AUTOMATIC ADL-BASED GENERATION OF DISASSEMBLING TOOLS

Alexandre K. I. Mendonça, Felipe G. Carvalho, Max R. O. Schultz, Luiz C. V. Santos, Olinto J. V. Furtado*
Federal University of Santa Catarina
Computer Science Department
Florianópolis – SC – Brazil
{mendonca, fgcarval, max, santos, olinto}@inf.ufsc.br

ABSTRACT

Embedded software evaluation for distinct CPU targets is required during SoC design space exploration. To cope with time-to-market, such exploration asks for automatically retargetable code generation and debugging. Since embedded software debugging tools rely on monitoring object code at the back-end so as to allow source code inspection at the front-end, disassembling tools are needed to bridge both ends. This paper describes the automatic generation of disassembling tools for an arbitrary instruction-set architecture (ISA), formally described in an Architecture Description Language (ADL). Experimental evidences of proper functionality, wide-range retargetability and runtime efficiency are provided for MIPS, SPARC, PowerPC and i8051.

1. INTRODUCTION

SoC development requires the evaluation of embedded software running on alternative target CPUs for the sake of design space exploration. To cope with time-to-market, such exploration asks for automatically retargetable code generation and debugging. On the one hand, assembling and linking tools are needed to generate object code. On the other hand, disassembling and debugging tools are required to monitor object code, thereby supporting software development.

We address the retargetable disassembling problem by automatically generating a disassembler from a formal description of an arbitrary target CPU. Since a debugger employs a disassembler to inspect object code, this work paves the way to achieve automatically retargetable debugging.

This paper is organized as follows. Section 2 briefly describes related work and summarizes our implementation infrastructure. Section 3 describes our technique for automatic generation of disassemblers. Experimental results are summarized in Section 4, whereas our conclusions are drawn in Section 5.

2. RELATED WORK

The formal description of CPUs often relies on Architecture Description Languages (ADLs). ArchC [1] is an example of open-source ADL. Although assembler [2] and linker [3] generation are already available from ArchC, this work was motivated by the lack of a disassembler generator within the ArchC tool chain.

```
1.  ac_format Type_I =
       "%op:6 %rs:5 %rt:5 %imm:16:s
2.  ac_instr<Type_I> sb, sh, sw, swl, swr;
3.  ac_asm_map reg {
4.     "$"[0..31] = [0..31];
5.     "$zero" = 0;
6.     "$at" = 1;
7.     "$kt"[0..1] = [26..27];
8.     "$gp" = 28;
9.     "$sp" = 29;
10.    "$fp" = 30;
11.    "$ra" = 31;
12. }
13. sb.set_asm("sb %reg, %imm(%reg)",
                    rt,   imm, rs);
14. sb.set_decoder(op=0x28);
```

Figure 1 –Example of ArchC description

2.1. ArchC

To explain how our technique extracts information from an ADL description, let's illustrate a few ArchC constructions by means of the example in Figure 1. Line 1 defines the fields of a given format type (`Type_I`). In line 2, instructions `sb`, `sh` and `sw` are tied to the defined format. The mapping between symbolic names and actual CPU registers is performed in lines 3 to 12. Line 13 defines the assembly syntax for instruction `sb`. Line 14 assigns an encoding value to that instruction's opcode field.

2.2. GNU Binutils

Manually retargetable binary utilities are available within the popular GNU Binutils package [4] like assembler (gas), linker (ld), debugger (gdb) and disassembler (objdump). Essentially, the Binutils package consists of an invariant ISA-independent core module and a few ISA-dependent modules that must be rewritten for each new target CPU. Among the ISA-dependent modules, there are two main libraries, namely Opcodes and BFD, which require retargeting.

The Opcodes library describes the ISA of a CPU (instruction encoding, register encoding, assembly syntax). Unfortunately, there is no standard for ISA description within this library.

The BFD library provides a format-independent (ELF, COFF, A.OUT, etc.) object file manipulation interface. It is split into two blocks: a front-end, which is the library's abstract interface with the application and a back-end, which implements that abstract interface for distinct object file formats.

3. DISASSEMBLER GENERATION

Our generation technique relies on the GNU Binutils package as implementation infrastructure. Given the native GNU Binutils disassembling tool (Objdump), its ISA-independent modules are kept untouched and its ISA-dependent modules are automatically generated. The information required for generating the ISA-dependent modules is automatically extracted from the ADL description of the target CPU.

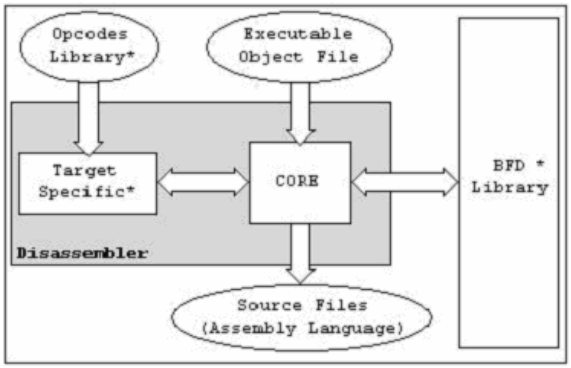

Figure 2: The structure of a disassembling tool

* Supported for the National Program of Microelectronics/CNPq, Proc. nº 132874/2005-9.

The disassembling tool structure is depicted in Figure 2, where the generated modules are marked with an asterisk. The compilation of all such modules results in a disassembling tool that reads an input object file and produces a corresponding output assembly file.

3.1. Opcodes Generation

Let [arch] represent a given ISA. The most important files generated by our tool are the following:

- include/opcodes/[arch].h: This file declares three data structures supporting instruction decoding and encoding, the mapping between register names and encodings and pseudo-instruction manipulation. (It should be noted that disassembling doesn't make use of pseudo-instructions to avoid ambiguity).
- opcodes/[arch]-opc.c: This file contains the above mentioned data structures, which are fed with the information extracted from the ADL description by the ArchC preprocessor.
- opcodes/[arch]-dis.c: This is the most important file for the disassembling process. It manipulates the above mentioned data structures and invokes BFD interface methods to read object files.

3.2. BFD Generation

ISA attributes extracted from the ADL description are encoded within this library. Since we have adopted the ELF format, only the ELF-related files are generated. The file bfd/cpu-[arch].c is the most important among them and contains information such as architecture name, word size and address size.

4. EXPERIMENTAL RESULTS

For the sake of tool validation, we have adopted the well-known Mibench [5] benchmark running on three distinct target CPUs (MIPS, PowerPC and SPARC). Since that benchmark suite requires file manipulation, a feature barely suitable for micro-controller targets, we were obliged to use instead the simpler Dalton benchmark suite [6] for target i8051. To generate assemblers, we have used the tool acasm [2] and to generate disassemblers, we have employed our acdsm tool.

4.1. Validation of generated tools

To validate the generated tools, we performed the following procedure:

- Step 1: Given a target CPU description, acdsm generated a disassembler, while acasm generated an assembler.
- Step 2: Given an input object file, it was fed to the disassembler, giving rise to an assembly output file.
- Step 3: The assembly file was fed to the assembler, resulting in an object output file.
- Step 4: The input and output object files were compared to check whether their ".text" sections matched or not.

Since all the comparisons matched for every benchmark program and each target CPU, there is a strong evidence of proper validation.

It could be argued that such procedure should compare assembly codes (input against output). However, the direct comparison of assembly codes is hampered by the presence of pseudo-instructions or instructions admitting multiple assembly syntaxes. For instance, the MIPS instruction "jump at register" can be written in two different ways: "jr $1" or "j $1". That's why reversed matching was used instead of direct matching, without loss of generality.

4.2. Validation of the generating tool

To check for proper retargetability of the generating tool, the procedure above was repeated for RISC (PowerPC, MIPS, SPARC) and CISC (i8051) targets, whose results are shown in Tables 1 and 2, respectively. The first two columns show the benchmark programs and respective number of files. The remaining columns show the sizes of ".text" sections and disassembling runtimes for each distinct target CPU. Note that the disassembling tools generated with acdsm exhibit a slight increase in runtime as compared to the GNU native disassembling tool (objdump). This increase is a small price to pay for the benefit of achieving automatic retargetability.

Table 1 – Results for RISC targets

Prog.	**#Files**	**".text" size** [Kb] **runtimes** [s] - **acdsm / objdump**		
		MIPS	SPARC	PowerPC
typeset	1	29 0.061 / 0.040	31.9 0.080 / 0.038	24.8 0.057 / 0.046
bitcount	9	4.9 0.016 / 0.009	4 0.015 / 0.010	4.1 0.014 / 0.009
susan	1	63.2 0.117 / 0.083	58 0.152 / 0.062	51.5 0.116 / 0.098
jpeg	60	310.4 0.482 / 0.327	234.3 0.577 / 0.225	223 0.477 / 0.414
fft	3	9.8 0.015 / 0.010	5.4 0.018 / 0.013	5.2 0.016 / 0.014

Table 2– Results for CISC target (i8051)

Prog.	**#Files**	**".text" size** [bytes]	**runtimes** [s]
int2bin	1	188	0.003
cast	1	213	0.003
sort	1	425	0.004
xram	1	214	0.004

5. CONCLUSIONS AND FUTURE WORK

The described ADL-based automatic generator of disassembling tools seems suitable to supporting embedded software development during SoC design space exploration. Experimental results provide evidences of proper functionality, wide-range retargetability and runtime efficiency. Besides, it paves the way to building up an automatically retargetable debugger.

6. ACKNOWLEDGMENT

We would like to thank Alexandro Baldassin (UNICAMP) for his cooperation with this work.

7. REFERENCES

[1] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araújo, C. Araújo, and E. Barros. The ArchC architecture description language. *International Journal of Parallel Programming*, 33(5):453–484, 2005.

[2] A. Baldassin et alii. Extending the ArchC language for automatic generation of assemblers. *Int. Symp. on Computer Architecture and High Performance Computing*, 2005.

[3] D. Casarotto and L. Santos. Automatic Link Editor Generation for Embedded CPU Cores. *4th Int. IEEE Northeast Workshop on Circuits and Systems*, Gatineau, Canada, 2006.

[4] The GNU Binutils package. Available at http://www.gnu.org/software/binutils.

[5] Benchmark Mibench. Available at http://www.eecs.umich.edu/mibench/index.html.

[6] Dalton Project. Available at http://www.cs.ucr.edu/~dalton/i8051/i8051syn.