W. Duncan Fraser
Christopher Hartman
CS 631
May 6th, 2015

C to ARMv5 Compiler

For my compiler, I created a C to ARMv5 compiler. I specifically chose ARMv5 as the only ARM emulator I could find that was useable and didn't require considerable effort to get up and running, implemented the ARMv5 feature set. The emulator can be found at the following address: http://armsim.cs.uvic.ca/

For my compiler I used Flex and Bison, and implemented an AST using C++ Polymorphism. A default node class acted ass the superclass for all the other grammar members, with multiple layers of inheritance down to the individual terminal classes such as variable assignment, access, and binary operators. This allowed for the use of virtual functions for code generation, which greatly simplified the process. Additional classes were built to keep track of registers and a symbol table, to ensure that the appropriate ARM registers and .data variables were being utilized.

I was able to implement all the mathematical operators available in ARM5, which is a fairly small amount. The complete list of data processing instructions is in the following table. :

Add with Carry. See ADC on page A4-4. Add. See ADD on page A4-6.
Logical AND. See AND on page A4-8.
Logical Bit Clear. See BIC on page A4-12.
Compare Negative. See CMN on page A4-26.
Compare. See CMP on page A4-28. "
Logical EOR. See EOR on page A4-32.
Move. See MOV on page A4-68.
Move Not. See MVN on page A4-82.
Logical OR. See ORR on page A4-84.
Reverse Subtract. See RSB on page A4-115.
Reverse Subtract with Carry. See RSC on page A4-117.
Subtract with Carry. See SBC on page A4-125.
Subtract. See SUB on page A4-208. "
Test Equivalence. See TEQ on page A4-228. Test. See TST on page A4-230.

In addition to the contents of the table, ARMv5 includes Multiply, but no divide.

The incredibly small instruction set became very limited when trying to implement any advanced features, as certain functionality that is taken for granted even in other assembly languages, is just missing and has to be hand rolled.

In addition to the basically binary operators, I implemented variable declaration, assignment, and arbitrary usage within expressions. I was also able to implement basic if statements and while loops. However, the implementation is very basic as the only conditional available is equality due to the limited nature of ARMv5.

I did not manage to finish implementing secondary functions, however the approach I was planning to take was changing my symbol table to a vector so I can iterate through it for scope. In addition, all currently in use registers would be pushed to the stack for retrieval after the function had finished executing. Registers 0-3 would be the primary registers for parameter passing. I did implement the return statement node, which functions within Main(). Return drops the lock on all registers, and moves the output of the function to r0, which is the standard return register.

If I were to change my approach, I would look for a better emulator that allowed me to implement a more modern version of ARM, as newer ARM chipsets are much closer to x86 in terms of built in instructions, with full bitwise operations, division, modulus, etc.

Attached is my final project code, as well as example programs and the corresponding ARMv5 output.