ARM Simulator

Alpa Shah [ajs248@cs.columbia.edu] - Columbia University

Abstract

A hardware simulator is a piece of software that emulates specific hardware devices, enabling execution of software, that is written and compiled for those devices, on alternate systems. This paper discusses various simulators for the ARM [1] processor, which is widely used in embedded devices like PDAs, cellular phones, ATMs, etc.

1. Introduction

Simulation / Virtual machine technology is an integral part of many computing systems today. Java, SimOS [2] and VMware [for running a complete OS as an application on another operating system], Connectix Virtual PC/Game station and Microsoft's .NET are different examples of such systems. This technology is incredibly useful as a secure means for execution of untrusted software in a sandbox environment, and an ideal platform for code-development for new hardware devices. It also helps preserve software that would otherwise become unusable as legacy hardware become obsolete.

The remainder of the paper is organized as follows: section 2 describes existing ARM simulators based on key ideas behind their implementations, section 3 describes the architecture of my simulator and concludes the paper.

2. Existing Work

There has been a lot of research on software simulation of the ARM processor. These can be categorized according to the level of simulation, whether at the architectural level or the instruction set, or the techniques used, e.g. dynamic recompilation of parts of the simulated software to natively run on the guest system.

2.1 Dynarecs [ARMphetamine]

A simulator normally interprets the binary code of software compiled for the target system. The Dynamic Recompilation [*Dynarecs*] [3] method involves translating parts of this binary code into native machine code at runtime. Native execution of the recompiled code leads to much faster execution of the simulated software. A lot of simulators are developed using this technique, like ARMphetamine [4] and tARMac [5] for the ARM processor, and Embra for the MIPS machine.

ARMphetamine and tARMac are fast and accurate ARM emulators. ARM code program segments are translated into native code as they are being emulated. A fetch-decode-execute emulator starts executing the ARM code, and when a specific block of the ARM code has been executed more times than a preset threshold, a translation routine is employed. This generates "covers" for each source instruction, i.e. chunks of native code that have the same semantics as the translated instructions. These covers are then executed every time the translated block of code needs to be run. The development platform for ARMphetamine and tARMac is Linux/x86.

2.2 Architecture level [SWARM]

SWARM [6] was designed as an ARM module to plug into the SimOS system developed at Stanford University. SimOS allows emulation of various parts of an ARM processor, using either the simple core, or the core and the caches. SWARM was intended not for running ARM binaries on an alternate platform, but rather to allow research into the modification of the internal datapaths of the ARM processor. It implements a small amount of internal co-processors at a basic level, and provides support for the full register/cache/external memory hierarchy.

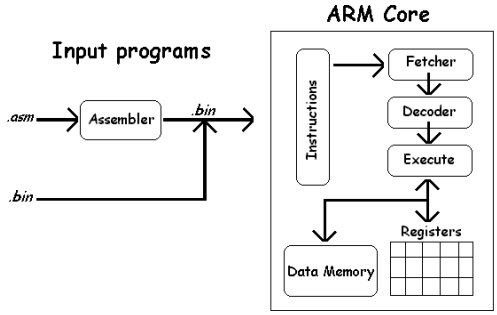
2.3 Instruction Level [SimARM]

SimARM is an instruction set simulator [ISS] that interprets ARM programs at the instruction level obviating the need for ARM hardware. ISSs are simpler to implement, but they are slower than simulators based on dynarecs due to the fact that all instructions are strictly interpreted.

ARMulator [1] is another ISS with a slight variation: it ensures identical cycle-count for instructions. This means that instructions take the same number of cycles to execute as if run on real ARM hardware. This is important for precise simulation since some compilers can optimize code that take advantage of the cycle-counts of specific instructions

3. My Simulator

I propose to build an ISS for the ARM processor like SimARM. The ARM processor has six different modes of operation that determine the number of registers available to instructions. The **user mode** is used for normal program execution, with 16 32-bit registers visible to instructions.



The system will be a fetch-decode-execute style interpreting simulator to load and execute ARM instructions, and it will comprise of: assembler, instruction memory, fetch-decode-execute unit, data memory and registers.

I plan to execute real ARM programs in the simulator, using C compilers like arm-gcc [7] to generate the binaries.

References:

1. **Advanced Risc Machines:** http://www.arm.com/
2. **SimOS project:** http://simos.stanford.edu/
3. **Dynamic Recompilations:** http://www.dynarec.com/
4. **ARMphetamine:** http://www.cs.bris.ac.uk/~brown/docs
5. **tARMac:** http://www.dcs.warwick.ac.uk/~csuix/project/
6. **SWARM:** http://dcs.gla.ac.uk/~micheal/phd/swarm.html
7. **ARM-GCC:** http://celab21.pc.elec.uq.edu.au/~tina/arm-gcc.html