Cameron Calv and Nicholas Sica
ECEC 355 Computer Architecture
Project One Report – Single-cycle RISC-V Simulation

Completing the Core Structure

The Core was completing in a matter of segments. Each segment took care of another responsibility of the Core. These segments are as follows, initializing the core, the core responding to a clock tick, the ALU, the control signals for the ALU, the control signals for the rest of the system and the building of immediate values.

To initialize the core, we allocate memory for all aspects of the core including space for the core itself, space for the instruction memory, and space for the registers. In case registers need to be loaded with specific values before execution, this is also where those values would be assigned. We also specify which function the core will use to keep track of time and clock signals. This function is the 'tickFunc'.

The tick function works by first reading the next instruction as determined by the program counter. Then using the instruction code, we set all of the control signals necessary for the rest of the system and the ALU to function accordingly. Based on the control signal we see whether or not to use a register's data for an operation or an immediate value. After the operation ensues, we initiate writing back to the register memory. Once again, the control signals dictate where the data is being stored. Finally, the program counter is set to the proper value so that upon the next tick, the correct instruction is pulled from memory to be executed. The tick function returns true if there are any remaining instructions left to execute.

The arithmetic logic unit (ALU) takes in two operands and a collection of control signals to determine which operation to perform on the operands. Within the ALU is a switch statement allowing for the result of the operation to be various logical operations, bit shifting, exponentiation, basic arithmetic, and comparisons. For the proper control signals to be generated for the ALU, the 'alucontrol' function takes in the specific portions of the instruction like funct7 and funct3 and sets the control signals for the ALU depending on the desired operation.

Of course, there is also a function for setting the general control signals. It uses the opcode of the instruction and sets the control signals appropriately. The control signals that are present and assigned by the various opcodes are as follows: regWrite, aluSrc, memWrite, aluOp, memToReg, memRead, beq, jal, jalr, sometimes bne, blt and bge.

Finally, the 'buildImm' function merely takes the instruction and translates from it the immediate value that is requested. The immediate built depends on the format of the code passed to the function.

After the simulator was complete, we ran the 'example_cpu_trace' set of instructions and the final values of the requested registers are as follows.
Register x9: 128
Register x11: 16

Testing the Matrix Operation Program

We tested the matrix operation program by first creating a file 'matrix' with a set of line delimited machine code instructions that represent the matrix multiplication and shift operations outlined in the project sheet. Various labels in the document outline the boundaries of where the 'matrix_opr' function and 'shift' functions are and branch and jump instructions facilitate navigating to those functions when necessary. Instructions in 'matrix' are passed to the core one after the other and the CPU takes care of everything. The initial values for the matrices are set within the initCore portion of the Core.c. To view the final states of the matrix, we print out the register values using Core.c.

Final Register Values of Matrix Operation Test

Data Address 0: 0	Data Address 312: 0	Data Address 624: 0	Data Address 936: 0
Data Address 8: 1	Data Address 320: 0	Data Address 632: 0	Data Address 944: 0
Data Address 16: 2	Data Address 328: 0	Data Address 640: 0	Data Address 952: 0
Data Address 24: 3	Data Address 336: 0	Data Address 648: 0	Data Address 960: 0
Data Address 32: 4	Data Address 344: 0	Data Address 656: 0	Data Address 968: 0
Data Address 40: 5	Data Address 352: 0	Data Address 664: 0	Data Address 976: 0
Data Address 48: 6	Data Address 360: 0	Data Address 672: 0	Data Address 984: 0
Data Address 56: 7	Data Address 368: 0	Data Address 680: 0	Data Address 992: 0
Data Address 64: 8	Data Address 376: 0	Data Address 688: 0	Data Address 1000: 136
Data Address 72: 9	Data Address 384: 0	Data Address 696: 0	Data Address 1008: 128
Data Address 80: 10	Data Address 392: 0	Data Address 704: 0	Data Address 1016: 0
Data Address 88: 11	Data Address 400: 0	Data Address 712: 0	
Data Address 96: 12	Data Address 408: 0	Data Address 720: 0	
Data Address 104: 13	Data Address 416: 0	Data Address 728: 0	
Data Address 112: 14	Data Address 424: 0	Data Address 736: 0	
Data Address 120: 15	Data Address 432: 0	Data Address 744: 0	
Data Address 128: **34**	Data Address 440: 0	Data Address 752: 0	
Data Address 136: **94**	Data Address 448: 0	Data Address 760: 0	
Data Address 144: **154**	Data Address 456: 0	Data Address 768: 0	
Data Address 152: **214**	Data Address 464: 0	Data Address 776: 0	
Data Address 160: 0	Data Address 472: 0	Data Address 784: 0	
Data Address 168: 0	Data Address 480: 0	Data Address 792: 0	
Data Address 176: 0	Data Address 488: 0	Data Address 800: 0	
Data Address 184: 0	Data Address 496: 0	Data Address 808: 0	
Data Address 192: 0	Data Address 504: 0	Data Address 816: 0	
Data Address 200: 0	Data Address 512: 0	Data Address 824: 0	
Data Address 208: 0	Data Address 520: 0	Data Address 832: 0	
Data Address 216: 0	Data Address 528: 0	Data Address 840: 0	
Data Address 224: 0	Data Address 536: 0	Data Address 848: 0	
Data Address 232: 0	Data Address 544: 0	Data Address 856: 0	
Data Address 240: 0	Data Address 552: 0	Data Address 864: 0	
Data Address 248: 0	Data Address 560: 0	Data Address 872: 0	
Data Address 256: 0	Data Address 568: 0	Data Address 880: 0	
Data Address 264: 0	Data Address 576: 0	Data Address 888: 0	
Data Address 272: 0	Data Address 584: 0	Data Address 896: 0	
Data Address 280: 0	Data Address 592: 0	Data Address 904: 0	
Data Address 288: 0	Data Address 600: 0	Data Address 912: 0	
Data Address 296: 0	Data Address 608: 0	Data Address 920: 0	
Data Address 304: 0	Data Address 616: 0	Data Address 928: 0	