Project - RTL Model

Part 4 - May 17, 2005

Jason Nemeth

I. Introduction

The purpose of this assignment was to refine the structural model of the microprocessor into a working RTL-level VHDL implementation. All elements of the previous reports were updated and included to provide an accurate portrayal of the project's progress thus far.

II. Requirements

The microprocessor must have the following capabilities:

1. Direct access to internal memory with at least 256 words.
2. An instruction set capable of executing at least the following two benchmarks:
   a. Output 10 signed integers in sorted (largest first) order where the numbers are input in random order.
   b. Output the mean (to the nearest integer) of 16 signed integers read into the processor.
3. One 8-bit input port, and one 8-bit output port.
4. Optional one single-bit Input enable line, and one single-bit Output ready line

The overall layout of the microprocessor is shown below in *Figure 1*. The behavioral model was created such that all these characteristics were met. The model is capable of modeling both provided benchmarks.

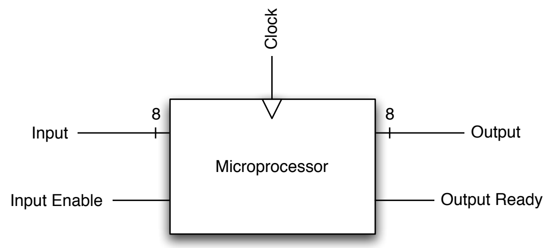

Figure 1. Layout of the Microprocessor

III. Design Decisions

Opcodes

Once the pseudo-code for the model's implementation was complete, the required opcodes were determined. Listed below in *Table 1* are the 8 opcodes utilized in the model.

Opcode	Operation
000	Push
001	Pop
010	JumpNot0
011	Jump
100	Add
101	Sub
110	ShiftR4
111	NoOp

Table 1. Opcodes and their associated operations

The above opcodes allow for the proper averaging and sorting of a number of 8-bit input values.

Structural Layout

The next step that needed to be completed was the actual layout of the microprocessor in structural terms. As could be noted by examining the opcodes listed above, this microprocessor utilizes a stack machine-based architecture. The design allows for a Program Counter, Instruction Register, and Stack Pointer, each directly mapped to memory locations within RAM. Directly-mapping each of these components allowed for a design completely free of external registers, using only the main 256-word RAM for operations.

A relatively shallow hierarchy was chosen due to the large number of components that needed to be directly mapped to RAM. A simplified overview of the system is shown in *Figure 2*. This figure shows the major interactions between the components, as well as the logic and control signals produced by the Controller that synchronize the system.

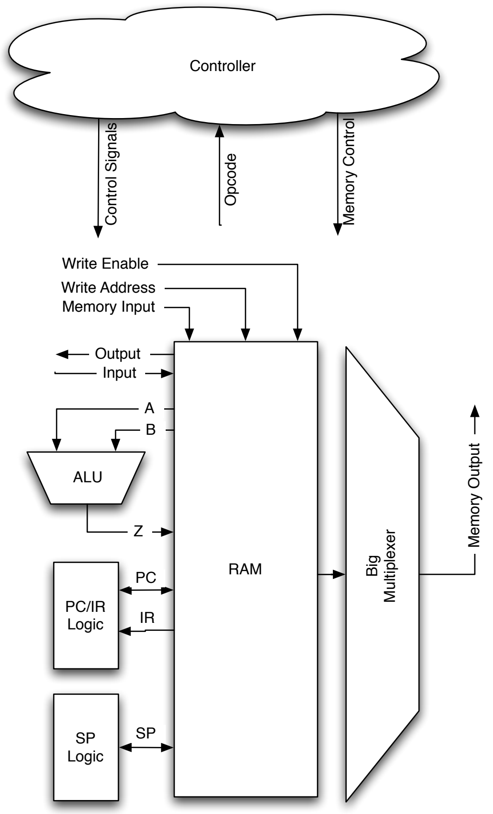

Figure 2. Overview of the microprocessor

Memory Implementation

Most of the time planning the model was spent working on the layout of the main memory. Due to the relative abundance of stack operations needed when using a stack architecture, a memory system was developed such that memory reads and writes can take place simultaneously. This allows for stack operations to be completed in a single clock cycle by simultaneously, for example, reading a memory address and pushing it onto the stack. A more detailed representation of the operation of the main memory core is shown in *Figure 3*. This operation is achieved by linking the output of the BigMux to the memory's input line during stack operations.

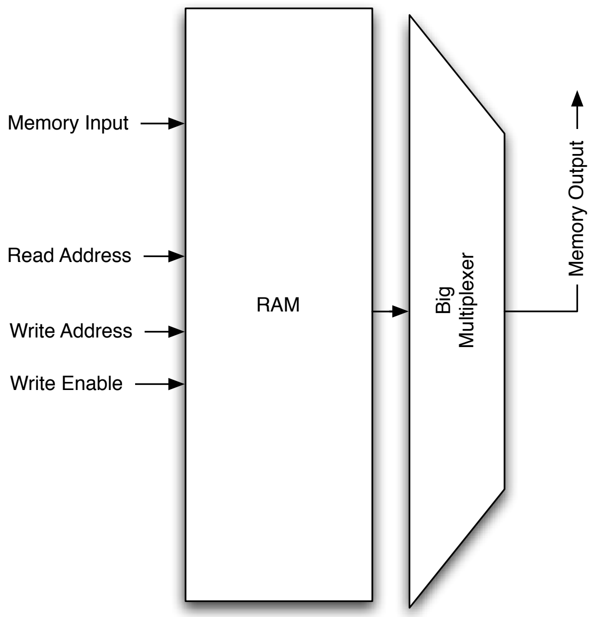

Figure 3. Memory access procedure layout

Memory Access

Since more than four, but less than eight opcodes were required for the system, each opcode must occupy three bits. Unfortunately, this leaves only five bits for addressing, covering a maximum of 32 memory locations. It was decided to directly map those 32 locations to positions 0 through 31 in main memory. Doing so allows for the direct access of each of those locations by the Push and Pop commands. Indeed, Push and Pop can only be utilized on items contained in the first 32 words of memory. It was therefore necessary to locate every important direct-access item within that area, including data storage locations. *Table 2* below shows the orientation of those items in memory.

Memory Location		Function
31	00011111	A
30	00011110	B
29	00011101	Z
28	00011100	PC
27	00011011	IR
26	00011010	SP
25	00011001	Constant9
24	00011000	Constant10
23	00010111	Constant0
22	00010110	Constant-1
21	00010101	Input
20	00010100	Output
19	00010011	j
18	00010010	i
17	00010001	DataTemp
16	00010000	Data16
15	00001111	Data15
14	00001110	Data14
13	00001101	Data13
12	00001100	Data12
11	00001011	Data11
10	00001010	Data10
9	00001001	Data9
8	00001000	Data8
7	00000111	Data7
6	00000110	Data6
5	00000101	Data5
4	00000100	Data4
3	00000011	Data3
2	00000010	Data2
1	00000001	Data1
0	00000000	Data0

Table 2. Special Memory Locations

The 3-bit opcode also had an effect on the Jump and JumpNot0 operations. Since only five bits are available, it is not possible to send the Program Counter to a fully-qualified memory address. Instead, relative addressing was utilized. This signed system allows for jumping ahead 15 or back 16 memory locations from the current address in the Program Counter. This is a restrictive design, and causes extra operations in the sort operation because Jumps greater than 15 memory locations are sometimes required, which involves multiple hops. This is an unfortunate side-effect, but was ultimately the best solution as merely assigning the upper three bits of the address to a specific value would not allow complete utilization of memory for program code. This solution allows for Jumps within the system, while allowing all 256 words to be addressable.

ALU Implementation

The ALU was implemented completely memory-mapped. Its inputs A and B are found in memory locations 31 and 30, and its output Z can be found in location 29, as shown previously in *Table 2*. This allows for the utilization of direct-mapped memory addresses to perform the mathematical functions Add, Subtract, and Shift Right 4 Bits. The ALU gets passed opcode information, performs the associated operation, and stores the result in Z. This result is then directly accessible by the Push and Pop operations.

Design of Operation

The microprocessor goes through three distinct phases during the course of executing a single instruction. Each phase takes 34ns, which is the speed of a single clock cycle, meaning an entire operation takes 102ns. The Fetch phase allows for the retrieval of an instruction from memory. The Decode phase allows for the translation of the instruction into opcodes, and also the incrementing of the Stack Pointer during a Push operation. The Execute phase executes the instruction being performed, including Add, Subtract, and ShiftRight4. When an instruction has completed execution, the machine returns to the Fetch phase once again, allowing the process to be completed all over again. The 30 ns clock cycle had to be increased slightly for the RTL implementation due to memory read and write problems. 30 ns was simply not enough time for the instruction to be decoded, an item to be read from memory through the large set of multiplexers, and rewritten into a different location. However, it was close. The RTL level was able to reliably operate at a clock cycle of 34 ns, as mentioned previously.

Memory Problems

The designed memory system has a problem when implemented at the RTL level. After extensive testing, the RTL memory structure performs flawlessly. However, it is currently not possible to initialize the memory including the required assembly code, Stack, and Program Counter pointers. This had not been considered due to how easy it is to initialize devices in a behavioral design. The changes needed to be made to the system to enable such an initialization were extensive. Unfortunately, this required the structural model of the memory to be used in the final calculations. This was easily performed, since the memory structure is very simple, by making the performance of the structural output waveforms match those of the known RTL level.

IV. Results

The RTL level model of the Microprocessor is, for the most part, up and running successfully. The model's behaviorally-described structures were taken down to the gate level. Despite the minor memory problems with the system, the model functions as designed. Unfortunately, the problems with the Sort function encountered in the previously-submitted Structural Model were unable to be resolved. The performance metric included below is calculated based on the average performance of the Bubble-Sort algorithm utilized, and how the sort was implemented in assembly. Had the sort worked, it would have been a good average value for the metric.

Benchmark	Time (μs)	Metric (Gate*μs)
Average	8.11	92738
Sort	15.6	178386

Table 3. Benchmark results and metrics

As can been seen in *Table 3* above, the RTL model beat the performance metrics estimated in the previous structural model, even though the clock speed had to be slowed slightly. This is due to the fact that significantly fewer gates were required. The structural model estimated approximately 15,000 total gates (including latches), while the actual gate count came to 11,435. This is likely due to a slight logic minimization made to the 2-into-1 multiplexer units. This minimization was, however, quite significant due to the approximately 2,080 2-into-1 multiplexers used in the design.

Table 4 shown on the next page gives a structure-by-structure breakdown of the number of gates required. Each of the structures presented in *Figure 2* on the second page of this report are included. The multiplexer required for 256-word access utilized the most gates by far, followed by the latches used to implement the memory itself. The other components, the ALU, Stack Pointer logic, and Program Counter/Instruction Register logic combined take up less than 4% of the gate area of the microprocessor.

Structure	Gates
BigMux	8192
ALU	136
Controller	70
RAM	2816
SP	115
PCIR	106

Table 4. Number of gates in each structure

V. Conclusions

The results of the RTL model will now be used to perform an analysis of the system, yet to be determined. The model could be used as-is to perform the analysis since the majority of its components are down to the RTL level. However, it may be possible to come up with a way to initialize the RTL memory structure before beginning the analysis. Unfortunately, at this point I am fairly certain that the bug in the Sort routine cannot be located, causing one of the two benchmarks to run improperly.

This model is the best indicator of the microprocessor's performance so far. By using less gates than expected, the system was able to beat the estimated performance metrics created for the previous Structural Model by a significant amount.