EE480 Assignment 1: Logick Encoding And Assembler

Implementor's Notes

Hank Dietz
Department of Electrical and Computer Engineering
University of Kentucky, Lexington, KY USA
hankd@engr.uky.edu

ABSTRACT

This project involved the design of an encoding scheme for the Logick instruction set. The encoding was then to be embodied in an AIK specification.

1. GENERAL APPROACH

The first issue is how to encode 20 instruction types in a 16-bit instruction words where 12 bits are needed for the operands of some instructions. The remaining 4 bits can only distinguish 16 instruction types. Complicating matters a bit, I also wanted the encoding of similar integer and LNS operations to differ in just one bit. I further imposed the constraint that I wanted the `sy` instruction to encode as 0, so that any instruction sequence wandering into empty memory will cause a trap to the operating system.

The resulting scheme divides what could be viewed as a 4-bit opcode into 3 bits of opcode and single bit that is 0 for integer operations, and 1 for LNS operations.

I also wanted the destination register to always be in next 4-bit field, but took advantage of the fact that register 0 is not a valid destination to keep the two sources in the last two 4-bit fields for the `cl` and `co` comparisons and also to distinguish `st` from `or`. I was able to keep the two registers being read as always the last two 4-bit fields, except for the `li` and `si` instructions, which are still consistent about where the destination register field is.

The condition code selection is really just 3 bits, but I put that into a 4-bit field by using 1 bit at the end to distingush between `br` and `jr`.

2. MACROS

Two intelligent macros are implemented that pick the smallest-size implementation of a pseudo-instruction. The `la` operation either does an `li` or an `li` followed by an `si`. Similarly, `jb` selects between a `br` and a `jr`, but loading the register is further optimized rather like `la` was. There is also the slight complication that `u10` is used to hold the target address, so it had better not be used to mean something else in the program.

3. CONSTANTS

Not much to report here; all constants were given the values that correspond to the order they were listed in the assignment.

4. ISSUES

Really nothing terrible. There was a potential name conflict between use of `ne` to mean negate or not-equal, but this was resolved by making the negate instruction be called `mi`, for minus. I avoided using `.alias` because I think this spec makes a better human reference for what the assembly language looks like. I also changed the symbolic names for some operands to make their field use more obvious – e.g., `st` has arguments t and s, not d and s.

The hardest thing to do is to test this. It really isn't testable in any obvious way... so this was tested only by manual inspection with a variety of simple test cases, including the one given in the handout. There were no errors flagged by AIK.

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