Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

Project 1: Additional AVR Instruction Implementation

Objectives:

The objective of the project was to addon additional instructions to an already existing AVR processor that lacked all AVR instructions. The instructions that were to be created and added were multiply unsigned (MUL), multiply signed (MULS), multiply signed with unsigned (MULSU), and twos complement (NEG).

Preliminary Analysis:

The instruction set that was added had the following machine codes and outputs:

Instruction	Machine Code	Output Register(s)	d and r address range
MUL	1001-11rd-dddd-rrrr	R1:R0	0-31
MULS	0000-0010-dddd-rrrr	R1:R0	16-31
MULSU	0000-0011-0ddd-0rrr	R1:R0	16-23
NEG	1001-010d-dddd-0001	Rd	0-31

Table 1:New AVR Instruction Set Codes

All MUL operations multiply two 8-bit numbers. Each instruction will vary between signed and unsigned inputs. MULS will multiply two signed inputs, MULSU will multiply a signed (d-address) and an unsigned (r-address) input and MUL will multiply two unsigned inputs. These operations will result in a 16-bit output with varying ranges depending on the operation.

Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

Augmented Control Unit:

Starting on the top level, the first thing of note is the augmented control unit. Additional selection criteria of the load select was needed for additional changes which will be discussed later. Figure 1 is the Quartus design of the changed load select. As can been seen in the lower part of the figure, additional logic has been added for the multiply circuits and the NEG command.

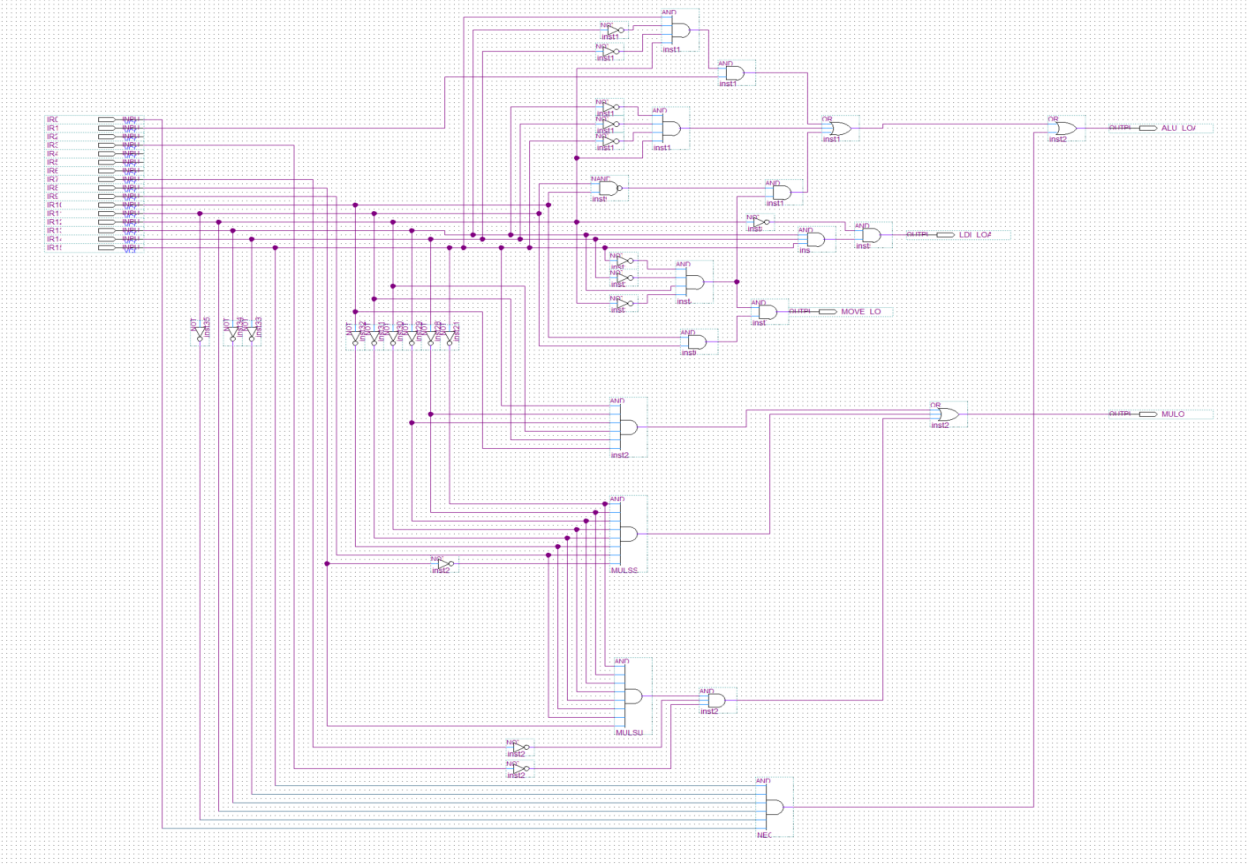

Figure 1:Augmented Load Select

MUL Operations Logic:

The next thing of note is the additional block handling the multiplication operations. Seen in figure 2, the function blocks on the left side are the multiply circuits that were created. These were created using concepts learned in Digital Logic, an extremely invaluable class that introduced many of the concepts used in this project. The large block on the right side is a MUX that selects from 4 16-bit numbers, the select switches are set by the instruction codes sent to the WIMPAVR. The zero flag and carry flag are calculated in each block and then fed into a MUX to select the correct function's output. This additional block was added because the existing ALU

only has an 8-bit output. Adding additional outputs to the existing ALU would require several changes in the logic, and this seemed to be the more straightforward path.

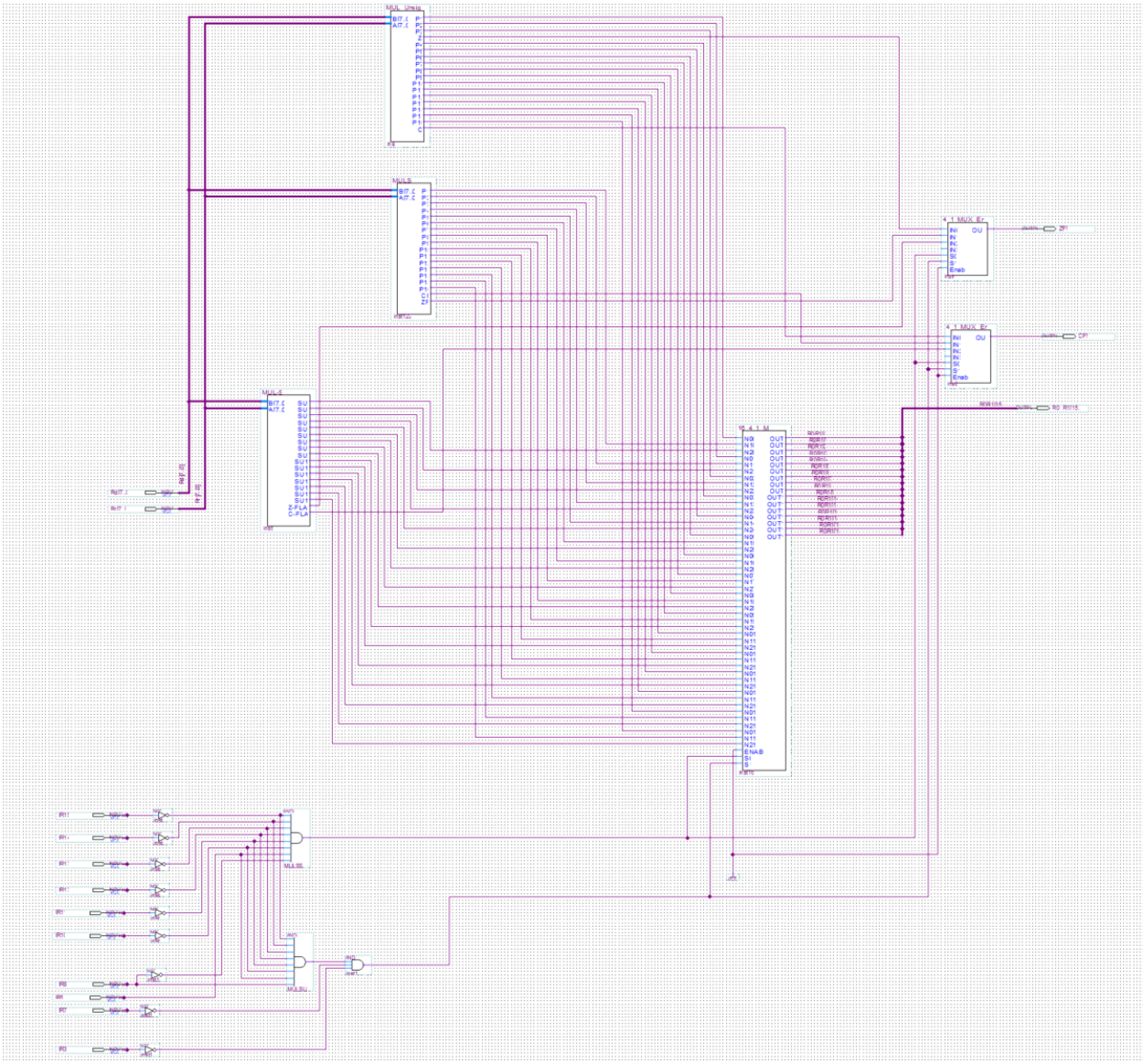

Figure 2:Multiplication Arithmetic Logic Unit

The output of the zero flag and the carry flag are handled on the top level by using 2 MUXs, figure 3 shows this. The select switches are set by the control unit's MUL LOAD output. An LED was chosen to verify that the multiplication action is taking place and allows for some debugging, this is shown in figure three as well.

Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

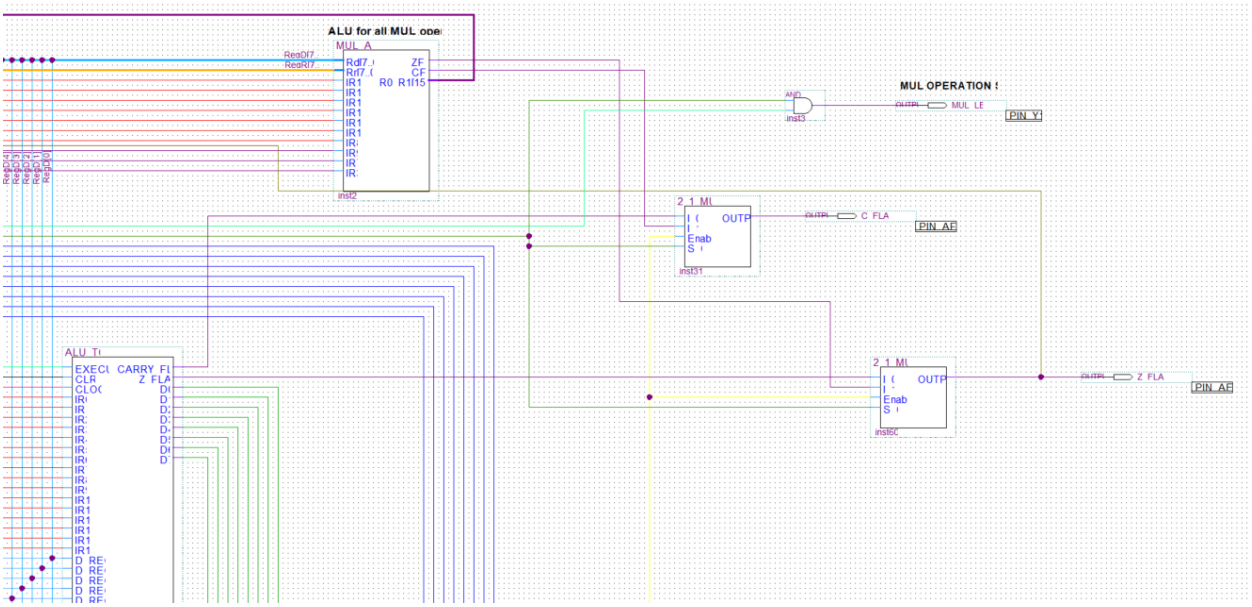

Figure 3: Zero Flag and Carry Flag Logic

Figure 4 shows the changes made to the ALU to accommodate the NEG command. This does a two's complement of the 8-bit input, and stores the result in register d. The logic for all outputs is handled by several MUXs of differing sizes and is integrated with the existing ALU logic. This was relatively straightforward since the output ports of the 8-bit adder and two's compliment were so similar. The NEG instruction code activates the MATH_ENABLE and is also used to set the 8_2_1 MUX switch high.

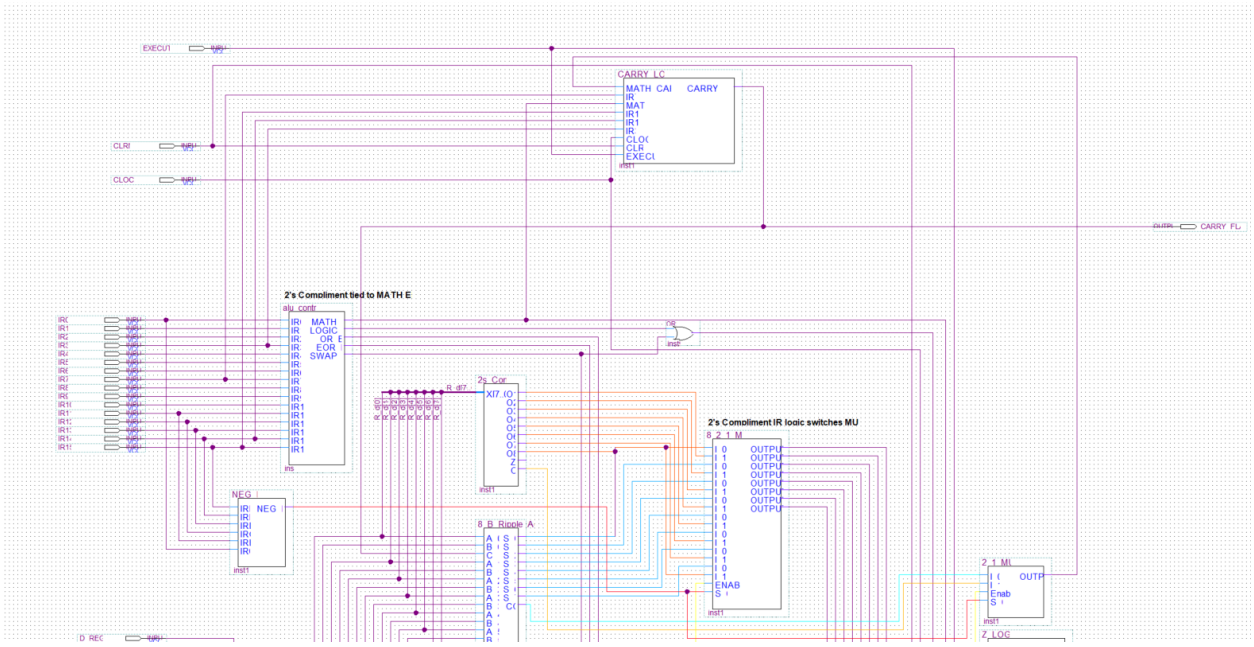

Figure 4:ALU Logic Changes

Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

Register Modifications:

The two MUXs entitled MUL_IN are each half of the 16-bit number with the top 8-bits being the set sent to R1 and the lower 8 bits sent to R0. When a MUL operation is detected, the MUX is switched to route the 16-bit output into the registers. The same is done with the write enable input of the registers. This section was the hardest to understand but was rewarding once it was completed. When a MUL operation is not detected, the registers function as normal.

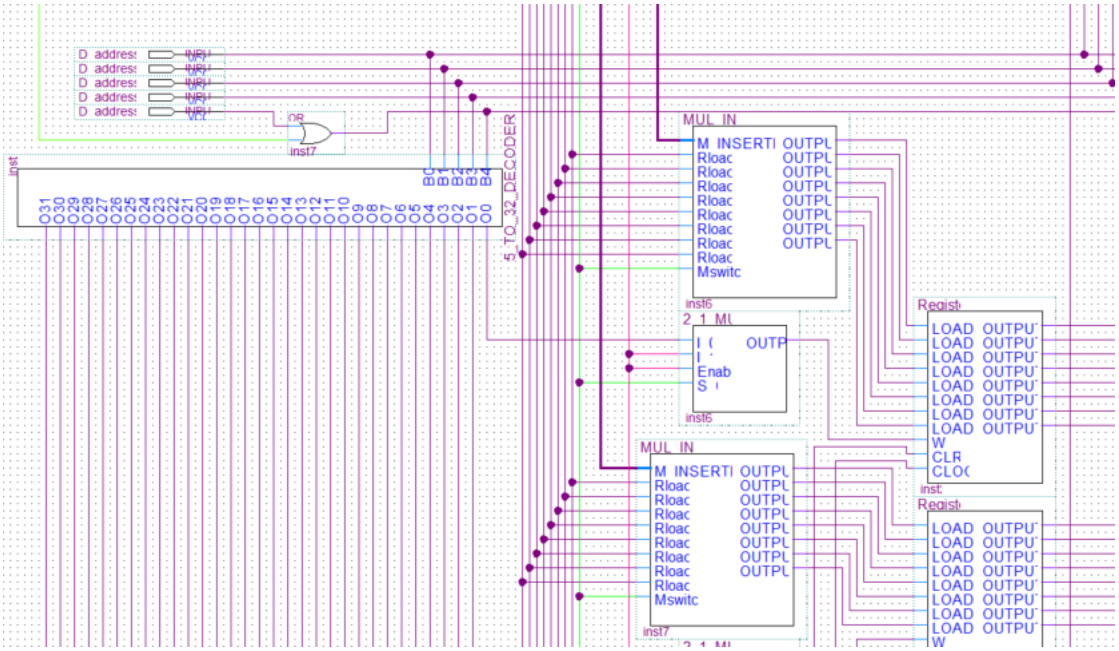

Figure 5: Register Block Additions and Changes

Instruction Set:

Included below is the instruction set that was designed to test our MUL, and NEG operations. It was made sure that every existing command was included to show that everything was fully operational after the modifications to the WIMPAVR. This made it a bit long but rewarding. First the instructions were loaded into the SRAM of the FPGA using the programmer Quartus project. Once it was uploaded, our modified WIMPAVR could carry out the given instructions. For troubleshooting the pushbutton was used as the clock edge input. But seeing it operate with the internal clock signal was very satisfying, and even though the clock was at a relatively low frequency it puts into perspective how amazing the computers we use everyday really are.

Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

Table 2: Instruction Set Machine Code

ADRESS	INSTRUCTION	MACHINE CODE
00H	LDI R31 (0A)H	E0FA
01H	LDI R30 (10)H	E1E0
02H	SEC	9408
03H	ADC R30, R31	1FEF
04H	MOV R29, R30	2FDE
05H	EOR R29, R30	27DE
06H	MUL R29, R30	9FDE
07H	LDI R31 (FF)H	EFFF
08H	OR R29, R31	2BDF
09H	ADC R31, R29	1FFD
0AH	CLC	9488
0BH	LDI R22 (60)H	E660
0CH	SWAP R30	95E2
0DH	LDI R30 (FF)H	EFEF
0EH	MUL R29, R30	9FDE
0FH	LDI R31, (00)H	E0F0
10H	AND R30, R31	23EF
11H	LDI R20 (80)H	E840
12H	LDI R21 (FF)H	EF5F
13H	MULSU R20, R21	0345
14H	MULS R20, R22	0246
15H	MOV R20, R30	2F4E
16H	MULS R20, R22	0246
17H	MULSU R20, R21	0345
18H	LDI R31 (12)H	E1F2
19H	NEG R31	95F1
1AH	LDI R31 (05)H	E0F5
1BH	CLC	9488
1CH	ADC R31, R21	1FF5
1DH	BRB(Z), STOP	F009
1EH	RJUMP, HERE	CFFC
1FH	NEG R31	95F1
20H	RJUMP, END	CFFF

Preston Carroll & Jack Stone
Cpe 3150
Date: 4/5/2023

Main Challenges:

The most difficult part of the project was defining the registers of which to store the outputs of the multiplications, specifically the MULSU as it uses a more limited range of d and r registers 16-24, compared to the other functions that were added. MULSU was the only operation that seemed to have any issues. But through troubleshooting it was determined one of the registers being multiplied was always zero. This gave the impression that it was not multiplying the correct registers together. By forcing the 5^{th} bit of the D address high when any MUL operation was detected, this ensured the address range would start at 16. This fixed the issue with MULSU, though probably had the adverse effect of making the range of addresses of the MUL operation 16-31. Where in the AVR instruction datasheet, it is defined as addresses 0-31. But given the time spent on the project, it was determined that this was a minor inconvenience. Additionally, because the output of the MUL operations was a 16-bit output at the maximum, the data needed to be stored and read from 2 registers.

Conclusion:

This project, which seemed a little daunting, allowed for an understanding of the way in which these types of processors work. It allows for a glimpse as to what is possible and the way in which many of the world's computers function daily. Understanding machine code gave a lot of needed understanding of these microcontrollers and helped transition into Project 2. The project also allowed for an understanding of what was to be expected from the rest of the course, and the challenges that lie ahead. This is an introductory project that expands the problem-solving capabilities and allows to search for answers down avenues that are somewhat abstract and nonsensical. With a clear end goal in mind, and the ability to make one's own map, it can have endless paths to success.