EE 443

Computer Engineering Analysis and Design Laboratory

Lab #8: The Complete Datapath

4/25/2021

**Prepared by:**

Shoa Russell

Instructor/TA comments and grading

**1.** **Objective and Background**

The objective of this lab was to create the control block and the PC counter for the 16-bit microprocessor and then connect all the modules previously created together to form the complete design. This lab used block diagram files to connect the modules together, this allowed for visually connecting the blocks together rather than writing VHDL code. Each of the block diagram files can be used hierarchically, allowing for combining of the VHDL code files into larger blocks which could then be used under a top level block.

**2. Equipment**

* Altera DE2 Cyclone II EP2C35F672C6N
* Quartus 13.0 Web edition

**3. Procedure**

In this lab we were asked to complete the data path by building the program counter and the control module. The Control module was written in VHDL and was used to send the enabling bits to the of the modules in the 16-bit microprocessor. The operation signal and the setting for different types of instructions can be seen below in Table 1 and the block diagram for the VHDL implementation can be seen in Figure 1. The control module was created using a switch case statement that looks at the four MSB of the instruction, these bits represent the opcode and dictate what the control signal values will be. The ALU operation was also sent on the output, for R-type instructions this was set to 3 LSB of the instruction, for I-types it was set to “010” for add and for BEQ it was set to “110” for subtract.



Table :Control module control outputs

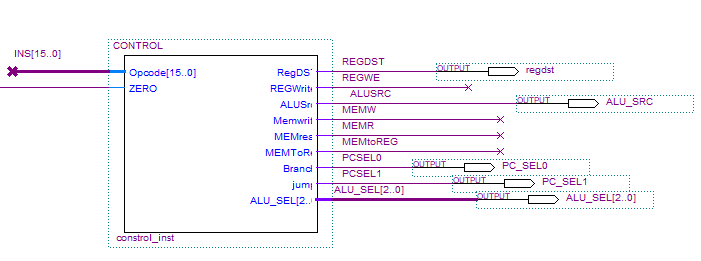


Figure : Control Module dbf

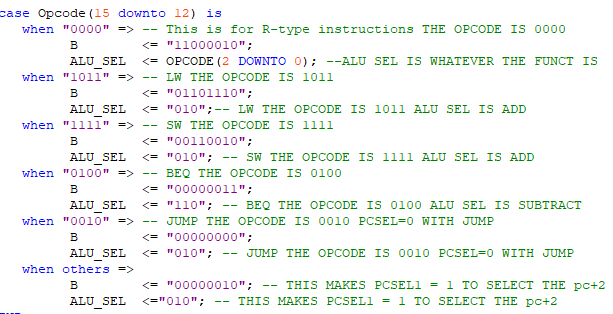


Figure : Case Statement for control signal

The program counter simply took the input value and at the rising edge of the clock sent that value out of the PC. The PC was then fed into the instruction memory to select which instruction would be executed next. The PC counter as well as several multiplexers and the instruction memory were combined to form the block diagram file named M16dp1. This can be seen in Figure 5 and Figure 8.

The 8X16 Register file was combined with the ALU to create a new bdf file called M16dp2, this block allowed us to perform ALU operation on the selected registers and then feed to result to a MUX external to the block. This mux was used to select between writing the data from memory or writing the ALU result back to the register. This can be seen in Figure 6 and Figure 7.

Within the REG8X16 we created in the previous lab, we were asked to hardwire two of the register REG0 and REG7 to 0x0000 and 0x0100. This will allow the registers to always return a constant, REG0 is always going to store a constant of 0. The value of R7 is the starting address for the data memory and will be used to index into the data memory accordingly. Hardwiring of these registers can be seen in Figure 2: Hardwiring REG0 and REG7.

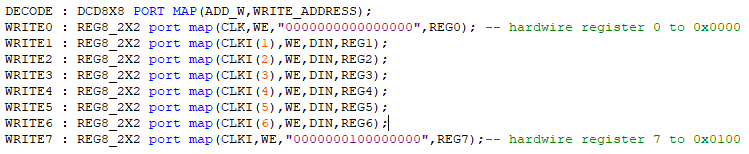


Figure 3: Hardwiring REG0 and REG7

Figure 4.24 from the Computer Organization and Design” was marked up to show how each of the files that we created in this lab are going to be used in the 16-bit microprocessor. The Sections highlighted in different colors show what was used for the two block diagrams that were created. Both of these block diagrams were included in the top-level block diagram M16dp.

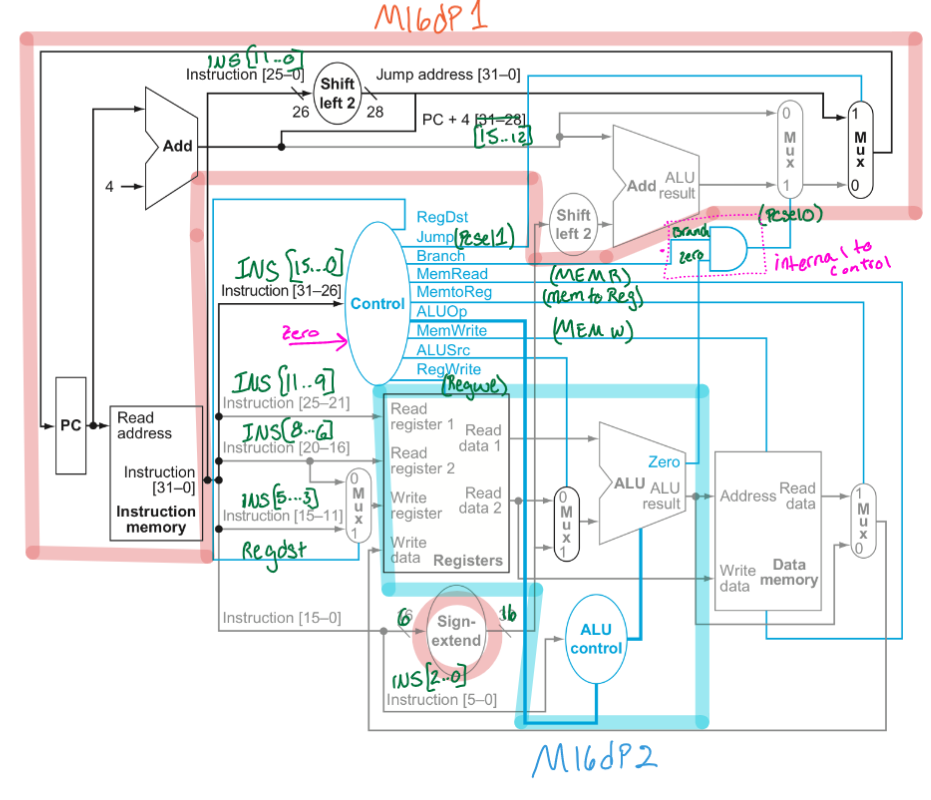


Figure 4: Marked up figure 4.24

The instruction memory was hardcoded with the operations seen in Table 2 below. Instruction 6 served as an out of bounds check when loading a memory location not within the memory range of 0x0100 and 0x011F, since Register 0 was hardcoded with 0x000 and the offset was only 4 it would not reach the bounds. Instruction 7 branched to the jump instruction which jumped to the beginning of the instructions. The data memory was also hardcoded with the two numbers seen in Figure 5.

|  |  |  |  |
| --- | --- | --- | --- |
| ins\_mem # | ACTION | INSTRUCTION | MACHINE CODE |
| 0 | R1 <= R7 + R0 | add $1,$0,$7 | 0000:000:111:001:010 |
| 1 | R2 <= R0+R7 | add $2,$0,$7 | 0000:000:111:010:010 |
| 2 | R6 <=M(R($7) +0) | lw $6, 0($7) | 1011:110:010:000000 |
| 3 | R3 <=M(R($7) +2) | lw $3, 2($7) | 1011:111:011:000010 |
| 4 | M(R($1) +4)=R1 | sw $1, 4($1) | 1111:001:001:000100 |
| 5 | R4 <= R3 + R3 | add $4,$3,$3 | 0000:011:011:100:010 |
| 6 | R1 <=M(R($0) +4) | lw $1, 4($0) | 1011:000:001:000100 |
| 7 | if $3=$3 pc=> offset | BEQ $3,$3, x0006 | 0100:011:011:000110 |
| 14 | Jump to ins(0) | Jump 0x0000 | 0010:000000000000 |

Table : memory instructions

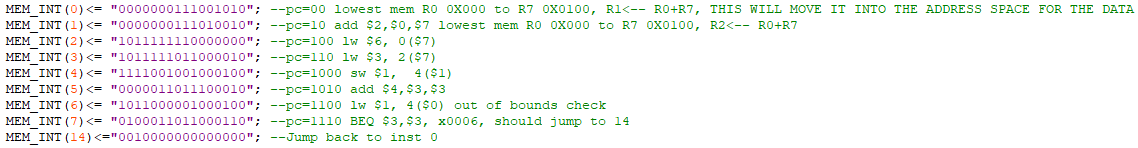


Figure : Instruction memory



Figure : Data memory storage

**4. Results**

The ALU operations were tested individually in the M16dp2 block, this helps to confirm that the registers are working along with the ALU. This block can be seen in Figure 10 and the annotated waveform can be seen in Figure 7. All the ALU operations were confirmed to be working as expected.

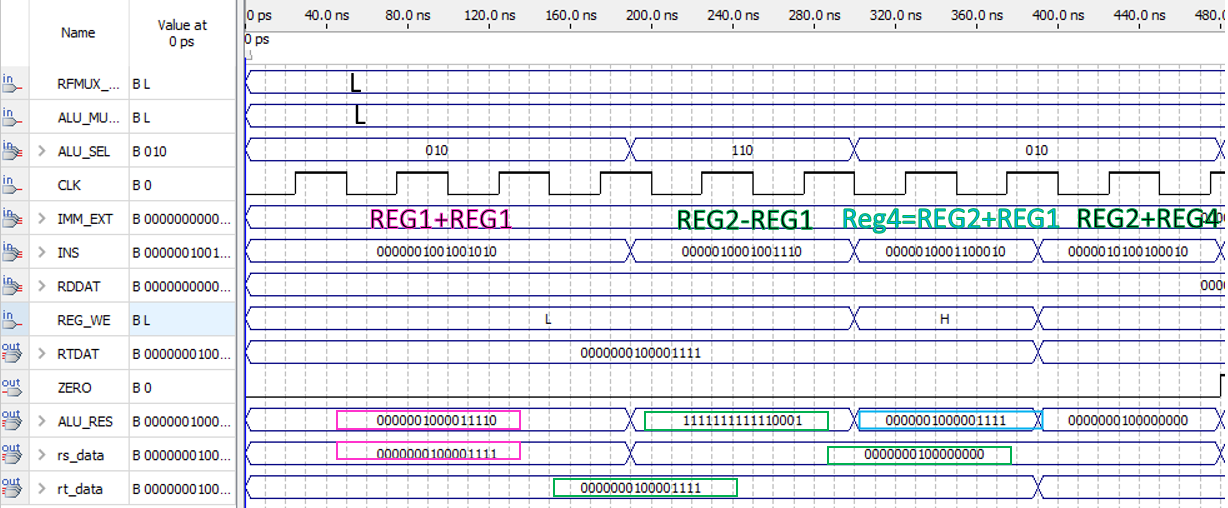


Figure :M16dp2

The branch was calculated using the formula below to determine the new value of the PC, note that the branch offset is only shifted to the left once since this is a 16-bit microprocessor instead of a 32-bit.This can be seen in the resulting waveform for M16dp in Figure 6 in pink. The branch jumps from instruction 7 to instruction 14, in instruction I have a jump command to jump back to the start of the instruction set, Instruction 0. The jump command can be seen in yellow in Figure 6.

BEQ $3,$3, x0006

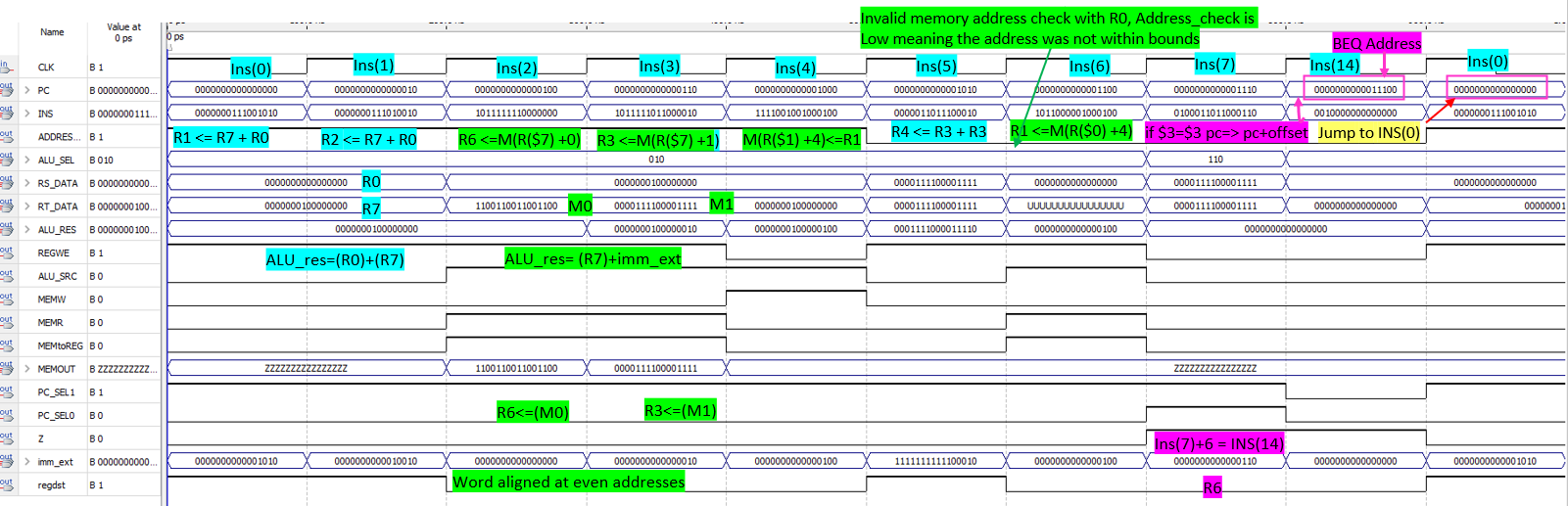


Figure : M16dp waveform

Overall the waveform for M16dp shows that writing and reading from data memory and the registers was possible. In several of the instructions I either read from the register and wrote to memory or read from memory and wrote to the register successfully. One issue that did occur was that the PC was not incrementing correctly, this was due to assigning the PCsel0 incorrectly. Another issue was that the Register 0 was set to a constant of 0 and thus was not able to be used to increment into the data memory as the address was out of bounds. To resolve this, I added the register & that was hardcoded with the first value of the data memory to the Reg0 and stored it in the other memory registers for use in accessing the data memory. Timing delays and the cell usage for the VHDL files in the project can be seen in Table 3.**6. Conclusion**

In conclusion this lab was able to be completed and validated that each of the operational codes given in the lab were functional. This lab was the final step in completing the 16-bit microprocessor and helps to give an understanding of how all the elements come together and work along side each other. Overall creating the block diagram files also helped to show exactly how the data path would be connecting one module to the others and was very illustrative of how the microprocessor would be used.

**7. Attachments**

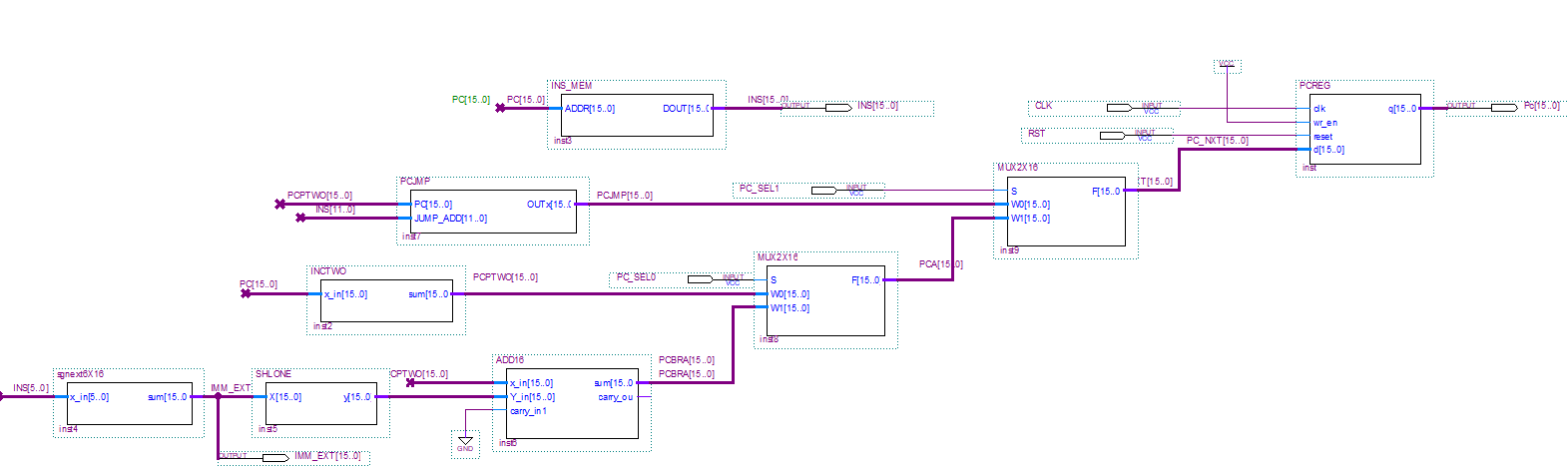


Figure : Incrementation of the PC counter and Instruction memory

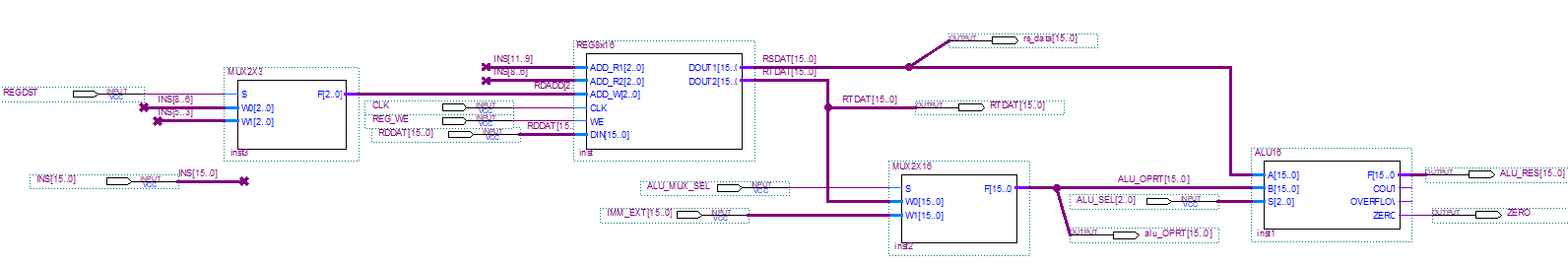


Figure : MP16dp2 combining the ALU and registers

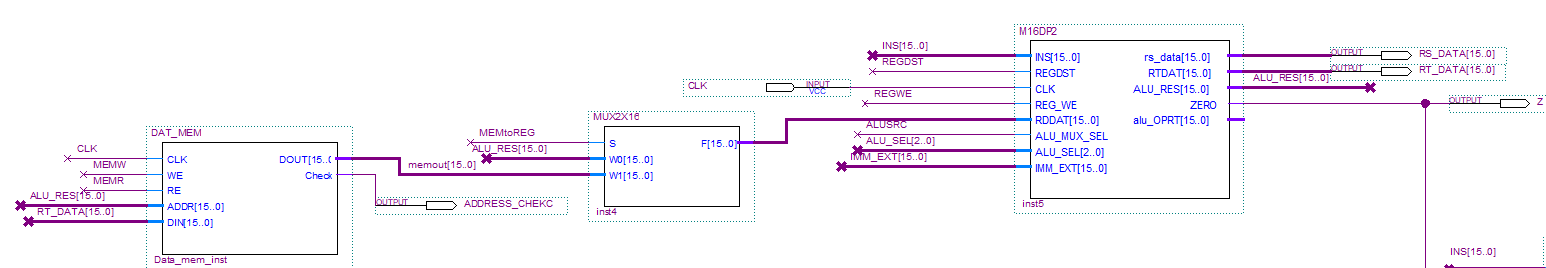


Figure : 2X16 Multiplexer for REG write selection.

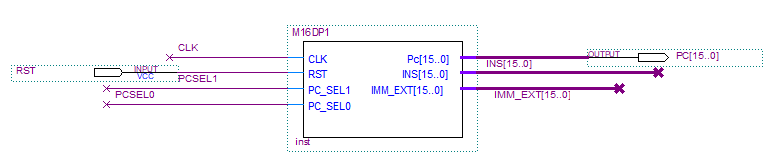


Figure :M16dp1 block file

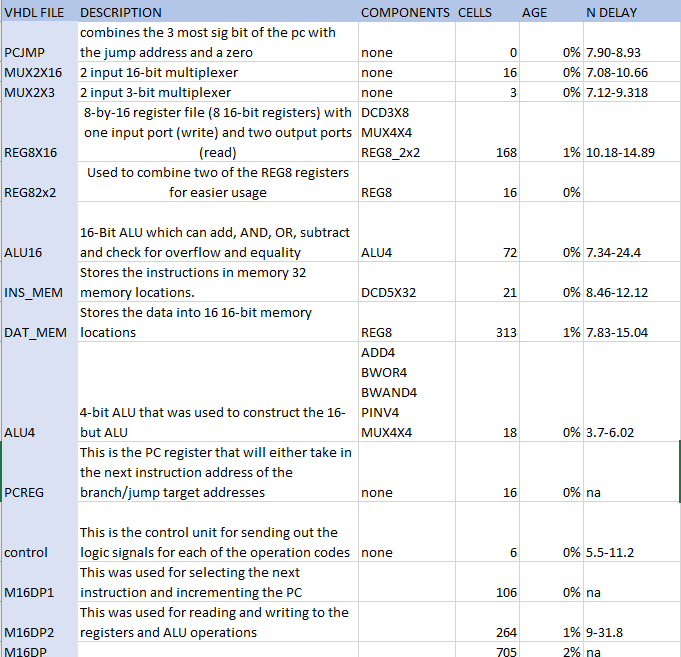


Table : cell usage and timing delays

# References

|  |  |
| --- | --- |
| [1] | D. A. P. a. J. L. Hennessy, Computer organization and design: the hardware/software interface, Morgan Kaufmann Publishers, 2021. |