# Cowgirl Platform

thebeekeeper

October 2005

## 1 Overview

Here's an ISA and implementation I've been working on. I had the idea quite a few years ago, but I'm finally getting around to doing it. I'm writing the entire thing in VHDL.

## 2 Instruction Set

The instruction set is pretty generic and uninteresting. It's pretty much a standard RISC set that you can get out of any textbook.

### 2.1 Instruction Set Overview

Here's a quick list of the instructions.

| | |
|---|---|
| add d r1 r2 | Add |
| sub d r1 r2 | Subtract |
| mul d r1 r2 | Multiply |
| div d r1 r2 | Divide |
| neg r | Arithmetic Inverse |
| not r | Binary Inverse |
| and d r1 r2 | Logical And |
| or d r1 r2 | Logical Or |
| xor d r1 r2 | Logical Xor |
| nand d r1 r2 | Logical Nand |
| nor d r1 r2 | Logical Nor |
| mov d r1 | Move Register Value |
| comp r1 r2 | Value Comparison |
| jmp r1 | Unconditional Jump |
| bne r1 | Branch if Not Equal |
| be r1 | Branch if Equal |
| ret | Return from Jump/Branch |
| sw d r1 | Store Word in r1 at d |
| lw d r1 | Load Word in r1 at d |
| lli d imm | Load Lower Immediate into d |
| lui d imm | Load Upper Immediate into d |
| nop | No Operation |
| halt | Halt the Processor |

## 2.2 Full Description

Here's what's really going on with these instructions.

| Bit Pattern | Instruction | Description |
|---|---|---|
| 0000 | nop | Do nothing |
| 0001 | ret | Return to address saved during jump/branch |
| 0010 ddd rrr rrr aaa | ALU Op | Arithmetic ALU operation. The bits in aaa determine which arithmetic operation |
| 0011 ddd rrr rrr aaa | ALU Op | Logical ALU operation. The bits in aaa determine which logical operation |
| 0100 ddd imm l | lli/lui | Load or store immediate. imm is 8 bits. l specifies lower/upper part 0-lo 1-hi |
| 0101 rrr 0000000 aa | Jump/Branch | Register stores address to jump to, aa specifies what type of jump |
| 0110 rrr nnnn 0 aa | Shift | Shift value in rrr by nnnn. aa specifies type, left/right, logical/arithmetic |
| 0111 rrr rrr | Compare | Compare values in registers. Set flags |
| 1000 ddd rrr | Move | Move value in rrr to ddd |
| 1001 ddd rrr 00000 s | Store/Load | Store value in rrr to memory location in ddd. s is 0 for store, 1 for load |
| 1111 | Halt | Stop the processor |

## 2.3 Sub-Opcodes

There are specifiers for some of the opcodes. I think I did that because it help me fit some things into 16 bits. I can't really remember. Anyways, here's how that works. These are what the aaa's mean in the previous table.

Table 1: Arithmetic Opcode Specifiers

| | |
|---|---|
| 000 | Add |
| 001 | Subtract |
| 010 | Multiply |
| 011 | Divide |
| 100 | Negate |

Table 2: Logical Opcode Specifiers

| | |
|---|---|
| 000 | Not |
| 001 | And |
| 010 | Or |
| 011 | Xor |
| 100 | Nand |
| 101 | Nor |

Table 3: Jump/Branch Specifiers

| | |
|---|---|
| 00 | Jmp |
| 01 | Bne |
| 10 | Be |

Table 4: Shift Specifiers

| | |
|---|---|
| 00 | SLA |
| 01 | SLL |
| 10 | SRA |
| 11 | SRL |

# 3 Programming Model

There's not really much to programming this processor. The important things are:

There are 8 General Purpose Registers. They're called r0 ... r1.

Umm... I don't know what else.

# 4 Development Tools

So that I don't have to hand assemble code, I wrote an assembler. And so I don't have to hand write VHDL to make a ROM, I wrote a program to do that.

## 4.1 cgasm

The program I wrote to convert assembly languange programs into binary machine code is called **cgasm**. You give it an asm file, and it generates a file with a bunch of 1's and 0's. It ignores lines that start with #. Here's an example file:

```
# Cowgirl Test Program - cgtest.cgasm
# add 2 + 2
lli r0 2
lui r0 0
lli r1 2
lui r1 0
add r2 r0 r1
```

So, if I want to make a binary file called cgtest.bin, I'd go:

```
cgasm cgtest.cgasm cgtest.bin
```

And that's it!

## 4.2 bin2rom

I wrote this quick program to generate a VHDL Rom module from the bin file. It's pretty simple. You just run it like this:

```
bin2rom cgtest.bin cgtest.vhdl
```

And it generates synthesizable VHDL for you. Simps!