Tutorial: My LLVM backend Book Overview

Reference:
http://jonathan2251.github.com/lbd/index.html

陳鍾樞

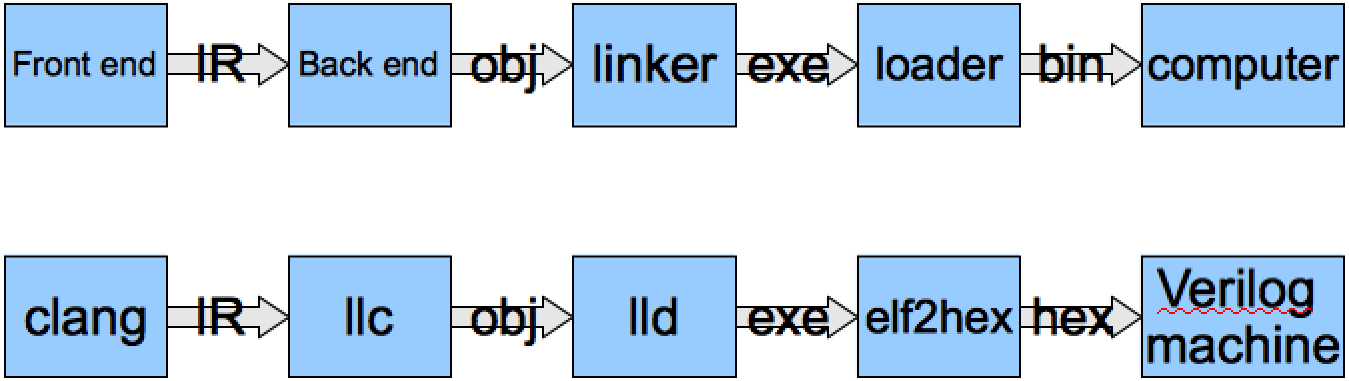
Agenda

- Motivation
- Book contents
- Introduce Chapters 2 and 3 (Not quite details)
- LLVM Backend document reference
- Cpu0 reference
- LLVM document
- Q & A

Motivation

- LLVM has excellent material in front end document but **NO good** document in backend.
- I learn backend by implement the LLVM backend code for Cpu0 which designed from my brother work for teaching purpose.
- I learn backend through writing this tutorial since the LLVM backend complexity.
- I join open source with this book.

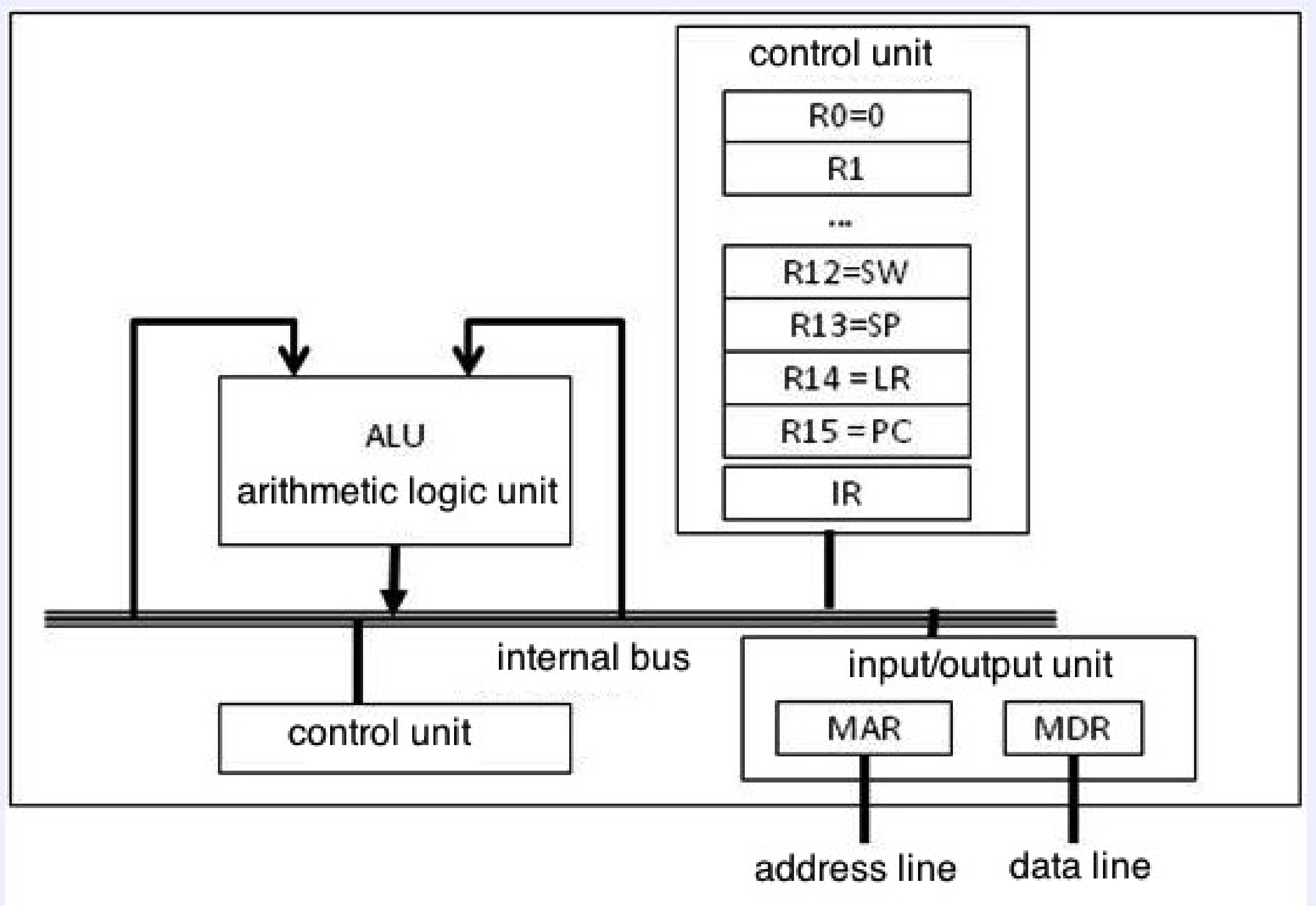
What I do


Front end
IR
Back end
obj
linker
exe
loader
bin
computer
clang
IR
llc
obj
lld
exe
elf2hex
hex
Verilog machine

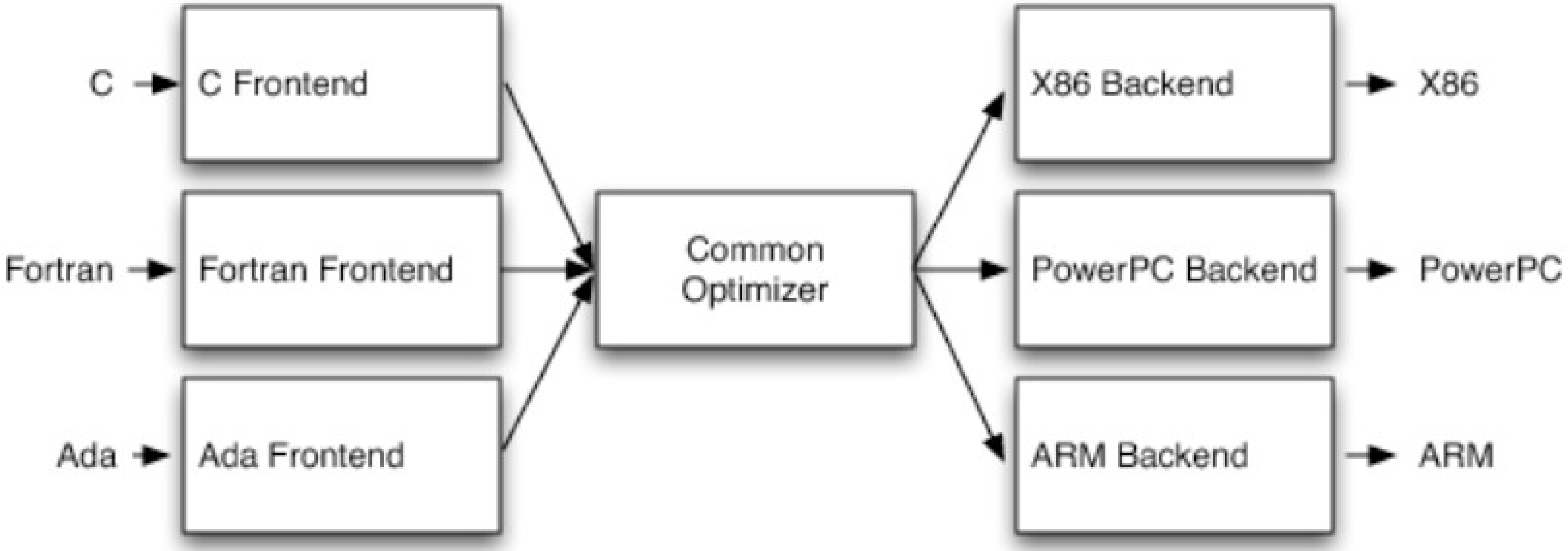
My book contents – Overview

- Chapter 1: Preface and outline of chapters
- Chapter 2: Cpu0 instruction set and LLVM structure
- Chapter 3 – 9: Cpu0 mathematic instructions, Asm, Obj(ELF), Global variables, int, struct and arrays, other type, if else, while, for loop, function call
- Chapter 10: ELF introduction and llvm-objdump -d support.
- Chapter 11: AsmParser support.
- Chapter 12: Verilog machine for Cpu0.
- Chapter 13: lld linker for Cpu0.
- Appendix A: LLVM source code and tools installation.

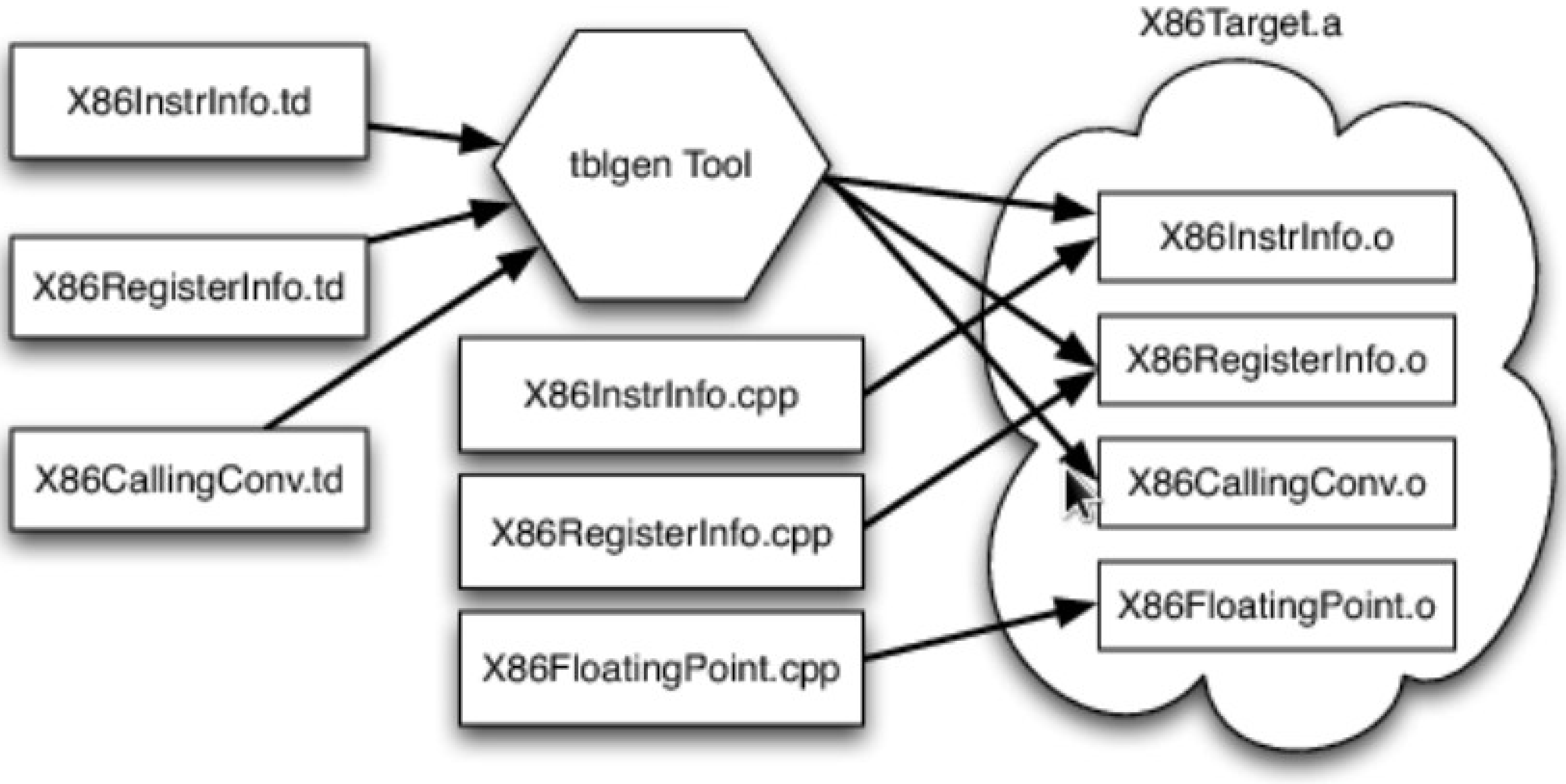
Ch 2 – Cpu0 arch.



Ch 2 – LLVM structure -- 3 tier



Ch 2 – .td file – target description



X86InstrInfo.td

X86RegisterInfo.td

X86CallingConv.td

tblgen Tool

X86InstrInfo.cpp

X86RegisterInfo.cpp

X86FloatingPoint.cpp

X86Target.a

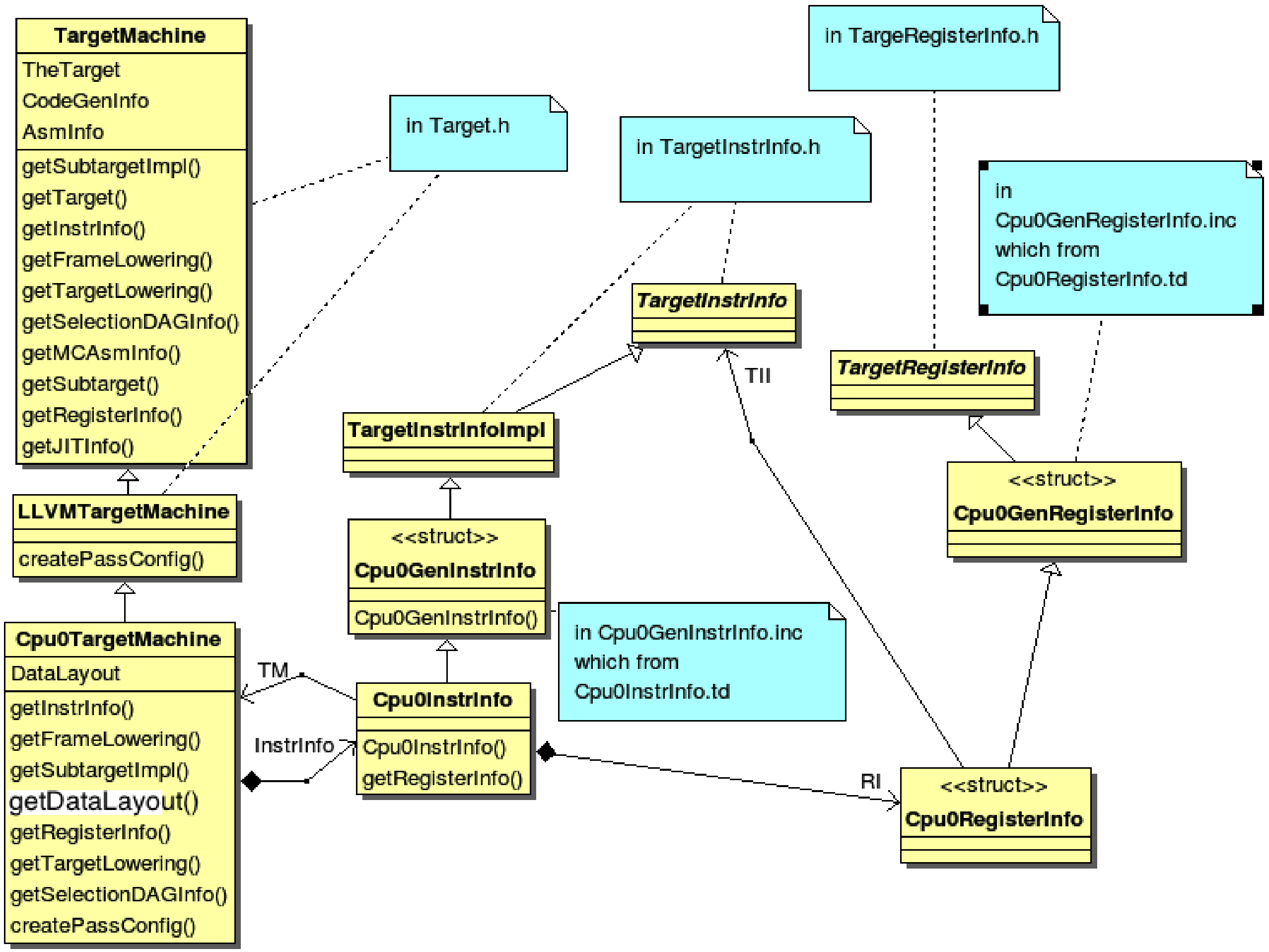
X86InstrInfo.o

X86RegisterInfo.o

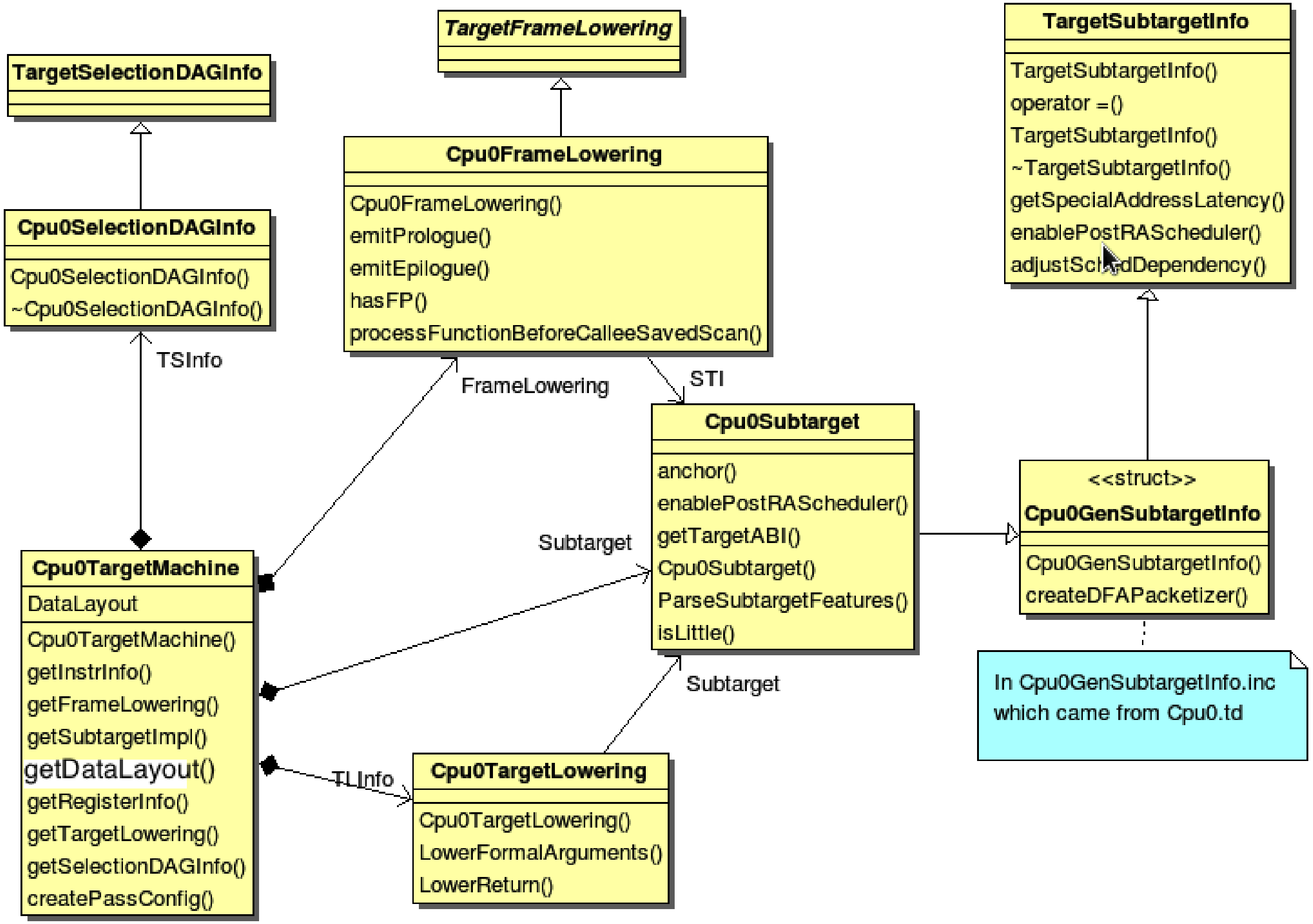
X86CallingConv.o

X86FloatingPoint.o

Ch 3 Backend structure – Cpu0TargetMachine

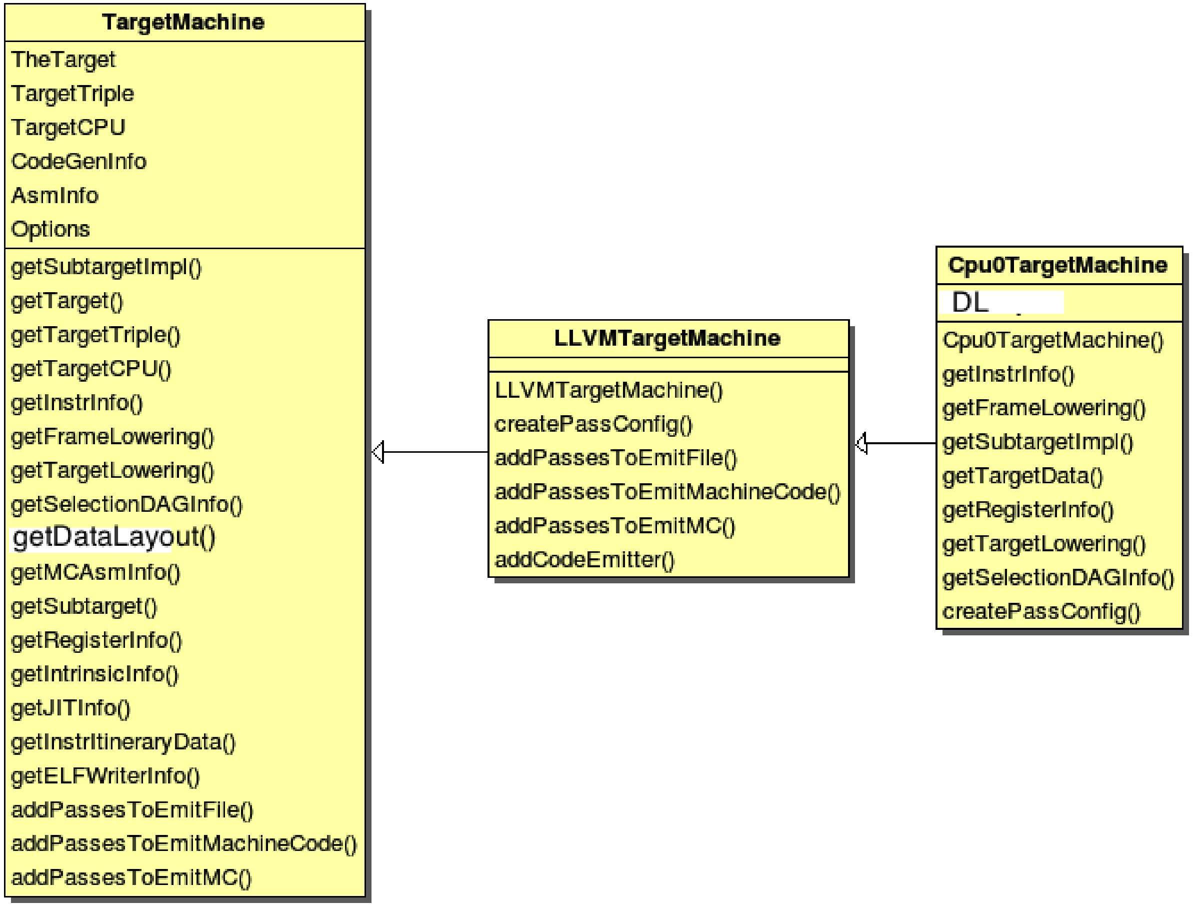


Ch 3 – Cpu0TargetMachine



In Cpu0GenSubtargetInfo.inc
which came from Cpu0.td

Ch 3 – TargetMachine members and operators



TargetMachine

TheTarget
TargetTriple
TargetCPU
CodeGenInfo
AsmInfo
Options

getSubtargetImpl()
getTarget()
getTargetTriple()
getTargetCPU()
getInstrInfo()
getFrameLowering()
getTargetLowering()
getSelectionDAGInfo()
getDataLayout()
getMCAsmInfo()
getSubtarget()
getRegisterInfo()
getIntrinsicInfo()
getJITInfo()
getInstrItineraryData()
getELFWriterInfo()
addPassesToEmitFile()
addPassesToEmitMachineCode()
addPassesToEmitMC()

LLVMTargetMachine

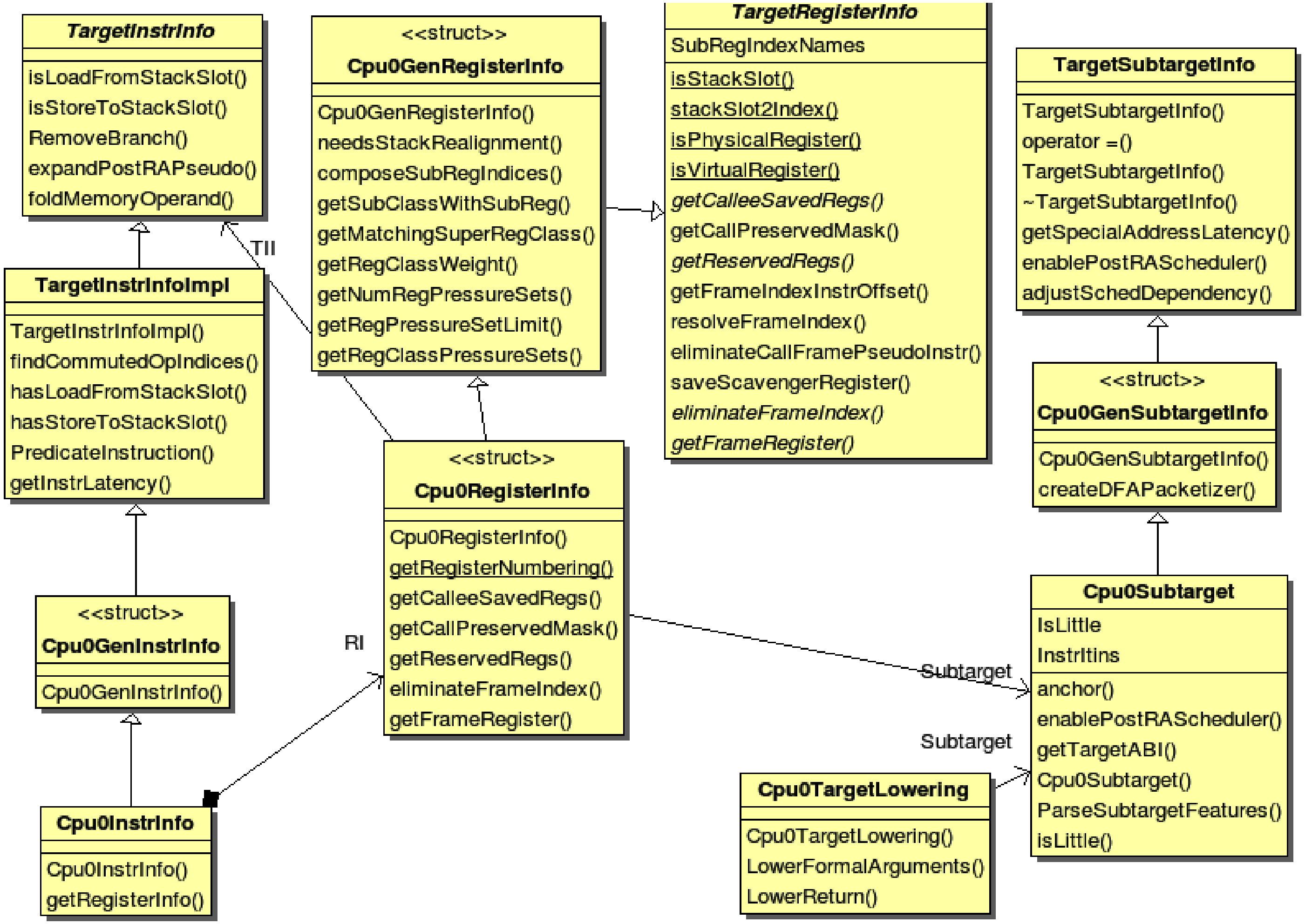
LLVMTargetMachine()
createPassConfig()
addPassesToEmitFile()
addPassesToEmitMachineCode()
addPassesToEmitMC()
addCodeEmitter()

Cpu0TargetMachine

DL

Cpu0TargetMachine()
getInstrInfo()
getFrameLowering()
getSubtargetImpl()
getTargetData()
getRegisterInfo()
getTargetLowering()
getSelectionDAGInfo()
createPassConfig()

Ch 3 – Other class members and operators



TargetInstrInfo
isLoadFromStackSlot()
isStoreToStackSlot()
RemoveBranch()
expandPostRAPseudo()
foldMemoryOperand()

TargetInstrInfoImpl
TargetInstrInfoImpl()
findCommutedOpIndices()
hasLoadFromStackSlot()
hasStoreToStackSlot()
PredicateInstruction()
getInstrLatency()

<<struct>>
Cpu0GenInstrInfo
Cpu0GenInstrInfo()

Cpu0InstrInfo
Cpu0InstrInfo()
getRegisterInfo()

<<struct>>
Cpu0GenRegisterInfo
Cpu0GenRegisterInfo()
needsStackRealignment()
composeSubRegIndices()
getSubClassWithSubReg()
getMatchingSuperRegClass()
getRegClassWeight()
getNumRegPressureSets()
getRegPressureSetLimit()
getRegClassPressureSets()

<<struct>>
Cpu0RegisterInfo
Cpu0RegisterInfo()
getRegisterNumbering()
getCalleeSavedRegs()
getCallPreservedMask()
getReservedRegs()
eliminateFrameIndex()
getFrameRegister()

TargetRegisterInfo
SubRegIndexNames
isStackSlot()
stackSlot2Index()
isPhysicalRegister()
isVirtualRegister()
getCalleeSavedRegs()
getCallPreservedMask()
getReservedRegs()
getFrameIndexInstrOffset()
resolveFrameIndex()
eliminateCallFramePseudoInstr()
saveScavengerRegister()
eliminateFrameIndex()
getFrameRegister()

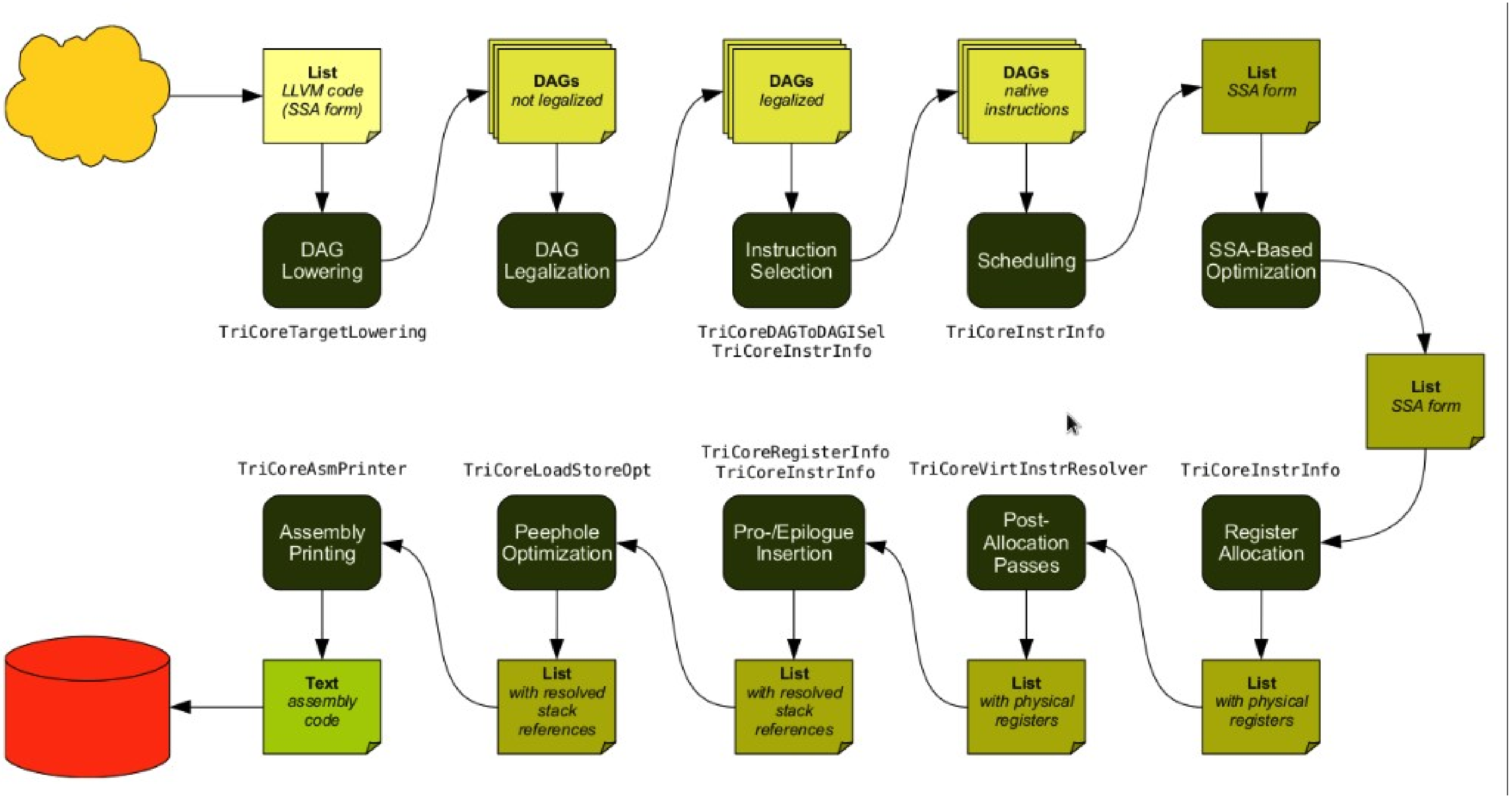
TargetSubtargetInfo
TargetSubtargetInfo()
operator =()
TargetSubtargetInfo()
~TargetSubtargetInfo()
getSpecialAddressLatency()
enablePostRAScheduler()
adjustSchedDependency()

<<struct>>
Cpu0GenSubtargetInfo
Cpu0GenSubtargetInfo()
createDFAPacketizer()

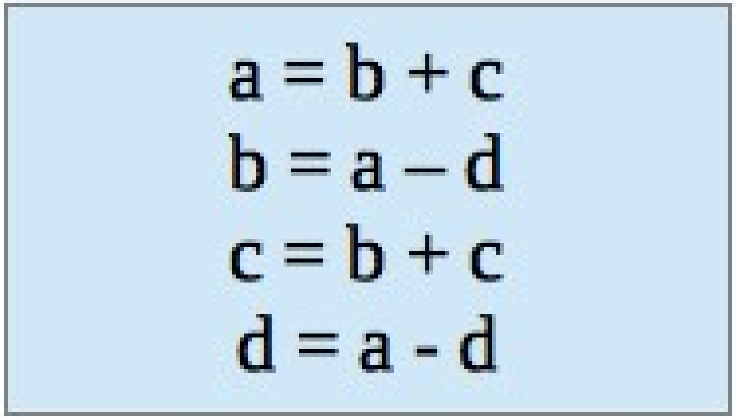
Cpu0Subtarget
IsLittle
InstrItins
anchor()
enablePostRAScheduler()
getTargetABI()
Cpu0Subtarget()
ParseSubtargetFeatures()
isLittle()

Cpu0TargetLowering
Cpu0TargetLowering()
LowerFormalArguments()
LowerReturn()

Ch 3 – Code generation sequence – Copied from tricore_llvm.pdf

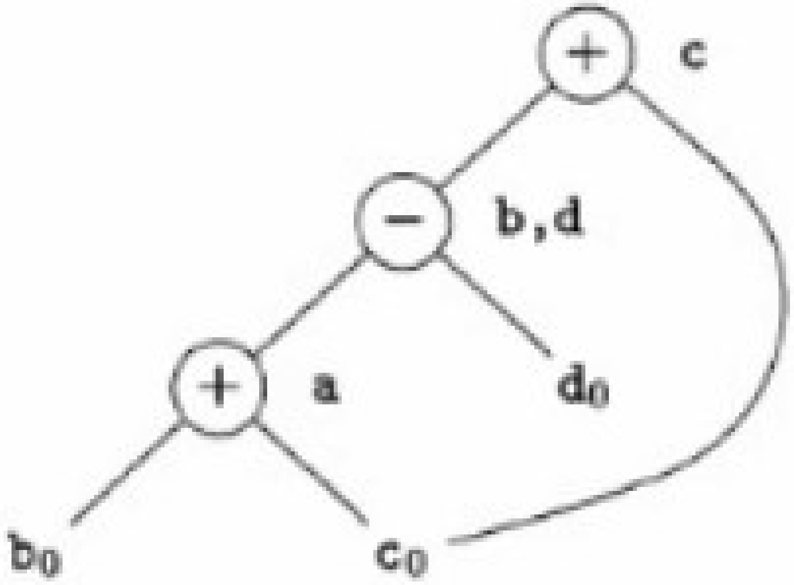


Ch 3 – DAG (Directed Acyclic Graph) example

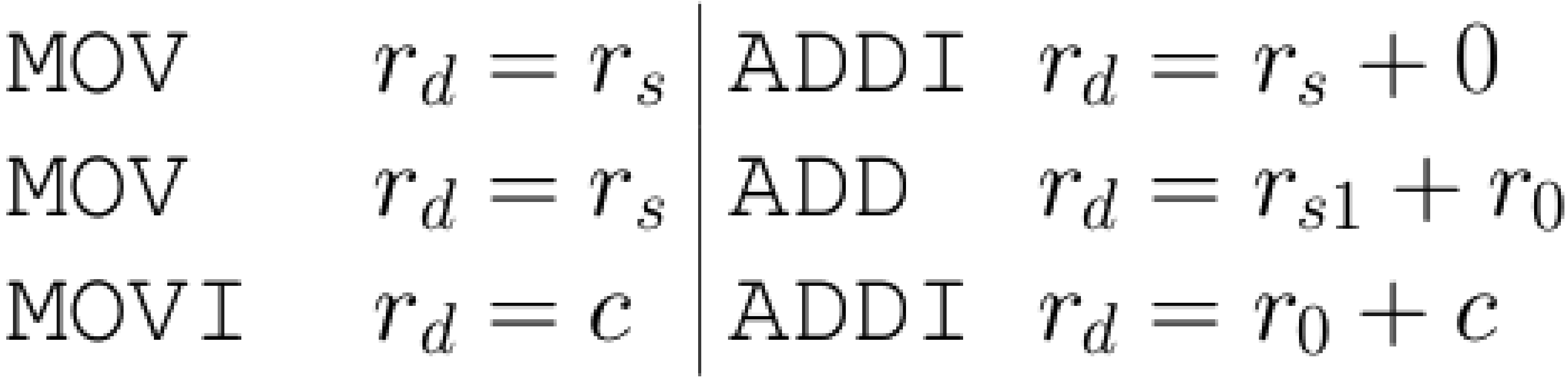


a = b + c
b = a – d
c = b + c
d = a - d

a = b + c
d = a – d
c = d + c



Ch 3 – IR and it’s corresponding machine instruction

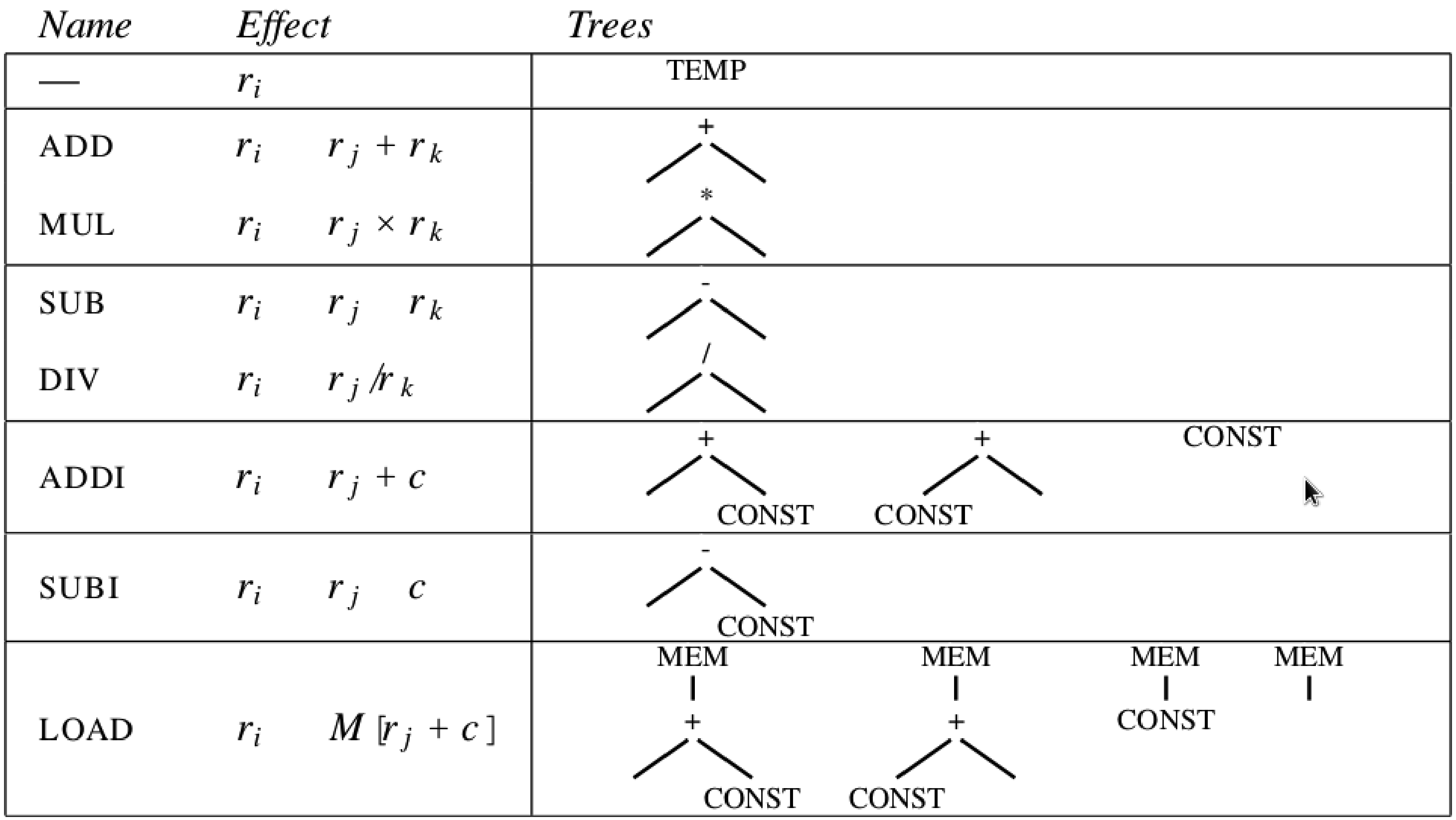


MOV	$r_d = r_s$	ADDI	$r_d = r_s + 0$
MOV	$r_d = r_s$	ADD	$r_d = r_{s1} + r_0$
MOVI	$r_d = c$	ADDI	$r_d = r_0 + c$

MOV rd, rs | ADDI rd, rs, 0

Ch 3 – Instruction DAG representation

Instruction Tree Patterns



Name	*Effect*	*Trees*
—	r_i	TEMP
ADD	$r_i \quad r_j + r_k$	+
MUL	$r_i \quad r_j \times r_k$	*
SUB	$r_i \quad r_j \quad r_k$	-
DIV	$r_i \quad r_j / r_k$	/
ADDI	$r_i \quad r_j + c$	+ CONST + CONST CONST
SUBI	$r_i \quad r_j \quad c$	- CONST
LOAD	$r_i \quad M[r_j + c]$	MEM + CONST MEM + CONST MEM CONST MEM

File *.td.

- *.td is Target Description.
- Role: pattern for translate LLVM IR (Immediate Representation Code) DAG into Machine Instructions
- As compiler book, pattern search for Code Generation. But more, it's a description language for pattern search and Instructions generation.

File *.td. – Example

```
class Cpu0Inst<dag outs, dag ins, string asmstr, list<dag> pattern,
               InstrItinClass itin, Format f>: Instruction
{
  field bits<32> Inst;
  Format Form = f;
  let Namespace = "Cpu0";
  let Size = 4;
  bits<8> Opcode = 0;
  // Top 8 bits are the 'opcode' field
  let Inst{31-24} = Opcode;
  let OutOperandList = outs;
  let InOperandList  = ins;
  let AsmString   = asmstr;
  let Pattern     = pattern;
  let Itinerary   = itin;
  // Attributes specific to Cpu0 instructions...
  bits<4> FormBits = Form.Value;
  // TSFlags layout should be kept in sync with Cpu0InstrInfo.h.
  let TSFlags{3-0}   = FormBits;
  let DecoderNamespace = "Cpu0";
  field bits<32> SoftFail = 0;
}
```

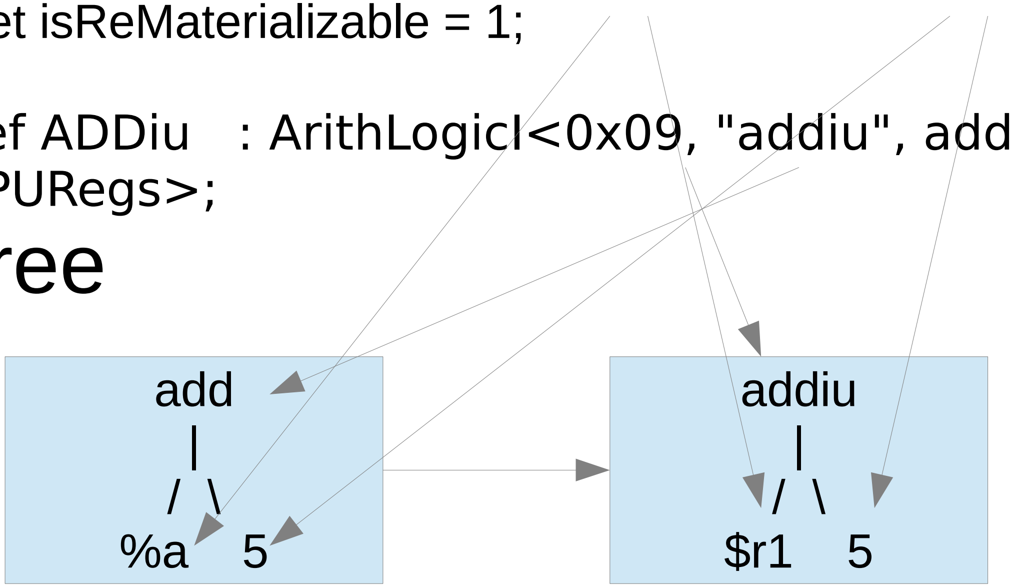
File *.td. – Example – cont.

```
class FL<bits<8> op, dag outs, dag ins, string asmstr, list<dag> pattern,
         InstrItinClass itin>: Cpu0Inst<outs, ins, asmstr, pattern, itin, FrmL>
{
  bits<4>  ra;
  bits<4>  rb;
  bits<16> imm16;
  let Opcode = op;
  let Inst{23-20} = ra;
  let Inst{19-16} = rb;
  let Inst{15-0}  = imm16;
}
// Arithmetic and logical instructions with 2 register operands.
class ArithLogicI<bits<8> op, string instr_asm, SDNode OpNode,
                  Operand Od, PatLeaf imm_type, RegisterClass RC> :
  FL<op, (outs RC:$ra), (ins RC:$rb, Od:$imm16),
     !strconcat(instr_asm, "\t$ra, $rb, $imm16"),
     [(set RC:$ra, (OpNode RC:$rb, imm_type:$imm16))], IIAlu> {
  let isReMaterializable = 1;
}
def ADDiu   : ArithLogicI<0x09, "addiu", add, simm16, immSExt16,
CPURegs>;
```

Instruction Selection/Generation

```
class ArithLogicI<bits<8> op, string instr_asm, SDNode OpNode,
                  Operand Od, PatLeaf imm_type, RegisterClass RC> :
  FL<op, (outs RC:$ra), (ins RC:$rb, Od:$imm16),
     !strconcat(instr_asm, "\t$ra, $rb, $imm16"),
     [(set RC:$ra, (OpNode RC:$rb, imm_type:$imm16))], IIAlu> {
  let isReMaterializable = 1;
}
def ADDiu   : ArithLogicI<0x09, "addiu", add, simm16, immSExt16,
CPURegs>;
```

- Tree



```
add
 |
/ \
%a   5
```

```
addiu
  |
 / \
$r1   5
```

- List
  - (add %a, 5) → (addiu $r1, 5)

Tutorial: Creating an LLVM Backend for the Cpu0 Architecture

- Over 600 pages of pdf book include full example code telling you how to write a llvm backend(llc), lld, elf2hex and verilog for Cpu0.
- This is a professional book in compiler implementation, not just a concept of book
- http://jonathan2251.github.com/lbd/index.html