TACKY Encoding And Assembler: Implementor's Notes

Brian Carlson
brian.carlson1@uky.edu
UKY ACM Chapter
Lexington, Kentucky

ABSTRACT

This project was concerned with the development of a custom encoding scheme of the TACKY instruction set using AIK. This is the initial design document without optimizations and will likely undergo changes as the implementation of the hardware begins.

1 GENERAL APPROACH

TACKY contains 24 instructions and 8 registers, requiring 5 bits to encode the instructions and 3 bits to encode the registers without optimizations. TACKY also contains 5 macros that lead to two 16 bit words being created.

To facilitate organizational structure and a good setup for bit packing later, I separated the instruction set into five groups:

- Statements containing just an instruction and register reference - this allows the assembler to fit an instruction into 5 bits and the register in use into 3 bits. This allows for packing two instructions per 16 bit word
- Statements containing an instruction and an 8 bit number - this requires the assembler to fit an instruction into 5 bits and the 8 bit number into 8 bits. This also requires 3 bits of padding to complete the 16 bit word
- Statements containing an instruction, a register reference, and an 8 bit number - this requires the assembler to fit an instruction into 5 bits, the register into 3 bits, and the 8 bit number into 8 bits. This completely fills the 16 bit word
- Macro statements containing an instruction, a register, and a 16 bit number - this requires the assembler to split the instruction into two other statements: a *pre imm*8 and the equivalent instruction, register, 8 bit number combination
- The *jp addr* macro doesn't require a register, so the encoding requires two sets of padded instructions *pre imm*8 and *jp*8 *imm*8 from the second group above where imm8 are different halves of the 16 bit addr.

2 ISSUES

The documentation for AIK wasn't organized in a manner that facilitated learning how to use it to the best capacity. I had trouble understanding how to implement some of the bit optimization hacks that had I thought of for encoding instructions due to the lack of clarity in the documentation.

3 IMPLEMENTATION

Below is the AIK implementation code:

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

; 8 bit operators
.C1 $.r := .this:5 .r:3
.C1 $.r0 , .C1 $.r1 := .this:5 .r0:3 .C1:5 .r1:3
.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 .imm:8 0:3
.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 := 17:5 (.imm & 0b1111111100000000):8 \
(20 + .this):5 (.imm & 0b0000000011111111):8
.alias .C4 cf ci jnz jz

jp addr := 17:5 (addr & 0b1111111100000000):8 0:3 18:5 \
(addr & 0b0000000011111111):8 0:3
```