Automatic Generation of LLVM Backends from LISA Using Processor Designer

Jeroen Dobbelaere, Synopsys, Inc.

SYNOPSYS®
Accelerating Innovation

LISA: Language for Instruction Set Architectures

```
RESOURCE {
  // General Purpose Registers
  REGISTER unsigned bit[32]  R[0..31];
  PIN OUT bool      pHALTED;
  unsigned bit[5]   signal_bp_idx_src0_dc;

  MemR1 prog_mem {
    ADDR_WIDTH(20);
    DATA_WIDTH(32);
    RD_LATENCY(1);
  };

  PIPELINE pipe={ PF ; FE; DC ; EX ; MEM; WB };
  PIPELINE_REGISTER IN pipe {
    unsigned bit[5] alu_op;
    unsigned bit[5] bp_idx_dst0;
    uint32 operand_src1;
    …
  }
}

OPERATION add_i IN pipe.DC {
  DECLARE {
    GROUP dst, lhs = { reg };
    GROUP rhs = { uimm16 };
    INSTANCE bypass_src0, alu_ex;
  }
  CODING { 0b001001 dst lhs rhs }
  SYNTAX { "add" ~" " dst ~"," lhs ~"," rhs }

  DOCUMENTATION {R[dst]=
                 R[lhs]+uimm16}}
  BEHAVIOR {
    OUT.alu_op = ALU_OP_ADD;
    OUT.bp_idx_dst0 = dst;
    OUT.operand_src1 =
            ZERO_EXTEND_16(rhs);
    signal_bp_idx_src0_dc = lhs;
  }
  ACTIVATION { bypass_src0, alu_ex }

  LLVM MATCH ADDI16 { // MachineInstruction ADDI16
    CODE { set $dst, (add $lhs, $rhs) }
  }
}

REGISTER reg {
  DECLARE { LABEL idx; }
  SYNTAX  { "r" ~idx   }
  CODING  { idx=0bx[5] }
  EXPRESSION { idx }
  DOCUMENTATION { General Purpose
                  Register [0..31] }
  LLVM { REGISTER IntRegs; }
}

IMMEDIATE uimm16 {
  DECLARE { LABEL imm; }
  CODING  { imm = 0bx[16] }
  SYNTAX  { SYMBOL(imm=#U16) }
  EXPRESSION { imm }
  DOCUMENTATION {Unsigned 16-bit Immediate}
  LLVM { IMMEDIATE imm16, immZExt16; }
}
```

Resources describe:
- processor state
- internal signals
- in/out pins
- memories and protocols

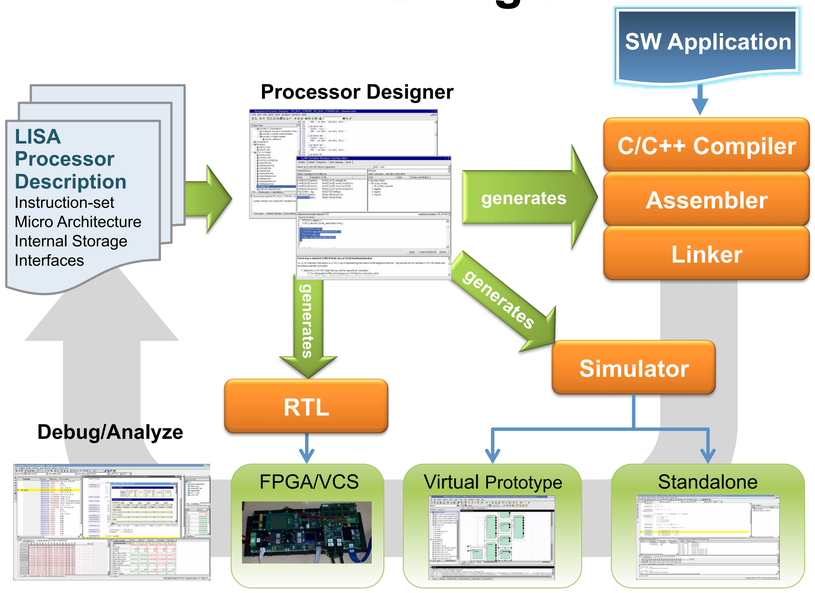
A PIPELINE consists of stages, pipeline registers, and operations that are associated with a stage of the pipeline.The behavior of an instruction can be spread across multiple stages and pipelines.

The encoding and assembly syntax of the instructions.They are used for assembler, disassembler, simulator, compiler and RTL.

The LLVM MATCH annotation describes the LLVM-IR behavior of the instruction.

Other properties (register class, immediate constraints, predication, assembly syntax, ...) are deduced from the information available in the operation hierarchy.

Processor Designer



More details: www.synopsys.com/pd

Custom scheduler takes care of hazards and multiple delay slots

Example:
- A load (`lw`) has its result available after 5 cycles
- A branch (`jmp`) has 3 delay slots

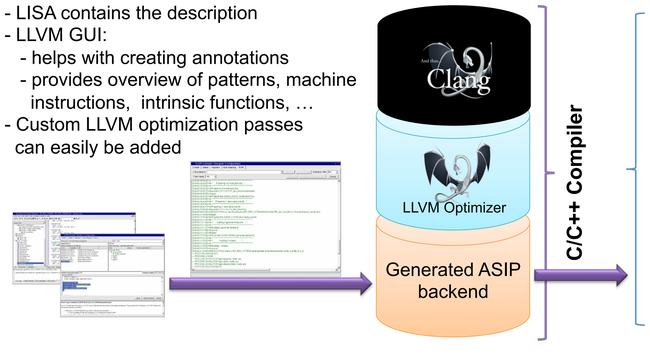
```
extern int table[];

int partialAdd(int* table, int a1, int a2)
{
  return table[3]+table[5]+a1+a2;
}
```

Results in:

```
 .text
        .globl   _partialAdd
        .type    _partialAdd,@function
_partialAdd:
  (0)   lw  R3 , R0 , 12
  (1)   lw  R0 , R0 , 20
  (2)   nop                    <- hazard resolution
  (3)   jmp  r31
  (4)   add  R1, R2, R1
  (5)   add  R1, R1, R3        <- 3 delay slots
  (6)   add  R0, R1, R0
_tmp0:
        .size    _partialAdd, _tmp0-_partialAdd
```

LLVM Compiler Generation Flow

- LISA contains the description
- LLVM GUI:
  - helps with creating annotations
  - provides overview of patterns, machine instructions, intrinsic functions, …
- Custom LLVM optimization passes can easily be added



- Based on LISA description, the backend is generated
- Generated backend is embedded in Clang/LLVM
- Compiler is built for the new target…
- …and is used to build the C/C++ and support libraries.
- Also from LISA, the cycle-accurate simulator and other software tools are built automatically
- Sanity tests are performed to verify that LLVM annotations and instruction behavior are in sync
- Benchmarks are run on the cycle-accurate simulator to verify the effect of changes in the architecture and/or LLVM annotations
- Fast round-trip time allows testing of multiple variants per hour

ASIP* Characteristics:

- Open pipelines with hazards and multiple delay slots.
- Non-standard register size (eg. 20bit)
- Non-byte memory addressing
- Predication: symmetric and asymmetric
- SIMD, VLIW, floating point, fixed point
- Multiple memory spaces

Application Specific Instruction-set Processor