The g3l Instruction Set Architecture

Meta

Author: Gregory N. Schmit

Date: 29 Oct 2017

Introduction

The goal of the g3l instruction set architecture (ISA) is to provide a simple application programming interface (API) to execute a single program that finds the largest number (and index of) in a given array in memory, and finds the remainder of a division operation (again, stored in memory).

The primary goal was to reduce the amount of instructions provided by the ISA.

Format of Memory and Instructions

Registers and memory are tribble-addressable. The bus for instructions is 8-bits long, however only 5-bits are relevant.

.data starts at `0b00000000` and .text starts at `0b10000000`. Memory is only 256 tribbles total.

Registers (8 total 12-bit registers)

There are eight 12-bit (tribble) registers.

- r0 - zero
- r1 - 3rd large stack
- r2 - 3rd large stack
- r3 - 3rd large stack
- r4 - 3rd large stack entry
- r5 - for loop counter
- r6 - mem ptr
- r7 - mem target

Instructions

Instruction	Argument	Description	Encoding (5-bit)	Section
6set0000		Set $r6 to 0b000000000000	0b11000	11xxx - Setting + Halt
6set0111		Set $r6 to 0b000000000111	0b11001	11xxx - Setting + Halt
7set0110		Set $r7 to 0b000000000110	0b11010	11xxx - Setting + Halt
7set0101		Set $r7 to 0b000000000101	0b11011	11xxx - Setting + Halt
7set0011		Set $r7 to 0b000000000011	0b11100	11xxx - Setting + Halt
7set0010		Set $r7 to 0b000000000010	0b11101	11xxx - Setting + Halt
7set0001		Set $r7 to 0b000000000001	0b11110	11xxx - Setting + Halt
halt		Halt	0b11111	11xxx - Setting + Halt
neginf	$rlox	Set low register ($rlox is element of $r0-$r3) to lowest number	0b000xx	000xx - Special Set to Negative Infinity
incr6		Increment $r6	0b00100	0010x - Incrementing

decr5		Decrement $r5	0b00101	0010x - Incrementing
8decr6		Decrement $r8 by 6	0b00110	00110 - Special Increment
lt6hi4		Load tribble pointed to by $r6 to $r4	0b01000	0100x - Load from Memory
lt6hi5		Load tribble pointed to by $r6 to $r5	0b01001	0100x - Load from Memory
st37		Store $r3 to memory pointed to by $r7	0b01010	01010 - Special Store to Memory
swplt12		Swap $r1 and $r2 if $r1<$r2	0b10000	100xx - Register Swap
swplt23		Swap $r2 and $r3 if $r2<$r3	0b10001	100xx - Register Swap
swplt34		Swap $r3 and $r4 if $r3<$r4	0b10010	100xx - Register Swap
swp36		Swap $r3 and $r6	0b10011	100xx - Register Swap
sub37		Subtract $r4 from $r3, store in $r3	0b10100	10100 - Special Subtract
b5nzj7		Branch if $r5 != 0 to PC-$r7	0b01100	011xx - Branching
		Branch if $r3 != $r7		011xx -

b34neqj7		to PC-$r7	0b01101	Branching
b34ltejf7		Branch if $r3 < $r4 to PC+$r7	0b01110	011xx - Branching
j2		Jump two instructions backwards (PC-2)	0b01111	011xx - Branching

Answers to Questions

1. Will your ALU be used for non-arithmetic instructions (e.g., address calculation, branches)? If so, how does that complicate your design?
    - Yes, mainly swapping which took a set of multiplexors that I didn't anticipate.
2. What is your most complex instruction, from the standpoint of the ALU?
    - Swapping registers
3. Is there anything you could have done in your ISA to make your ALU design job easier?
    - No, the ALU seemed relatively straightforward.
4. Is there anything you could have done in your ISA to make your register file design job easier?
    - Yes, think harder about the instructions I wanted to support, especially now I realize that maybe only swapping two specific registers would've been easier and faster.
5. Now that your ALU is designed, are there any instructions that would be particularly straightforward to add given the hardware that is already there?
    - Yes, any setters or arithmetic would be easy to implement.