and stores the result in MatOut. I used pointer arithmetic to access and store values, which helped me improve my understanding of pointers.

Challenges faced:

The biggest challenge was working with 9x8 matrices, not just the pointer calculations but also the complexity of implementing the multiplication. To simplify the process, I started with smaller 3x2 and 2x3 matrices, working through the multiplication by hand to understand it better. After succeeding with smaller matrices, I applied the same logic to the larger matrices and tested the results successfully.

Comparing with assembly code and analyze:

Comparing the C code with its assembly version reveals how high-level loops are translated into lower-level instructions. In C, nested loops for matrix multiplication correspond to specific labeled sections in assembly, such as LBB0_1 for the i loop, LBB0_3 for the j loop, and LBB0_5 for the k loop. Initializing values to zero in C is shown as str wzr in assembly. Additionally, conditionals within loops are handled by subs and cset instructions in assembly, helping me understand how control flow is managed at a lower level.

Task 3: hw3_struct_union_func.c

Summarizing my efforts to code the function that I have:

This task with structures and unions to work different data types. Initially, it was challenging to know what is the difference between structures and unions, but this assignment helped me understand their differences, benefits, and use cases. I corrected code errors and experimented by storing data in structures and unions, printing them to see how each worked.

Challenges faced:

The most difficult part was understanding the working of unions. At first, I didn't realize that a union shares memory for all its variables, so when I tried to assign values to multiple variables at once, I saw garbage data. This experience helped me understand the memory-saving aspect of unions, which will be useful for writing more efficient code in the future.

Comparing with assembly code and analyze:

This time I printed the values in union and structures, so while I saw the assembly code, I knew how assembly works with print strings. l_.str.# is the where the string place saved, and those will use with in "str   x1, [sp]". So I could know where the structure and union are implemented in assembly code.

Conclusion:

From the three tasks, I have some understanding of low-level programming concepts and how C translates to the assembly language. The bitwise operations, matrix multiplication, and structures/unions provided practical insights into data manipulation and memory management. This created a new curiosity about compiler design, specifically how C code is translated into assembly and further into machine code.