LLVM Backend for SD-8516

February to April 2026

I started by copying the Lanai files and renaming everything to SD8516. When it compiled I moved forward.

Yes, this took three months.

April and May 2026

During this time I worked on the back-end. There are many tutorials on the 'net such as the CPU0 tutorial. I chose LLVM because I heard the documentation was more useful. Thank God, because I was going to need everything I could get. It's impossible to describe this process. I should have kept more notes on the wiki about it. I may have been able to save myself several days worth of grepping to find random stuff in the target's source tree. This period I call the insanity days. 4 and 8 hour days of doing nothing but trying to lower LLVM code into SDA assembly.

This period is binary. It compiles or it does not. There is no incremental improvement here. I learned a lot and actually changed my ISA a bit because of what I learned doing this. But, I do not remember anything I did here. It is a black hole. I've probably blocked it out of my memory. It was that horrifying.

Notes from April/May

- That stupid string that looks like e-p:32:8-i16:8-i32:8-i64:8-n16 is in:
    - `llvm/lib/TargetParser/TargetDataLayout.cpp`

Total cost of this note alone: Probably 6 hours.

- COMMON files I need to find:
    - `llvm/lib/Target/SD8516/SD8516InstrFormats.`
    - `llvm/lib/Target/SD8516/SD8516InstrInfo.`
    - `llvm/lib/Target/SD8516/SD8516ISelLowering.`

Common advice:

- Look very carefully at the error message. 90% of the time you're missing a header, a function declaration, or something's type is wrong. The solution is often easier than you think.
- Use git or you will mess stuff up and be unable to go back.

June 2026

A sudden thrust of energy, cola and Doritos and the back-end was in sight. A lot of this was fueled by

excitement over many small wins. Such as getting something to work. Once I saw it start to emit SDA assembly I was hooked. Many small changes to the assembler were made too, to support LLVM stuff that I didn't want to try and lower out. One example of an extreme change I made was to remove MOV touching the zero flag because I couldn't figure out how to set it in the files. Its supposed to be [SR] (status register). But there are multiple places you're supposed to add it and I started hallucinating that I had added it in certain places then when I went back to check it was gone. Whatever, I'll just remove the flags from MOV! Now i'm done with that part. HAHA. Hehe hoho.

Suddenly I was out of the tunnel and I didn't even realize it. I decided to skip some stuff that would have made the generated code “better”. Like jumptables and some function calls (that are instead inlined, which bloats code).

- Remember: jumptables for large switches
- Remember: mul and div calling functions versus inlining
- Notes to Self from June
- Runtime Library Notes

Truncating i8 stores

Truncating stores for i8– STAL for char.

“If it's legal, keep it legal” says reflect what the hardware actually does and skip the legalizer's splitting work. The reason it's wrong here isn't decisive here specifically is that the Legal path's matcher re-expresses the i8 trunc pattern at line ~278 (which is what byte-arithmetic narrowing relies on) and the GR8 at ~286 (ordinary char stores). So in this one case Legal means maintaining a third pattern that does what two existing ones already do.

In the end it emits STAL [ptr] either way.

Marking things SR

Marking things SR does not protect against all stages of LLVM lowering. Sometimes disparate parts try to insert things between CMP and Bcc/Jcc. The solution is to fuse them in the backend. Ask yourself, does a branch instruction that relies on a CMP EVER *need* anything between the CMP and the branch? If so, since the branch is guaranteed, why not move the code into the branch?

Meeting LLVM halfway

There are a lot of 'easy fixes' such as adding a LEA style instruction, that make it easier to write the backend. That's because LLVM, or at least the Lanai I was copying, use LEA. The fact is, it's a kludge to add an instruction solely to fix something in the backend. But we have that luxury. In the case of LEA, I could have added LD_IDXI16 or LEA and it would have masked the problem, likely for a long time. But the actual fix makes things bulletproof. That is the correct solution. Make LLVM work for the ISA, then change it later for optimization only. And, only if you want to. The more you optimize, the more you shift away from your own vision of the CPU/ISA and towards something generic.

This is a hidden issue regarding C, compilers, and CPUs. The CPU has, for a long time, adapted to C. As C has adapted to the CPU. It's a symbiosis. But C won't last forever and maybe there are better ways to do things. If VAX can ship an insque() maybe insque should be standard?

Not everything is a lowest common denominator. That's bland. C, or some language else, should be able to reflect upon what makes a CPU special and unique. That's part of the joy of assembly; exploring truly new mindscapes of architecture. Realizing why certain ideas work better than others considering the hardware.