TACKY Multi-Cycle Implementation: Implementor's Notes

Brian Carlson
Samuel McCauley
James Gallagher
brian.carlson1@uky.edu
sammccauley117@gmail.com
James.Gallagher@uky.edu
Lexington, Kentucky

ABSTRACT

This project was concerned with creating a multi-cycle implementation of the processor and memory for the TACKY instruction set. Using an amalgamation of the instruction set encoding specifications from the second assignment, we encoded our instruction set using AIK.

1 GENERAL APPROACH

In order to complete this project, the group followed a top down approach. The processor is contained in one Verilog module, with help from external floating point calculation modules.

- The instructions that run on this processor are compiled using AIK and our pre-defined specification (see Implimentation Details). These instructions are loaded into the processor using Verilog's VMEM.
- The processor splits the instructions into two categories that are executed separately. The first is the packable VLIW instructions such as add, sub, div, mul, etc. The second was the non-packable instructions such as pre, ci8, cf8, etc.
- The 16-bit registers actually had a special 17th bit that differentiated whether the register was currently holding a float or an integer. A 1 in the 17th bit correlated to a float, a 0 correlated to an integer.

2 TEST PLAN

The validity of these instruction results were viewed using both a simple testbench that implements $dumpvars as well as simply calling $display to show the contents of each register at every processor cycle.

For this assignment we began testing by inputting very simple programs like a test that immediately terminated to make sure the halting and control logic on our processor worked. Then we tested a simple addition program, which gave the correct output. we also tested a simple floating point arithmetic program, a more complex integer arithmetic program, and finally a large complex program that contained most of the instructions in TACKY.

3 IMPLEMENTATION DETAILS

Below is the AIK implementation code:

```
; 8 bit operators
.C1 $.r := .this:5 .r:3
.C1 $.r0r , .C1 $.r1r := .this:5 .r0r:3 .C1:5 .r1r:3
.alias .C1 a2r r2a lf li st cvt sh slt add sub mul div
    not xor and or jr

; 13 bit + 3 bit padding
.C2 .imm := .this:5 0:3 .imm:8
.alias .C2 17 pre jp8 sys

; 16 bit
.C3 $.r, .imm := .this:5 .r:3 .imm:8
.alias .C3 20 cf8 ci8 jnz8 jz8

; 32 bit macros
.C4 $.r, .imm := pre:5 (.imm>>8):8 (20 + .this):5 .imm:8
.alias .C4 cf ci jnz jz

jp addr := pre:5  0:3 (addr>>8):8 18:5 0:3 addr:8

.const {r0 r1 r2 r3 r4 ra rv sp}
```

4 ISSUES

We did not run into any errors when testing our project, but we did not do 100 percent line coverage tests, so it is possible there are errors. We tested a couple of instructions from each category of instruction, things like integer arithmetic, float arithmetic, jumps, and other instructions.