Brett Chow, Aleksandar Aleksandrov

ECEC 355

Project Three

Question 1:

X22 → i ; x21 → j ; x23 → 4 ; x25 → base addr out[i] ; x9 → data out[] ; x7 → shift output ;

x26 → base addr mat[]

```
    1.  addi x22, x0, 0          //i = 0
    2.  addi x23, x0, 4          // x23 = 4
Loop 1:
    3.  bge x22, x23, 80         // i >= 4 → Quits entire program and goes to step after 22
    4.  slli x10, x22, 3         // x10 =  i *2^3
    5.  add x10, x10, x25        // x10 = out[i] address
    6.  addi x9, x0, 0           // x9 = 0 → stores out[i] data
    7.  addi x21, x0, 0          // j = 0
Loop 2:
    8.  bge x21, x23, 28         // j >=4 → Quits and goes to step 15
    9.  slli x11, x22, 2         // x11 = i*2^2
    10. add x11, x11, x21        // x11 = (i*2^2) + j
    11. jal x1, 28               // go to shift (step 18)
    12. add x9, x9, x7           // out[i] += x7
    13. addi x21, x21, 1         // j++
    14. beq x0, x0, -24          // go back to step 8 (start of Loop2)


    15. sd x9, 0(x10)            // out[i] = x9 (result from Loop2)
    16. addi x22, x22, 1         // i++
    17. beq x0, x0, -56          // go back to step 3 (start of Loop1)


Shift:
    18. slli x8, x11, 3          // (i*4+j) * 2^3
    19. add x8, x8, x26          // x8 = mat[i*4+j] address
    20. ld x7, 0(x8)             //x7 = mat[i*4+j] data
    21. sll x7, x7, x22          // mat[i*4+j] << i
    22. jalr x0, 0(x1)           // go back to step 12 (middle of loop 2)
```

Question 2:

In Parser.{h,c}, we made changes where we are able to support new commands that we haven't used in previous projects. Some of the commands that we added are bge, beq, sll, sd, jal, and jalr. Within our Parser code, we had to coordinate which command is which type of binary instruction, such as R type, S type, etc. The new types we implemented were S type and UJ type. We implemented similar if statements to each type where we would shift left a certain number of times to capture the commands features, such as rd, opcode, and funct3. With each command, we also had to set its opcode, funct3, and funct7 values in our setInstructionInf function. This prompted us to also set functions that allowed us to convert S and UJ type instructions into binary instructions.

Question 3:

In Core.{h,c}, we made changes in order to calculate the out matrix through our C program data path that we created in the previous week. First, our task assigns an array of mat[16] that have elements of {0, 1, 2, 3, …, 15}. Within our code, we had core->data_mem[i] items equal to elements in the mat array. Since the array is 64 bits, we assigned each data_mem item with an offset of 8. The other changes we made were adding S and UJ functions in locations in our Core program that needed them, such as in our ImmeGen function and identifytype function. Furthermore, we had to emphasize on writing the commands for jal and jalr. Since these commands jump to new instructions based on the program counter, we had to make if statements for jal and jalr that would allow the program to jump to new instructions based on the program counter increasing or decreasing with an offset of 4. Overall, our Core program needed to complement commands that we were missing before in order to run RISC-V instructions that would output values of the matrix out.

Question 4:

The layout of data memory after completion of the simulation has each of the elements of mat[16] = {0, 1, 2, 3, …, 15} equal to a specific location in data_mem. Each of these locations have an offset of 8. This means that core->data_mem[0] would be equal to the first element in mat which is 0. Then, core->data_mem[8] would be equal to the second element in mat which is 1. This would go on until every mat element is in a data_mem location. Since each element in mat requires 64 bits, the last element of mat would be in data_mem[120].

Summarization of Experiment:

Based on our project, we had to translate the given C program into RISC-V instructions. This program required instructions to process two loops, in which values would be calculated based on the elements of mat and shifting its value to get a higher value. These values would add onto out[i] elements and we needed to figure out what the values of out[0, 1, 2, and 3] are when going through the two loops 4 times. In our parser and core programs, we needed to implement new command instructions, such as bge, jal, jalr, and sd to figure out the values of the array out. Through our RISC-V instructions, we specified the number for each step and detailed which instructions correlated to which part of the given C program. All this information helped us code Core and Parser to support the simulation of the translated RISC-V program.