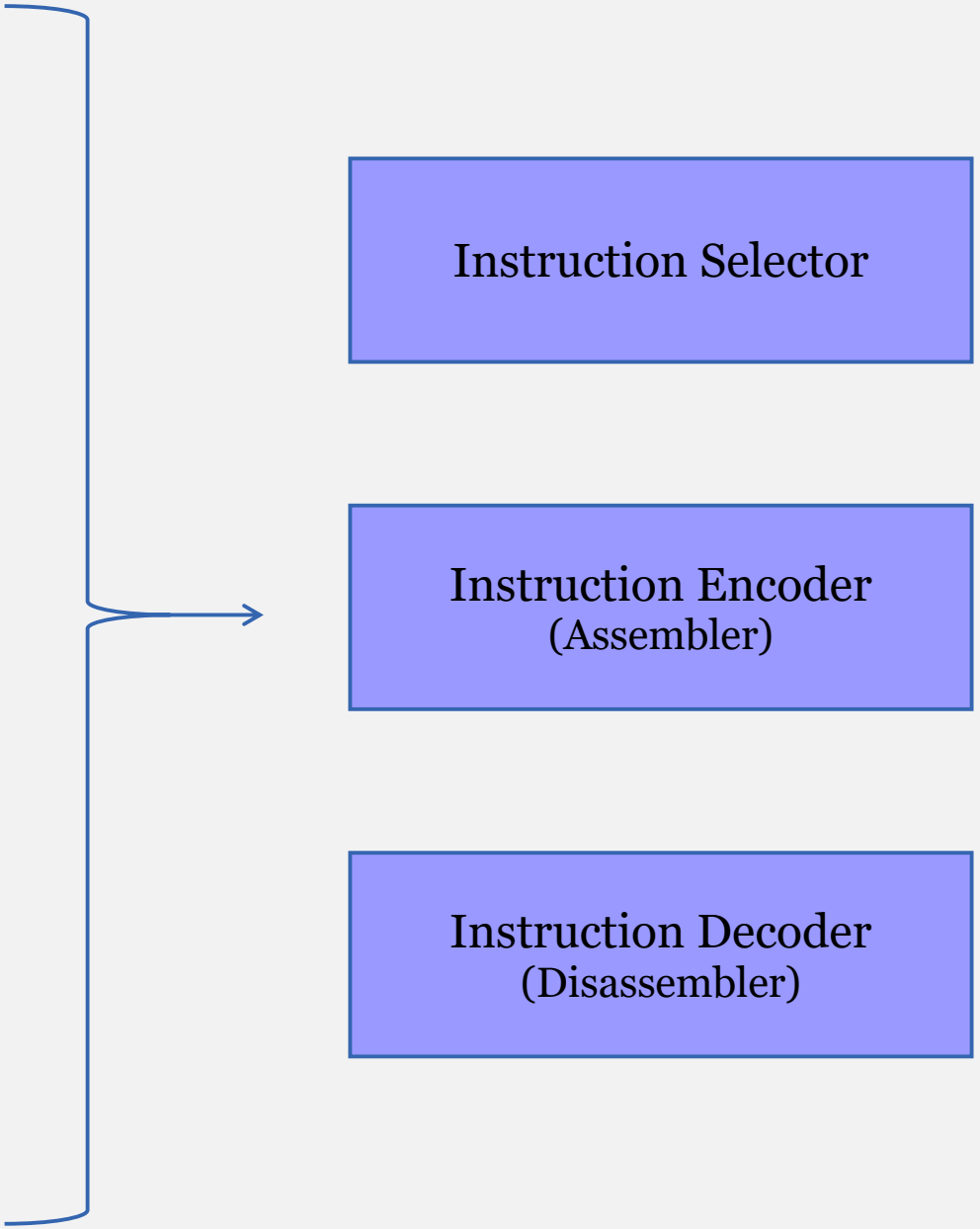
EMBECOSM®

AAPSim

Building a simulator using LLVM MC

LLVM: Insns, Encoding, Decoding

```
class Inst_rrr<bits<4> opclass, bits<8> opcode,
               string asmstr>
    : InstAAP<opclass, opcode asmstr> {
  field bits<32> Inst;
  let Inst{8-6} = rD{2-0};
  let Inst{24-22} = rD{5-3};
  let Inst{5-3} = rA{2-0};
  let Inst{21-19} = rA{5-3};
  let Inst{2-0} = rB{2-0};
  let Inst{18-16} = rB{5-3};
}

multiclass ALU_r<bits<8> opcode, string opname,
                 SDNode OpNode> {
  def _r : Inst_rrr <0x1, opcode,
      !strconcat(opname, "\t$rD, $rA, $rB")>;
}

defm ADD : ALU_r<0x1, "add", add>;
```



EMBECOSM®

Using LLVM as a Simulator

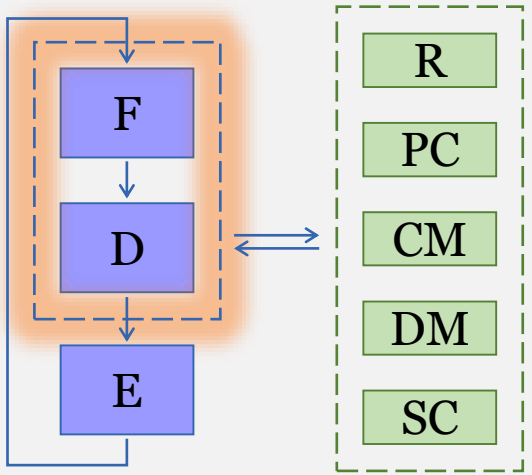
- Need a simulator to run execution tests
- Early in project, instruction set under development
  - Edit `AAPInstrInfo.td`, update compiler assembler *and simulator* at once
- TableGen generates disassembler for us to use
  - also MCInst is a useful container for our simulator

Simulator State

```
class AAPSimState {

  // Registers

  uint16_t base_regs[64];

  uint32_t pc_w : 24;


  // Special registers

  uint16_t exitcode;      // Exit code register

  uint16_t overflow : 1;  // Overflow bit register

  SimStatus status;       // Simulator status


  // One code and data namespace each

  // LLVM expectes 8-bit addressed memory, so
  // provide as such

  uint8_t *code_memory;

  uint8_t *data_memory;

};
```

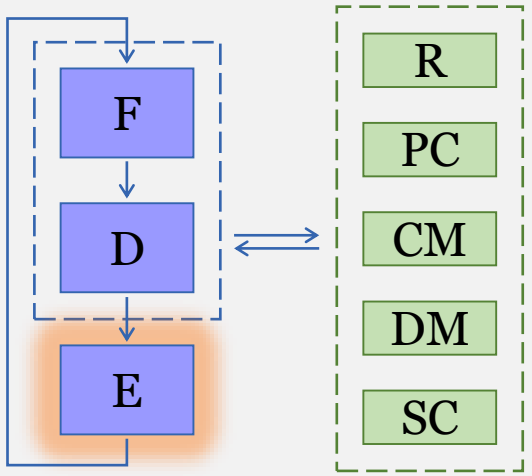
Fetch/Decode

```
SimStatus AAPSimulator::step () {
  MCInst Inst;   uint64_t Size;
  uint32_t pc_w = State.getPC();
  ArrayRef<uint8_t> *Bytes = State.getCodeArray();

  if (DisAsm->getInstruction(Inst, Size, Bytes->slice(pc_w << 1),
                             (pc_w << 1), nulls(), nulls())) {
    // Decode was successful, calculate default new PC
    uint32_t newpc_w = pc_w + (Size >> 1);
    SimStatus status;
    // TODO Execute instruction
    State.setPC(newpc_w);
    return status;
  }
  else {
    // Unable to read/decode an instruction. If the memory raised an
    // exception, pass this on, otherwise return invalid instruction.
    if (State.getStatus() != SimStatus::SIM_OK)
      return SimStatus::SIM_INVALID_INSN;
    return State.getStatus();
  }
}
```



Execution

```
SimStatus AAPSimulator::exec (MCInst &Inst, uint32_t &newpc_w) {
  switch (Inst.getOpcode()) {
    // ... other cases not shown ...
    case AAP::ADD_r:
    case AAP::ADD_r_short: {
      uint32_t ValA = getRegister (Inst.getOperand (1));
      uint32_t ValB = getRegister (Inst.getOperand (2));
      uint32_t Result = ValA + ValB;
      State.setReg (RegDst, static_cast<uint16_t> (Result));
      // Test for overflow
      int32_t Res_s = static_cast<int32_t> (Res);
      State.setOverflow (static_cast<int16_t> (Res_s) != Res_s ? 1 : 0);
      break;
    }
  } // end opcode switch
  return SimStatus::SIM_OK;
}
```



AAP::ADD_r

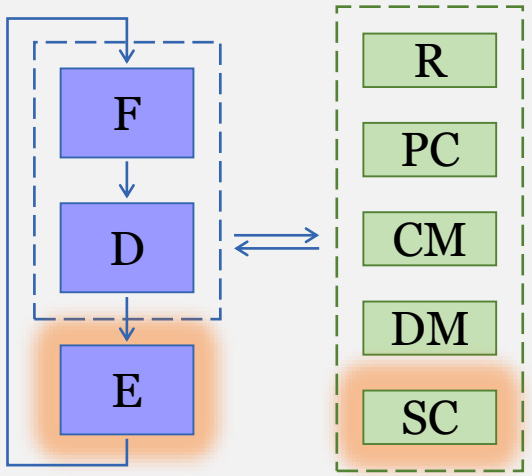
1	0	0	0	0	0	1	d_2	d_1	d_0	a_2	a_1	a_0	b_2	b_1	b
0	0	0	0	0	0	0	d_5	d_4	d_3	a_5	a_4	a_3	b_5	b_4	b_3

AAP::ADD_r_short

0	0	0	0	0	0	1	d_2	d_1	d_0	a_2	a_1	a_0	b_2	b_1	b_0

Simulator I/O

```
switch (Inst.getOpcode ()) {
   // NOP Handling - other cases not shown
   case AAP::NOP:
   case AAP::NOP_short: {
     int Reg = getRegister (Inst.getOperand (0));
     uint16_t Command = Inst.getOperand (1);
     char c = static_cast<char>(Reg);
     switch (Command) {
       case 0: return SimStatus::SIM_BREAKPOINT;
       case 1: break;
       case 2:
         State.setExitCode (Reg);
         return SimStatus::SIM_QUIT;
       case 3:
         outs () << c;
         break;
     break;
   }
 }
```



Simulator Future

- Use tablegen more
  - Use Instruction DAG patters to generate execute stage?
  - Use Scheduling information to generate pipelined simulators?
- Generalize AAPSimulator
  - Would a MCSimulator library be useful?

Demo

embecosm.com