1 Introduction

2 Typed Assembly Languages

Typed assembly langauge (TAL) provides types to to memory and registers through assertions on labels. It does not make much sense to "type" an instruction, as these already are already defined on only one type.

Label types contain information on what types are contained within relevant registers and memory addresses.

A well-formed typed assembly program contains no references to values that are not within one such type—and does not violate them if one such exists.

Given some types t of data

$$t ::= \{r_1 : t_1, \cdots r_n : t_n\} \tag{1}$$

allows us to assign types to registers following a label, so

```
fib: {a1 : int, sp : [sp :: int], ra : {a0 : int}}
          addi     a1, a1, -1
          bltz     ...
```

is a function that takes on ints, and jumps to a place where a0 is expected to be an int. We also type the stack to contain this, as well (since we will be pushing and popping for the recursive calls). This makes sure that fib pushes and pops the right type.

```
ouch: {a1 : int}
          jmp      a1
```

$$\forall \alpha, \beta. \{r_1 : \alpha, r_2 : \beta, r_3 : \{r_1 : \beta, r_2 : \alpha\}\} \tag{2}$$

might describe a label that swaps the values of two registers.

Most commonly used in correctness analysis and ..

2.1 Judgements / rules

2.2 abstract Machine

Possibly skip