EE480 Assignment 3: AXA Pipeline

Implementor's Notes

Peyton Moore
Electrical and Computer Engineering Student
University of Kentucky, Lexington, KY USA
Peyton.Moore1@uky.edu

Michael Probst
Electrical and Computer Engineering Student
University of Kentucky, Lexington, KY USA
M.Probst@uky.edu

Alex Schuster
Electrical and Computer Engineering Student
University of Kentucky, Lexington, KY USA
Alexander.schuster@uky.edu

ABSTRACT

This is a pipeline processor design that using the AXA instruction set created by Dr. Hank Dietz. The instruction set is design to have forward and backward instruction sets. For simplicity sake, this processor design is only designed using the forward instruction set and contains the UNDO buffer for reverse instruction sets.

1. GENERAL APPROACH

This processor design was created following the AXA assembler instruction set provided by Dr. Hank Dietz. All instructions are implemented with the forward instruction only and push and pull to the UNDO buffer needed for reverse instructions.
The pipeline design is based on a multi-cycle design provided by Alex Schuster's team. For the design of the pipeline, there will be five stages. Stage 1 is an instruction fetch. Stage 2 is a register read. Stage 3 is a data memory access. Stage 4 is an ALU operation and the register write.
Stage 1 of the pipeline will increment pc counter as well as fetch the instruction. When it is able to succeed with this, an instruction register tied to stage 2 will be set to allow Stage 2 to progress. Stage 1 will handle all jumps by setting the pc counter to the jump address. This stage will also pass the src type between stages. For example, if an instruction contains an I4 source, the source will be passed to stage 3. This stage will write the PC to the undo buffer if the **land** command is received.
Stage 2 of the pipeline will handle register reading of the pipeline. This involves reading the src value of a register is sent with an instruction. if the from the registers are taken correctly then Stage 3 is able to run. This stage also stores the destination register in the undo buffer if the following instructions are given: **lhi,llo,or,dup,and,shr**. Stage 2 checks to see if the previous instruction and the next instruciton that Stage 1 provides have data dependicies. If they do we wait until the current instruciton is finished.
Stage 3 of the pipeline will handle the data memory accessing of the pipeline. If an address to a memory is sent with the instruction. This stage will handle accessing the data. If there is no address sent, then Stage 3 will pass on the src type found in Stage 1 or Stage 2 depending on the type via a mux. If the data contains info for a jump, that is fed back to stage 1.
Stage 4 of the pipeline will handle ALU operations and writing to the register file.
The processor design works by using 16 bit word instructions that allocate the following bits for any instruction:

OPCODE: 6 Bits
Instruction source: 2 Bits
Destination:4 Bits
Source: 4 Bits

The processor designed had a special 16 bit word instruction for the following instructions **xhi,xlo,lhi,llo**, the follow bits are:

OPCODE: 4 Bits
Instruction source: 8 Bits
Destination:4 Bits

The instruction **l16** is a synthesized instruction based on the instructions using an immediate 8 bit value. It works by using two of the instructions based on the best pair to use.
The OPcodes are designed in this to be OPinstruction name. For example the **ex** instruction OPcode is **OPex**.
This led to a design that implemented finding the instruction based on the OPcode and then matching it to an execution based on the instruction source. The following instruction sources are supported:

$s: A source Register
i4: A 4-bit immediate value
@$s: A source register for indirect data memory address
i4$: A 4-bit unsigned immediate value to be used to index the undo buffer

The processor was designed to be one module that took the inputs of a clock, reset, and outputted a halt. The following memory blocks were allocated:

Mem0: Register File
Mem1: Data Memory
Mem2: Instruction Memory

A test bench was used to test the processor. The test bench

ACM ISBN N.A..
DOI: N.A.

module produced a clock that ticked every 10 simulation cycles and would continuously run until a halt occurs. If a halt occurs, the processor would stop. The test bench is designed to be checked using a waveform checker and comparing the output of the register to the input registers. Each instruction was tested individually.
Testing was done using sixteen registers in **Mem0** block and data put in **Mem1** block. The instruction set used was handled through an AIK assembler interpreter. The instructions file produced by this was put in **Mem2** block. A test case that runs through each command can be found in **test.txt**.

2. ISSUES

There were no issues with implementation and testing.

3. REFERENCES

This implementated used PinKY from EE480 Fall 2018 pipeline as a reference.
This can be found at
http://aggregate.org/EE480/pipepinky.html