Matthew Champagne **Final Project** ESE 345 Fall 2021

ID- 112540003 Ryan Thielke

Goals-

The goal of this project was to make a 4 stage pipelined MMX processing unit and an assembler.

Block Diagram-

Graphical user interface, diagram, schematic

Description automatically generated

\*Auto Generated Block Diagram from Aldec Active HDL

Diagram, schematic

Description automatically generated

\*Hand drawn Diagram with all signal widths, signal transitions, and blocks labeled. The abbreviation SL or SLV followed by a number stand for the standard logic and standard logic vector data type respectively. Full resolution pdf was submitted digitally.

Design Procedure-

When designing a CPU I think it is easiest to start with the data path because it is the most straight forward to implement. Starting with the ALU you implement each instruction listed in the requirements and assume certain registers contain the values you need for operations without worrying how the data will get there. The next block I implemented was the Instruction Fetch Unit which consists of the Program Counter and the Instruction Memory. With this block you can load in instructions from a text file and have the program counter point to one of the instructions each clock cycle. The next block I implemented was the Instruction Fetch/Decode Register. This register takes in the 25 bit instruction and splits it up into every possible field that is possible from the three instruction formats. The register passes those fields to the Register File, Control unit, and the Instruction Decode/Execute Register. The Control Unit is a simple combination block that generates the ALU opcode and the register write. The Register File is implemented as a three read one write per cycle with bypass. All of these blocks pass there values to the biggest register in the project the Instruction Decode/Execute Register. This register passes some of its values to the ALU Register Decoder. This unit basically outputs the proper values to the ALU based on its inputs. The output of the ALU goes to the Execute/Write Back register. Its job is to perform the write on the register file and pass the proper signals to the Forwarding Unit. The forwarding takes in the address of the destination register of one cycle and compares it to the source registers of a later cycle to find dependencies. Once it finds dependencies it signals the ALU Register Decoder to use the forwarded value in one of the three outputs to computer the proper value. While this one instruction is being processed each cycle a new instruction is being loaded in.

Concluding Remarks-

This MMX Unit is not very complicated to implement because it has no branching logic. This means that it cannot execute and conditional statements or jumps which simplify the implementation significantly. If it did have branching logic the pipeline implementation would have to be a lot more complex to avoid hazards from branching while still dealing with data hazards. The Program Counter value would also have to be manipulated very carefully to return to the proper instruction after the pipeline has been flushed. Without branching logic this unit is essentially a pipelined calculator. Not having conditionals makes programming anything other than mathematical calculations impossible.

Program Used In Simulation-

The point of this program is to see whether or n! grows faster. It first computes the first five terms of each function and places them in registers 3 through 7 and 9 though 12 respectively. You will notice that one function uses 5 registers and the other uses 4 registers this is because n! first two terms starting from zero are identical. It then takes the values and finds the max of the two values term by term and places them into registers 14 through 18. The next step is to XOR registers 14-18 with registers 3-7. The point of this is to see if the MAX register is equal to the exponential function. This is possible because something XOR with itself is equal to zero. So the output of the program is registers 14-18 which tell you whether the MAX value is equal or not equal to the exponential function. The results will show that this is true for the first four terms starting from zero. Then the factorial function grows faster after the fourth term.

A screen shot of a computer

Description automatically generated with low confidence

\*Program to compute whether an exponential function or a factorial function grows faster

Simulation Results-

Waveforms-

Application, table

Description automatically generated

\*Showing the data flow through the CPU in a pipelined architecture. Rows are labeled with the blocks they are from to show the dataflow. Note cop 0x0 is load immediate, cop 0x1 is Signed Integer Multiply add low with saturation, and cop 0xF is MAX.

Graphical user interface, application

Description automatically generated with medium confidence

\*End state of the register file after the program is run