Why CoGenT?

- Hand-crafting compiler and simulator components is:
  - Time-consuming
  - Error-prone

- Current available simulators are:
  - Difficult to modify
  - Difficult to port
  - Difficult to instrument
  - Difficult to optimize

The CoGenT Project

- Co-generation of Compiler and Simulator Tools
- Uses a single set of specifications to produce:
  - Compiler back-end components
    - Register allocators
    - Code generators
    - Instruction schedulers
  - System tools
    - Assemblers / Disassemblers
    - Debuggers
  - Simulators
    - Functional/Instruction-accurate
    - Cycle-based

Why a common machine description?

- Compiler research requires fast & accurate simulators
- Architecture research requires optimized compiler backends
- Generate both from a common machine description
- Common language:
  - Eliminates redundancy
  - Simplifies MD compiler
  - Assures synchronization of simulator and compiler
- Previous work solves only some of these problems
The CoGenT Language - CMDL

- Specification language for:
  - Instruction set syntax
  - Instruction set semantics
  - Machine store descriptions
- C/Java-inspired syntax
- Elements of other machine description languages
- Allows partial descriptions (details filled in by tools)
- Allows class hierarchies to aid description

Bits: The only principal type

- Everything is bits
- Single bits are valid
- Bit Arrays are also allowed
  - Defined by length
  - Multidimensional arrays are supported
- References provide named sub-regions
- Type modifiers express abstract qualities
Type Modifiers and Declarations

- Current Modifiers:
  - Signedness: unsigned/signed
  - Endianness: big/little
  - Mutability: overlay/field

- Handling conflicts
- Type declarations using type ...

Examples

- These type identifiers are equivalent:
  - Posix: uint32_t
  - CoGenT: big unsigned bit[32]

- These type declarations are equivalent:
  - typedef uint32_t unsigned int
  - type uint32 = big unsigned bit[32]

- References:
  uint32 foo;
  bit hi @ foo[0];
  bit lo @ foo[31];

Endianness

- Endianness specifies meaning of indexing
- Example: bit[2] foo = 0b10;
  - Big Endian foo[0] = 1
  - Little Endian foo[1] = 1

- Bits reordered through assignment
- Indexing converted through coercion

Subranges

- Named ranges on types (rather than instances)
- Example:
  big unsigned bit[32] ieee32fp;
  subrange sign = bit @ ieee32fp[0];
  subrange exponent =
    unsigned big bit[8] @ ieee32fp[1];

- No recursive subranges
- Different signedness and endianness allowed
**Classes**

- Data + Methods
- Simple single inheritance
- No references
- Restricted form of inclusion
- Useful for specifying orthogonal instruction sets

**Instruction Definition Example**

Our friend, the PowerPC add immediate (addi) instruction

- All PPC instructions have 6-bit opcode field
- addi is a D-Form instruction

| 0 | 14 | RT | 6 | RA | 11 | SI | 16 | 31 |

**Constraints**

- CoGenT classes allow constraints on members
- For parsing, a constraint specifies an expected value
- For emitting, a constraint specifies a ‘magic’ constant

**Example, continued**

```c
class ppc {
    unsigned big bit[32] ppc_inst;
    unsigned big bit[6] OPCD @ ppc-inst[0];
}

class d-form extends ppc {
    unsigned bit bit[5] RA @ ppc_inst[11];
}

class addi extends d-form {
    unsigned big bit[5] RT @ ppc_inst[6];
    unsigned big bit[16] SI @ ppc_inst[16];
}
```
**Semantics**

- Instructions also have semantics
- Semantics are class methods
- Use C-style operators
- Additional, useful operators added (sign-extend, etc.)

**Mixins**

- Allows overloading without actually overloading
- Convenient for orthogonal instruction groups
- For instance, `getaddr()`

```plaintext
mixin getAddrByWord requires (disp) {
    getaddr(disp) { return (bit[32])disp<<2; }
}
mixin getAddrByByte requires (disp) {
    getaddr(disp) { return (bit[32])disp; }
}
```

**Status + Goals**

- Base language spec. stable (we even have a manual!)
- Prototype parser implementation nearly complete
- Front-end analysis in progress
- Microarchitecture description spec in progress