Cogswell and Segall [3] describe a binary-to-binary translation from the 68k to C designed to preserve timing as well as functionality. It tracks the number of clock cycles each basic block would have consumed on the original processor and periodically compares this with wall-clock time, delaying if the new code is getting ahead of schedule. They describe only a few details of their techique.

Engblom and Nilsson [5] describe a simulation environment for multiple processors running in parallel. Their goal is to run the simulation at the same speed as the target hardware (possible because they assume a much more powerful simulator) and keep all the pieces synchronized. Like Cogswell and Segall, they inserting cycle-counting code and each simulated processor periodically relinquishes control if it gets too far ahead. Their problem is more challenging because they attempt to keep multiple processors operating in lockstep.

Steven Collins [4] describes the video hardware for the Atari 800, the Sinclair Spectrum, and the Commodore 64 in loving, technical detail. The Atari 800 and Commodore 64 were similar in that they had dedicated graphics hardware support for sprites, characters, bitmaps, scrolling, and collision detection. Notable are the Atari 800's display lists, which amount to little programs running in the graphics hardware.

Cifuentes PhD thesis [1] describes *dcc*, a 80286 binary-to-C translator for DOS binaries. Her goal in the end is high-level C code. She makes the implicit assumption that the binary was generated by a high-level language compiler.

Cifuentes, Lewis, and Ung [2] describe Walkabout, a regargetable binary translation framework. This is essentially a just-in-time compiler applied to actual machine instructions instead of bytecode for a virtual machine. Their system operates as an interpreter until it find a hot path, then recompiles then. They use the SLED machine description language that is part of the New Jersey Machine Code toolkit (NJMC).

Ramsey and Fernández [6] describe the New Jersey Machine-Code Toolkit, and environment capable of synthesizing assemblers and disassemblers from a description of the machine. It is written in Icon.

In a later paper, Ramsey and Fernández [7] describe the SLED language portion of the New Jersey Machine Code toolkit. It's a clean description, but it only describes the encoding, decoding, and printing of instructions and specifically does not attempt to describe the semantics of the instructions.

References

[1] Cristina Cifuentes. *Reverse Compilation Techniques*. PhD thesis, Queensland University of Technology, Brisbane, Australia, July 1994.

[2] Cristina Cifuentes, Brian Lewis, and David Ung. Walkabout — a retargetable dynamic binary translation framework. In *Fourth Workshop on Binary Translation*, Charlottesville, Virginia, September 2002.

[3] Bryce Cogswell and Zary Segall. Timing insensitive binary to binary translation of real time systems. In *Proceedings of the First Annual Rapid Prototyping of Application Specific Signal Processors (RASSP) Conference*, Arlington, Virginia, August 1994.

[4] Steven Collins. Computer graphics during the 8-bit computer game era. Technical Report TCD-CS-1998-15, Trinity College, Dublin, Ireland, September 1998.

[5] Jakob Engblom and Magnus Nilsson. Time-accurate simulation: Making a PC behave like a 8-bit embedded CPU. Technical Report 2002-024, Department of Information Technology, Uppsala Unversity, Sweden, July 2002.

[6] Norman Ramsey and Mary F. Fernandez. The new jersey machine-code toolkit. In *Proceedings of the USENIX Technical Conference*, pages 289–302, New Orleans, Louisiana, January 1995.

[7] Norman Ramsey and Mary F. Fernández. Specifying representations of machine instructions. *ACM Transactions on Programming Languages and Systems*, 19(3):492–524, May 1997.