Questions on the LLVM backend for a new architecture

Instruction:

This document contains materials about what issues have been confronted on the way to get the exclusive LLVM backend emitting text assembly code for a new architecture XCC.

First, all the XCC codes are being made based on SPARC LLVM backend code (with the prefix Sparc~.*) which was already offered by LLVM developers group. This modification seems to be much easier for me to compose the codes for XCC assembly than starting from a blank sheet. Anyway, I became to get some issues about the SPARC LLVM code in connection with XCC.

Of course, LLVM developers group also offers other kinds of LLVM backend for ARM, Alpha, IA64, X86 and others. But SPARC code looks like something more stabilized because ,for example, I found that LLVM code for ARM makes error even for the simplest source code while spitting out an ARM assembly code. (*I will show you this example later in this document*) But I also referred ARM code while making my code because this is quite simple and short so it made me capture the point more easily.

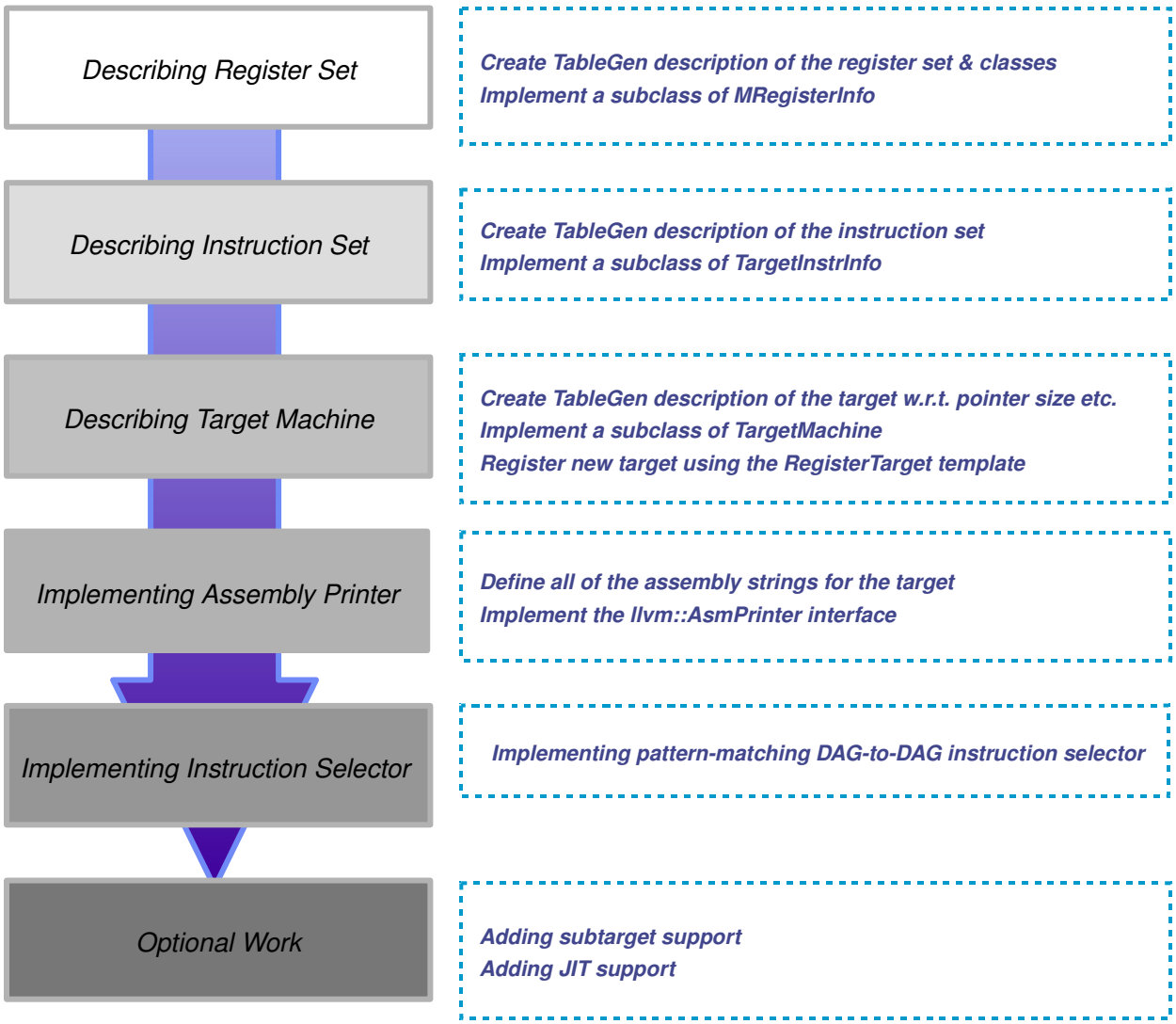
Unfortunately, I do not have such a detailed XCC architecture manual now. All I have is just XCC Target Language (XTL) manual which is good for me to have a big picture but not such a good one for actual programming because it only contains quite conceptual information for XCC.

Anyway, I'd like to enunciate the issues for this backend work as follows:

I marked my questions in bold and blue face.

Progress Chart for LLVM backend:

The chart shown below is what LLVM developers recommended in their website (http://llvm.org/docs/WritingAnLLVMBackend.html). I am following this procedure.



Describing Register Set:

Identifying Registers

In LLVM backend code for SPARC (*see* SparcRegisterInfo.td line 14~32), there are three kinds of registers, that is, integer register, floating-point register, double floating-point register.

In LLVM backend code for ARM (*see* ARMRegisterInfo.td line 15~19), there are only one kind of register, i.e. integer register. I think there might be more registers for ARM. But it seems that LLVM guys defined only the integer register in this code.

(Q) Is this guess of mine correct?

Then, how about XCC? My fellow recommended me to follow the SPARC way, but this is not defined truth yet.

(Q) Do I need to know exact register information for XCC such as what kinds of registers which I have to define in the code are needed, how many bits they need and so on? Or Can I just follow SPARC's way, i.e. 3 kinds of registers defined in LLVM SPARC code.

bits<>

This is also relevant with registers. SPARC code(*see* SparcRegisterInfo.td line 14~32) has been defined to have 32-bit integer register whereas 16-bit integer register for ARM(*see* ARMRegisterInfo.td line 15~19).

Then how about XCC? How should I define the size of integer register for XCC and sizes for other kinds of register, if there are?

(Q) This part somewhat falls on the previous question. Anyway, do I need to know exact information about this from XCC architecture manual which I do not have now? Or Can I just follow SPARC's way?

Describing Instruction Set:

Instruction Format

SPARC has several instruction formats such as Format #2 and Format #3 more than just *InstSP* class which is the basic (*see* SparcInstrFormats.td). However, ARM has only the basic *InstARM* class(*see* ARMInstrInfo.td line 32~38).

Does XCC have any special instruction format like SPARC or not?

(Q) To resolve this problem, do I need to know exact information about

whether XCC needs special instruction format from XCC architecture manual? Or Can I just follow SPARC's or ARM's way?

My fellow ascribed this to the special feature of SPARC having subtargets V8 and V9, this would be also a sort of issues to be confirmed, though.

These special instruction format offers SPARC more steps to have its TableGen definition.

For example,

In ARM code(*see* ARMInstrInfo.td line 48~), definition *ldr* is derived from *InstARM* directly like this:

```
def ldr : InstARM<>.  ;
```

However, in SPARC code (*see* SparcInstrFormats.td and SparcInstrInfo.td line 250~) class *F3* was derived from *InstSP* and class *F3_1* was made based on *F3*, and then *LDSBrr* was derived from *F3_1* in this way:

```
class F3< … > : InstSP< … > { … }
class F3_1<… > : F3< … > { … }
def LDSBrr : F3_1< … >
```

bits<> (in Instruction Format)

If you see class F3_1 which is defined in SparcInstrFormats.td line 77~88, you can see lots of bits<> in it. This type of format was already defined in SparcV8 manual, page 44.

I am still not sure bit information introduced in ARMInstrInfo.td line 23 which mentioned that ARM has plus/minus 12 bit offset is same kind of concept with that of SPARC.

Then what about XCC? Unfortunately, the XTL manual I have doesn't seem to have this kinda information. This should be something I should know to make further progress.

(Q) If XCC needs special instruction formats, this should be in the instructions manual, right?

Instruction implementation

Add instructions defined in SparcV8 manual, page 108 was implemented on LLVM code like this: (*see* SparcInstrInfo.td line 438)

```
def ADDrr   : F3_1<2, 0b000000,
```

```
(ops IntRegs:$dst, IntRegs:$b, IntRegs:$c),
"add $b, $c, $dst",
 [(set IntRegs:$dst, (add IntRegs:$b, IntRegs:$c))]>;
```

(Q) I can't find where 0b000000 came from. Strange number… Please let me know.

Describing Target Machine:

Pointer size

On the way to have an exclusive LLVM backend, I should pass the toll gate named "Creating the target that describes the pointer size" which is not precise to me, either. **(Q) how and where can I define this in the code?**

My fellow guessed it might be the same with the size of registers.

(Q) Is this right?

Implementing Instruction Selector:

Implementing pattern-matching DAG-to-DAG instruction selector

Frankly speaking, this is completely unfamiliar with me. What is this for? In the LLVM website, developer guys recommended this to be used as an instruction selector.

I need to see http://llvm.org/docs/CodeGenerator.html.

And Others:

Difference in pseudo instructions between XCC and SPARC etc.

LLVM SPARC code (*see* SparcInstrInfo.td line 174 ~ 239) offers those for pseudo instructions.

```
// Pseudo instructions.
class Pseudo<dag ops, string asmstr, list<dag> pattern>
  : InstSP<ops, asmstr, pattern>;
def ADJCALLSTACKDOWN : Pseudo<(ops i32imm:$amt),
                        "!ADJCALLSTACKDOWN $amt",
                        [(callseq_start imm:$amt)]>;
```

(Q) My issue is whether I can use this code which is now implemented for SPARC assembly code in order to compose XCC pseudo instructions through modification. Right?

"llc" *error* when generating ARM assembly code

This is what I mentioned above (at the paragraph for instruction) briefly. "llc" is a command for compiling LLVM bytecode into target assembly code. (*see* http://llvm.org/docs/CommandGuide/html/llc.html)

To use this command, I made the simplest C code "hello, world" at first as the LLVM website recommended.

```
#include <stdio.h>
  int main() {
    printf("hello world\n");
    return 0;
  }
```

And then, I complied this code into a LLVM bytecode file through "$ llvm-gcc hello.c -o hello". This procedure makes me have hello.bc which is the corresponding LLVM bytecode.

Finally, I tried to convert this code to ARM assembly code through "$ llc -march=arm hello.bc -o hello.arm" HOWEVER, this makes erros as follows:

```
llc: ARMISelDAGToDAG.cpp:73: llvm::SDOperand LowerCALL(llvm::SDOperand,
llvm::SelectionDAG&): Assertion `isVarArg == false && "VarArg not supported"'
failed.
llc((anonymous namespace)::PrintStackTrace()+0x15)[0x850437d]
llc((anonymous namespace)::SignalHandler(int)+0x139)[0x8504645]
Aborted
```

(Q) I could make assembly codes for other targets like SPARC, X86, Native assembly … so this error is strange to me. Please let me know the reason.

Instruction Pattern Stuff

The part about "Instruction Pattern Stuff" is shown in SparcInstrInfo.td line 43~. I tried to find this in LLVM website but didn't succeed.

(Q) Would you mind telling me where I can find this information about this?