Pedagogic Value in Understanding Computer Architecture of Implementing the Marie Computer from Null and Lobur in the Logic Emulation Software, Multimedia Logic

Timothy Daryl Stanley Daniel Prigmore Scott Mikolyski

George Embrey Leslie Fife Don Colton

Brigham Young University - Hawaii
BYUH #1854
Laie, Hawaii 96762-1294
1-808-293-3388

Stanleyt@byuh.edu

ABSTRACT

In our computer architecture course, we ask students to design an instruction set for an eight- or sixteen-bit computer and then implement that design in an emulated computer using a logic emulation package. In Winter semester 2006 a team of three students decided to design and implement the "Marie" computer instruction set described in the book by Null and Lobur[2]. The project proved to be highly motivational for this team, and they produced an excellent result. This paper describes that design process, the resulting computer, and learning from the project.

Categories and Subject Descriptors

C.1.1 Computer Systems Organization: PROCESSOR ARCHITECTURES Single Data Stream Architectures *Von Neumann architectures*

General Terms

Design

Keywords

Microprocessor design, Computer architecture, Logic Emulation, Education.

1. INTRODUCTION

Starting from the premise students learn by doing, we ask students in our computer architecture course to design and implement an instruction set and architecture in a logic emulation software package called Multimedia Logic (MML)[1]. The reasons for this choice include that MML is open source, free, and has a rich, visually pleasing set of input and output devices.

WCAE '07, June 9, 2007 San Diego, CA

The stage is set for this task by working through a couple of designs with the students. These are typically 8-bit computers using alternate design architectures. For example one is von Neumann with a several-step execution cycle and the second is a Harvard architecture with single-cycle instruction execution. These are designed from an instruction set and register design. More detail on this process is given in the papers "From Archi Torture to Architecture: Undergraduate students design and implement computers using the multimedia logic emulator" [2], and "Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs" [3]. Students are then asked to start by providing an instruction set and register layout. We then discuss challenges presented by the architectural choices, and discuss solutions. This work is normally done in small teams of two to three students, but if an individual would prefer to work alone, they have that option.

Using this approach many novel computers and circuits have been designed and build by our students. Some of these include 16-bit Harvard and von Neumann designs. Some related circuitry developed includes an eight-bit multiplier, a sixteen-bit register array of sixteen registers, and a 128 bit XOR encryption unit. This paper focuses on a particular project from Winter Semester 2006, called the Marie computer with is described in the book by Null and Lobur. This Marie computer was designed and implemented by a three-student team.

2. MARIE BY NULL AND LOBUR

In the book "The Essentials of Computer Organization and Architecture"[4] the authors introduce Marie, "Machine Architecture that is Really Intuitive and Easy." They provide with the resources for their book, a Java application that emulates the Marie. It also edits and assembles programs written for Marie and provides assembly listings and core dumps of the Marie memory space. The Marie emulated computer interface is shown in figure 1.

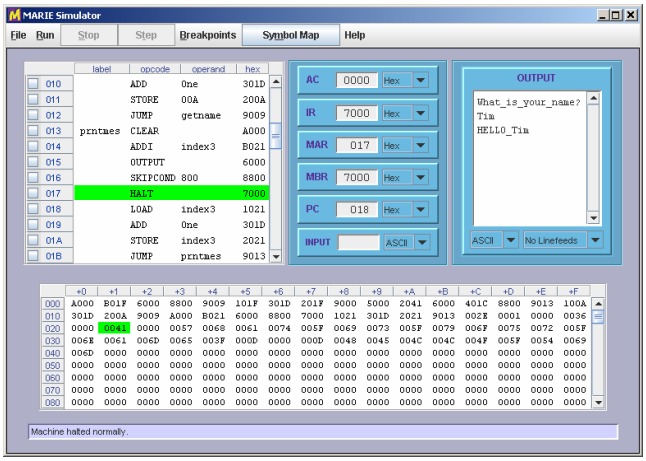

Figure 1 Marie computer emulator from Null and Lobur [4]

One novel feature is a data path animation that illustrates the steps that occur as each instruction is executed. As each component of the data path is used it is highlighted. This data path animator is shown in figure 2.

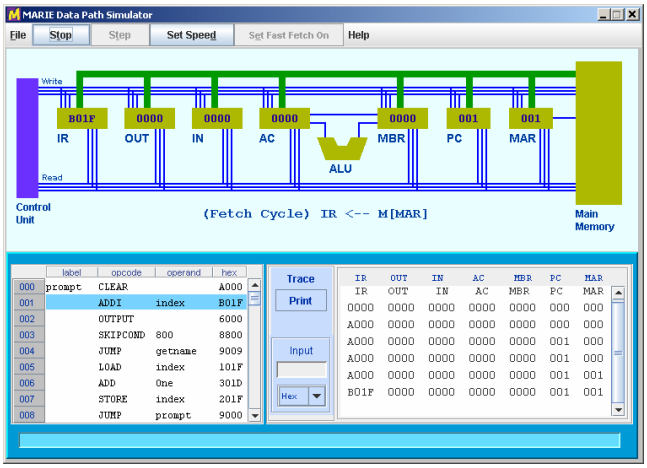

Figure 2 Marie data path simulator by Null and Lobur [4]

The Marie was designed to have a limited but adequate set of thirteen instructions. These instructions were chosen to allow direct and indirect addressing and provide enough capability to allow real assembly language programs to be written, but without the onerous task of learning a complex instruction set. Between the very complete description in the book and the java Marie application this is a very well defined instruction set and architecture. Figure 3 gives the complete Marie instruction set along with register transfer notation (RTN) for the process by which each instruction is executed. Note that some instructions like clear the accumulator and halt can be executed in one cycle while others like save and jump (JnS) require seven steps. This turned out to be one of the major design challenges of this project.

Opcode	Instruction	RTN
0000	JnS *X*	MBR ← PC MAR ← *X* M[MAR] ← MBR MBR ← *X* AC ← 1 AC ← AC + MBR PC ← AC
0001	Load *X*	MAR ← *X* MBR ← M[MAR], AC ← MBR
0010	Store *X*	MAR ← *X*, MBR ← AC M[MAR] ← MBR
0011	Add *X*	MAR ← *X* MBR ← M[MAR] AC ← AC + MBR
0100	Subt *X*	MAR ← *X* MBR ← M[MAR] AC ← AC - MBR
0101	Input	AC ← InREG
0110	Output	OutREG ← AC
0111	Halt	
1000	Skipcond	If IR[11–10] = 00 then If AC < 0 then PC ← PC + 1 Else If IR[11–10] = 01 then If AC = 0 then PC ← PC + 1 Else If IR[11–10] = 10 then If AC > 0 then PC ← PC + 1
1001	Jump *X*	PC ← IR[11–0]
1010	Clear	AC 0
1011	AddI *X*	MAR ← X MBR ← M[MAR] MAR ← MBR MBR ← M[MAR] AC ← AC + MBR
1100	JumpI *X*	MAR ← *X* MBR ← M[MAR] PC ← MBR

Figure 3 Marie instruction set with register transfer steps [4]

2.1 Decision to Design and Build Marie

Since the professor was using Marie in his computer organization class, he wanted an implementation of the actual Marie computer to use as a visual aid for these organization classes. Also the Marie is very well defined and comes with an excellent assembler. Also a number of programs were available to test the design. So, even though it is more complex than the usual design projects accomplished for our computer architecture classes, our team of three students took the challenge to design and implement the Marie in Multi-Media Logic. The project took about 300 hours over the semester to complete, but the students reported finding the project compelling and learning a lot.

2.2 Implementing Marie in Multimedia Logic

Implementing the Marie computer took twelve pages in Multimedia Logic. Each page fills a single canvas comfortably. These twelve pages are included as figures 4 through 15. They are presented for completeness, even though many are similar. Notice that many registers are composed of sixteen data latches (D flip flops). One exception is the in and out registers which only use the lower eight bits. The images are captured with the computer running a "Hello World!" program.

Figure 4, shows the Memory Address Register (MAR) with it's interface devices. The MAR connects to the bus and to the address lines of the memory.

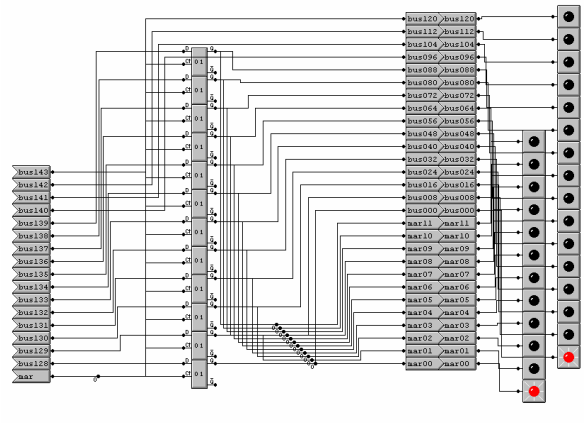

Figure 4 Memory Address Register (MAR) for Marie computer

Figure 5, shows the Memory Buffer Register (MBR) which holds data read from or to be loaded into memory and the accumulator, and provides one operand to math operations.

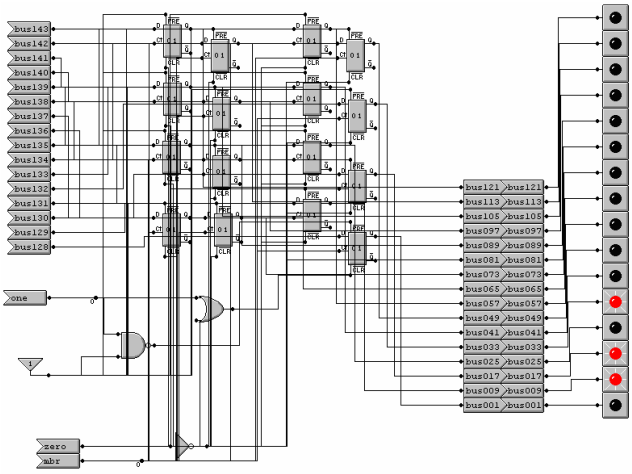

Figure 5 Memory Buffer Register (MBR) for Marie computer

Figure 6, shows the accumulator and the ALU. Since the accumulator always provides one of the inputs to the ALU this is hard-wired between the output of the accumulator and the "A" input of the ALU, and also between the output of the ALU and the input of the accumulator.

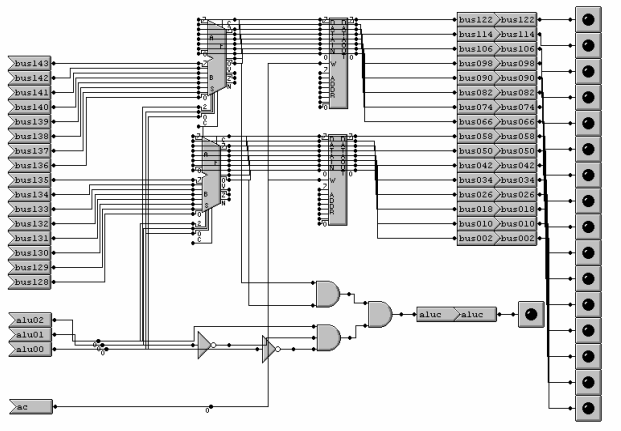

Figure 6 Accumulator and ALU for Marie computer

Figure 7 shows the input register which is loaded from the keyboard in the I/O module and read by input commands. In the case of this particular program, "Hello World", input is not used so the input register shows as "undefined" as indicated by the partially illuminated LEDs.

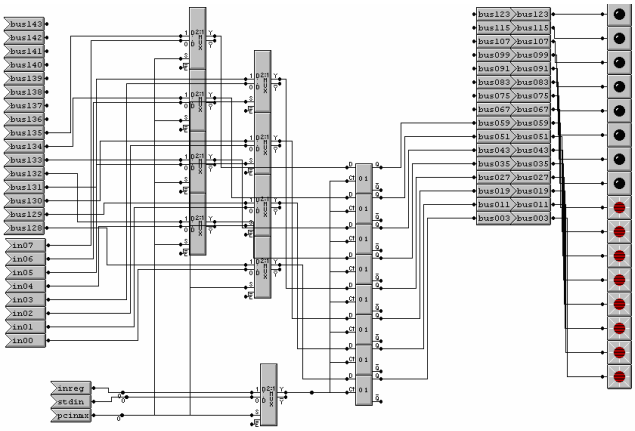

Figure 7 Input Register for Marie computer

Figure 8 shows the output register which holds data to be displayed on the ACSII display device. Again, it is the lower eight bits from the data bus when an output command is given. While the actual Java Marie from Null and Lobur provides the option of hex, decimal, and ASCII I/O we have only provided ASCII I/O in this implementation. However the other options could be easily added.

Figure 9 shows the instruction register which holds the instruction currently being executed. It's output is the principle input to the instruction decoding module.

Figure 10 shows the memory. This is a sixteen data bit memory with twelve address lines. A twelve bit address is used since four bits of each instruction is used for operation code, and the remaining twelve bits are available for an address. This is a true von Neumann design where data and instructions share the same memory space. With the limited instruction set available in Marie the ability to modify program memory during execution allows an effective and efficient, if dangerous indirect write command.

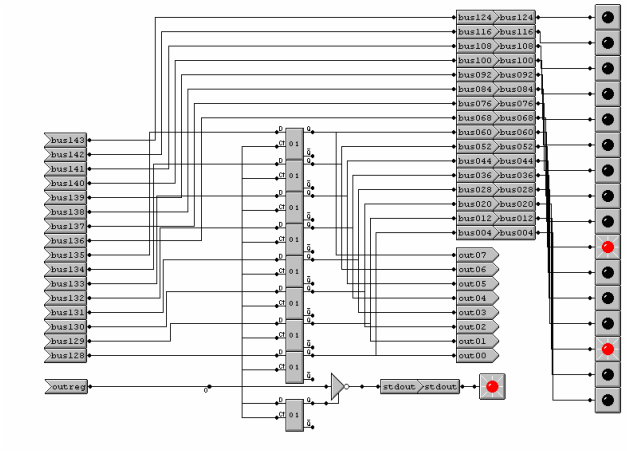

Figure 8 Output Register for Marie computer

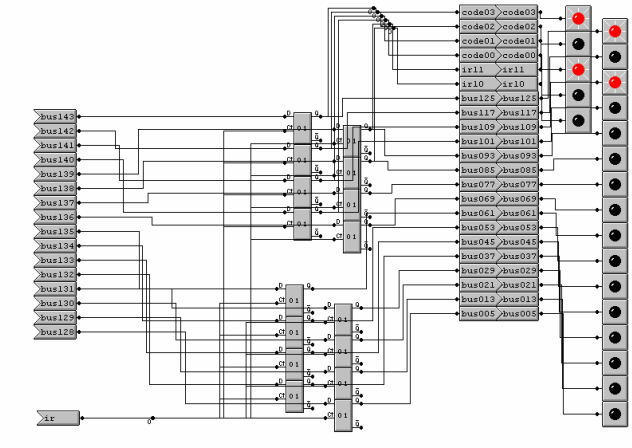

Figure 9 Instruction Register (IR) for Marie computer

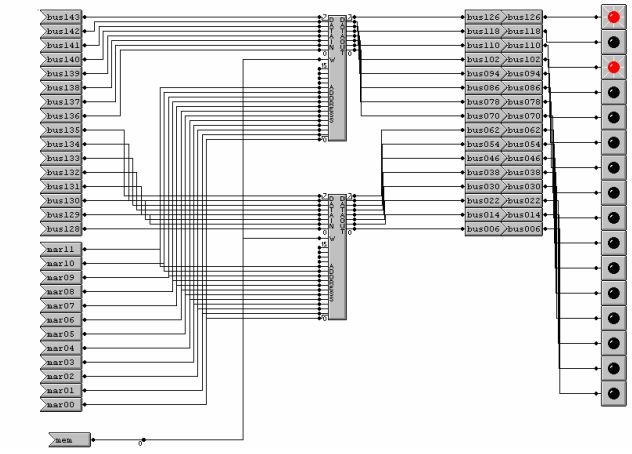

Figure 10 Memory for Marie computer

Figure 11 shows the program counter. In many multi-cycle designs the same ALU is used to both increment the program counter and perform mathematical and comparison operations. In this case to simplify the data and control paths a separate ALU is used to increment the program counter as shown in figure 11.

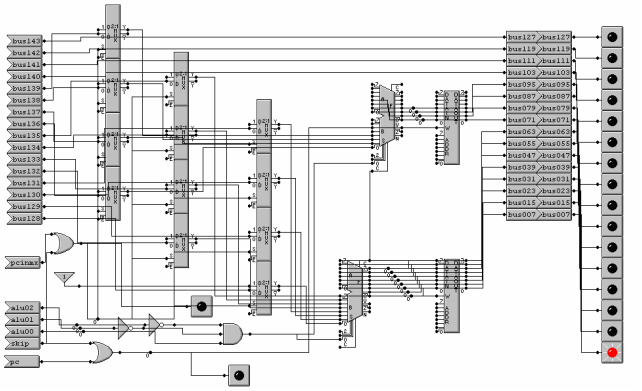

Figure 11 Program Counter (PC) with incrementer for Marie computer

Figure 12 shows the data bus. The data bus involves the heavy use of multiplexers to enable only one device to drive the bus at a time.

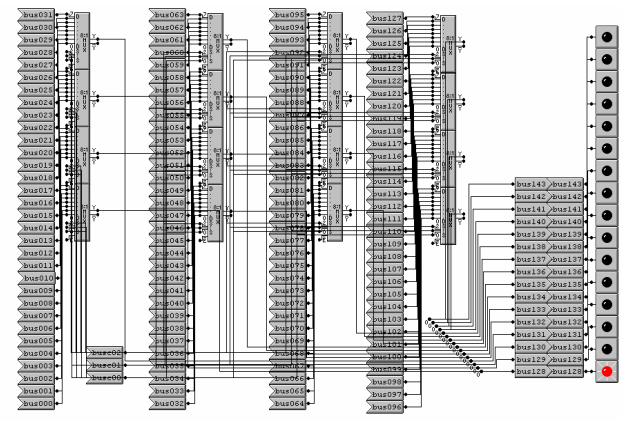

Figure 12 Bus for the Marie computer

Figure 13 shows the control logic, which is the most difficult part of the design. The purpose of the control logic is to convert each instruction into a sequence of control line states to accomplish the instruction being executed. Some commands have many sub tasks while some have none, further complicating the instruction decode process. The key component in the decoder is the "Read-Only" memory that takes as inputs the current state and provides as an output the control line signals. The switch at the upper left of this figure enables and disables the clock for this computer to start and stop program execution. Below the clock control switch is a push button that allows the computer to be stepped one clock pulse at a time. The lower array of switches provides op codes for diagnostic purposes.

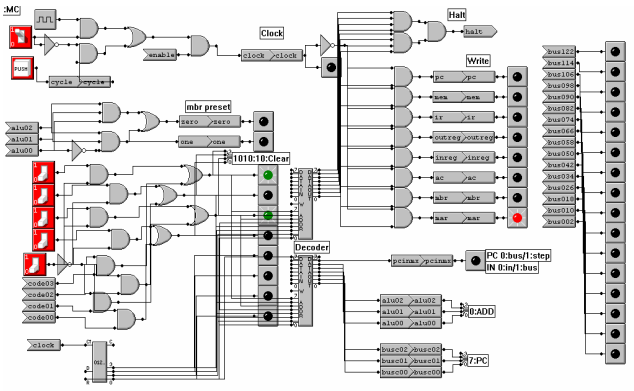

Figure 13 Control logic for Marie computer

Figure 14 shows the conditional execution logic which allows the next step to be skipped depending on the value in the accumulator. A parameter passed with the instruction determines if the instruction is skipped if the accumulator is greater than zero, equal to zero, or less than zero. In our implementation only the test for the Accumulator equal to zero and the accumulator not equal to zero work.

.

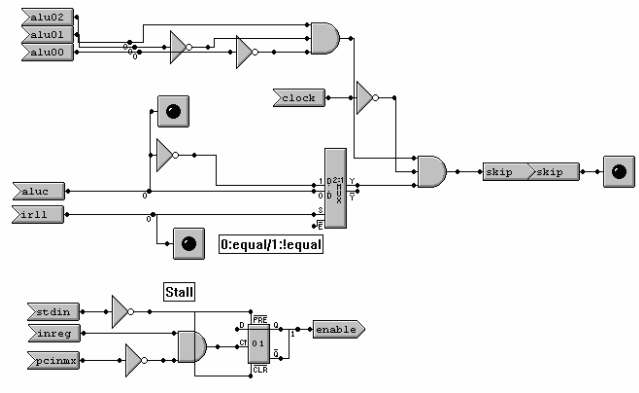

Figure 14 Conditional execution logic

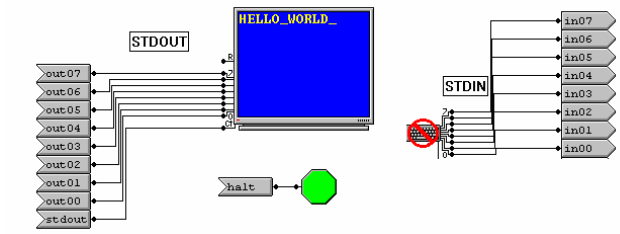

Figure 15 Input / Output interface for Marie computer

Figure 15, the final module is the Input, output and halt control. This is the module a user would interact with. When the simulation is executing, mouse clicking on the keyboard icon will remove the red from the figure and the computer keyboard will provide data into the circuit

2.3 Pedagogic Value of this project

In the students own words: "Fortunately, every member of our group was highly motivated not only to succeed, but to excel at our task. We decided from the onset to exceed the expectations of our professor. Approaching the project with a desire to learn how computers really work was the primary source of motivation for our group and the key to our success. This method made a huge impact on our project because an understanding of the various components that are used to build a computer allowed us to try several different ways to accomplish a task when previous attempts failed. Additionally, this approach helped our group develop a machine that was as straightforward as we could make it while still being functional.

Apart from learning the inner workings of computers, we learned how to function as a team. Group members were assigned various components of the MARIE computer to work on but we found that typically the efforts of the individual needed to be revised by the group as a whole. The most effective strategy we arrived at was to have at least two members of the group in front of one monitor to brainstorm, create, and revise. This team-centered approach increased our productivity dramatically over working independently. Furthermore, the group methodology ensured that another person was almost daily monitoring the effort we put in to the project and lazy or sluggish work would not be tolerated. This forced each member of the group to contribute and work hard.

The most rewarding moment during the semester came when our MARIE computer ran its first program from start to finish. After countless hours of building the machine and the often frustrating times fine-tuning it, our first program to run was a simple "hello world". The satisfaction that came from completing the monumental task we had is indescribably gratifying. Hard work and persistence definitely paid off. What is more, we now have a significantly greater understanding of how computers operate."

3. CONCLUSIONS

A three student team has designed and built the Marie computer in Multimedia Logic. The computer can take code assembled with Null and Lobur's java emulator and run it. Since this is a full emulation of the Marie computer, access is available to all of the control lines, data bus, and intermediate states that are not accessible in the java emulator. Through this process they have learned computer architecture by doing, and have produced a valuable teaching aid for computer organization classes learning assembly language using the Marie instruction set.

Some have asked why build an emulated MARIE computer when an excellent assembler, simulator, and data path simulator already exist. The reason is a desire to have students fully appreciate David Patterson's comment that a CPU is "a data path and control circuitry. [5]" These students fully appreciate data paths and control circuitry because they have designed and built both.

Others have asked, "How large of a project is needed to meet this understanding. The goals of this educational experience could have been met with an eight bit computer design, but highly motivated students don't just meet the minimum requirement.

4. ACKNOWLEDGMENTS

Our thanks to George Mills, the author of Multimedia Logic, for making his product available without cost on his web site www.softronix.com. Also thanks to Linda Null and Julia Lobur for developing the Marie instruction set and building an emulator and data path animator for it.

5. REFERENCES

[1] Mills, George, Multimedia Logic, www.softronix.com

[2] Stanley, Wong, Prigmore, Benson, Fishler, Fife and Colton, From Archi Torture to Architecture: Undergraduate students design and implement computers using the multimedia logic emulator, Computer Science Education, Vol. 17, No. 2, June 2007, pp. 143 – 154

[3] Stanley, Xuan, Fife, Colton, Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs, Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007.

[4] Null, L., and Lobur, J. *The Essentials of Computer Organization and Architecture.* Jones and Bartlett Publishers, Sudbury, MA 200

[5] Patterson., and Hennessy. *Computer Organization and Design, the Hardware/Software Interface.* Morgan Kaufmann Publishers, San Francisco, CA, 2005