

# In-Lecture Hardware Demonstrations with a Logic Analyzer to Illustrate Pipelined Execution and Cache/Memory Behavior

Naraig Manjikian  
Department of Electrical and Computer Engineering  
Queen's University  
Kingston, Ontario, Canada K7L 3N6  
email: nmanjiki@ee.queensu.ca

*Abstract*—This paper describes the use of a logic analyzer for in-lecture demonstrations of pipelining and memory hierarchy on representative hardware. The aim is to enhance understanding of fundamental concepts without excessive technical detail. The hardware used in demonstrations is described along with the examples used and their pedagogical aims. Student feedback on this approach is also presented.

## 1 Introduction

The senior undergraduate course *ELEC470 Computer System Architecture* at Queen's University is a lecture course that emphasizes architectural aspects of pipelining and memory hierarchy. Similar courses at other institutions may include a practical component based on programmable logic devices [2, 7]. At Queen's University, students in ELEC470 already have hardware experience with programmable logic in three earlier courses, and they must also take a full-year project design and implementation course.

Nonetheless, it is valuable to relate concepts to practical systems. To this end, a commercial evaluation board based on a MIPS processor has been used with a logic analyzer for in-lecture hardware demonstrations. This arrangement, complemented with written documentation, provides an excellent means of observing and explaining the behavior and timing of an actual pipelined processor with caches and external DRAM memory.

This paper outlines the context for in-lecture demonstrations in ELEC470, describes the hardware and examples used in demonstrations, provides feedback from students on this approach, and concludes with future plans.

## 2 Context

The following list outlines courses that are specific to Computer Engineering at Queen's University:

- Digital logic design (including a separate laboratory component based on FPGAs),
- Basic computer architecture (processor organization with a laboratory component on assembly-language programming),
- Microprocessor systems (details of bus/memory timing and hardware/software interfacing with I/O ports; includes a laboratory component with a take-home microcomputer kit and FPGAs),
- Digital systems engineering (high-speed electrical considerations, arithmetic circuits, storage, testability, fault tolerance; with a major FPGA-based project),
- Computer system architecture (performance, instruction set design, pipelining, caches, memory hierarchy, I/O systems, multiprocessing).

The ELEC470 course that is the subject of this paper is the final course in the sequence given above. ELEC470 is based on the undergraduate book by Hennessy and Patterson [3]. All chapters are covered except for arithmetic (that topic is covered in an earlier course). The course material includes details on SRAM/DRAM chips and their timing, as well as the IDT RC36100 MIPS microprocessor [5], with permission obtained to reproduce technical documentation for educational and classroom usage. Finally, the SimpleScalar [1] simulator is also used in quantitative and illustrative exercises.

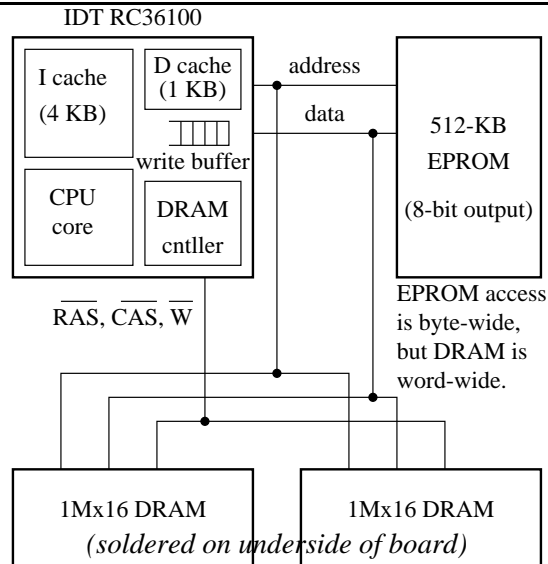


Figure 1: IDT RC36100 Evaluation Board

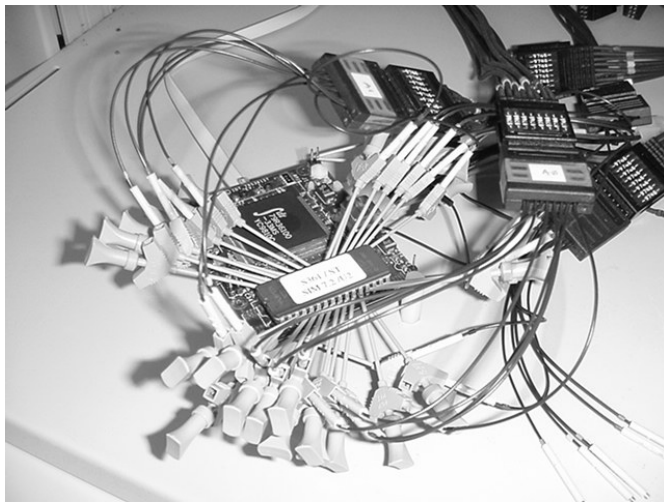


Figure 2: RC36100 Evaluation Board with probes

Since ELEC470 emphasizes pipelining and memory hierarchy, using hardware to demonstrate relevant concepts has strong educational value. Contemporary PC or workstation computers are not suited for in-lecture demonstrations with a logic analyzer because gaining access to signals is nontrivial, and their complexity is not conducive for illustrating fundamental concepts. This paper outlines the use of simpler hardware and describes examples used in demonstrations.

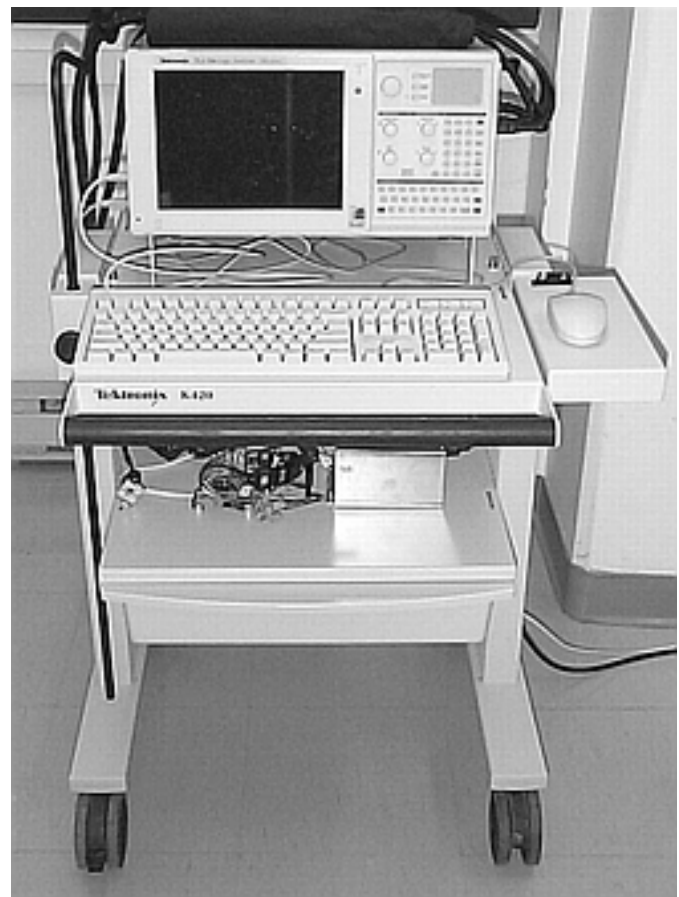


Figure 3: Tektronix TLA704 Analyzer on cart

### 3 Overview of Hardware

**Microprocessor System** An evaluation board [4] based on the IDT RC36100 MIPS microprocessor [5] is used because it is simple yet representative of embedded hardware designs. The IDT RC36100 is a highly-integrated product that includes on-chip instruction and data caches, timers, serial ports, parallel ports, and a DRAM controller. The RC36100 evaluation board consists of a microprocessor, EPROM chip, and two DRAM chips, as shown in Figure 1. A crystal oscillator and RS-232 level converter are the only other support chips on the board. Figure 2 shows a photograph of the actual 2.5in×3in board with probes connected for logic analysis.

The RC36100 is derived from the MIPS R3000 that used a write-through cache policy. Hence, the RC36100 integrates a 4-deep write buffer to allow the processor to

modify the data cache and continue fetching one instruction every cycle from the instruction cache while writes to DRAM proceed from the write buffer at slower speeds.

The EPROM chip on the board contains a monitor program that accepts commands to download machine code into DRAM memory, view/modify memory contents, assemble/disassemble instructions, and execute code. To generate machine code, IDT provides a version of the GCC compiler and related tools. These tools are hosted on a separate PC or workstation, and machine code is downloaded to the board through a serial data link.

**Logic Analyzer** The Tektronix TLA704 logic analyzer combines a PC platform running Microsoft Windows with hardware that samples digital signals at 2 GHz and graphical software that displays the acquired samples. Figure 3 shows a TLA704 mounted on a cart that provides mobility for in-lecture demonstrations; the shelf carries the RC36100 evaluation board and a power supply. The TLA704 has an integrated LCD display and also a VGA port to connect to a projector in a lecture hall. This projection capability has been used in a prerequisite course *ELEC371 Microprocessor Systems* that is also taught by the author [6]. Students in ELEC470 therefore have familiarity with the output display format of the logic analyzer. The TLA704 also serves as the terminal for the RC36100 evaluation board through a serial link. A terminal program running on the TLA704 is used to enter commands for the EPROM-based monitor. The Windows software allows switching between the terminal program and the logic analyzer software as needed.

**Probing for Logic Analysis** The RC36100 chip is fabricated in a package with fine-pitch pin spacing that prevents connection of individual logic analyzer probes. Fortunately, the pitch of the EPROM and DRAM pins on the evaluation board permits the connection of probes in the manner shown in Figure 2 without requiring a special probe module. Address and control lines, along with a subset of data lines, are probed for logic analysis.

The RC36100 includes an integrated DRAM controller, hence the external address lines carry multiplexed row/column addresses. The *original* address must therefore be reconstructed from observations on the address bus. Address pins A11..A2 carry both row and column information, and these pins represent the *true* column address during fast page mode accesses. For rapid identi-

fication of the original address, the *effective* column address is formed from A11..A0. The on-chip DRAM controller sets A1..A0 to zero because they are not used in row/column addressing of the external DRAM chips.

## 4 Demonstrating Delay Slots

The ELEC470 course material based on the Hennessy/Patterson textbook [3] emphasizes the use of load-use stalls and flushing of instructions following a branch or jump. The alternative approach of delay slots is presented as an example of how earlier implementations sought to simplify hardware by shifting the responsibility of code scheduling to the compiler.

An initial in-lecture demonstration using the RC36100 centers on delay slots in pipelined execution. The pedagogical aims are to increase understanding of delay slots and to develop the ability to predict and explain behavior. The effect of delay slots is illustrated by presenting assembly-language code segments, and then observing their execution on the RC36100 evaluation board. The annotated code below demonstrates to students an interesting consequence of a jump delay slot.

```
<IDT>dis 0xa0010000
a0010000 lui  a0,0xa002 # a0=0xa0020000
a0010004 li   t0,0x0  # t0=0x00000000
a0010008 j    0xa001000c # to next instr
a001000c addi t0,t0,0x1 # t0 = t0 + 1
a0010010 sw   t0,0x0(a0) # mem[a0+0]=t0
a0010014 lui  v0,0xbfc0 # v0=0xbfc00000
a0010018 jr   v0        # to monitor
a001001c nop          # delay slot
<IDT>go 0xa0010000    # execute code
<IDT>dump 0xa0020000 # examine mem
a0020000: 00000002        # final t0 value
```

The `addi` that is the jump target is also in the jump delay slot. The value written to `0xa002000` from `t0` indicates that the register was incremented *twice*.

Another example illustrates the load delay slot:

```
lui s0,0xa002 # s0=0xa0020000
lui t0,0x1234 # t0=0x12340000
ori t0,t0,0x5678 # t0=0x12345678
lw  t0,0x0(s0) # t0=mem[s0]
sw  t0,0x4(s0) # mem[s0+4]=old t0
lui v0,0xbfc0 # v0=0xbfc00000
jr  v0        # return to monitor
<IDT>dump 0xa0020000
a0020000: ffffffff 12345678
```

The value at `0xa002004` is not from the `lw`, but is determined instead by the `ori`. Such examples reinforce the concept of proper code scheduling with delay slots.

## 5 Demonstrating Caches and DRAM

ELEC470 covers memory hierarchy from basic caching to DRAM timing details. The RC36100 provides a simple external interface to service cache misses and writes from its write buffer. The low-level behavior can be observed with the logic analyzer and related directly to assembly-language and C code. The pedagogical aim is to instill a complete understanding of execution behavior in the presence of caches and a write buffer.

Using the GCC compiler that is provided with the RC36100 evaluation board, the following C code is used as part of a demonstration of the memory hierarchy.

```
#define DATA_PATTERN 0xa5a5a5a5
#define N 8

void TestFunction ()
{
    int i;

    /* used to trigger the analyzer */
    write_marker = DATA_PATTERN;

    for (i = 0; i < N; i++)
        a[i] = b[i];
}
```

The corresponding assembly-language code is shown in Figure 4. The effective column address for `write_marker` is printed to the terminal by the `main()` program. With a pause for a keypress, the trigger condition for the logic analyzer can be set with the address and data patterns in order to capture the write-through DRAM access for `write_marker`.

Once a key is pressed, the `main()` program calls the test function *twice* in succession in order to demonstrate the effect of caching on execution time. The first time that the function is called, there will be instruction cache misses, and data cache misses for array elements in the right-hand side of the assignment statement. Writes in the left-hand side of the assignment statement do not incur cache misses because of the write-through policy.

The logic analyzer output display after triggering is shown in Figure 5. The trigger marker (identified with

‘T’) is positioned by the analyzer. The other markers (labelled ‘1’ and ‘2’) are positioned manually at the end of the first call of the test function and at the end of the second call. For the first call, instruction cache misses fetch four-word blocks from DRAM using fast page mode (left side of Figure 5). The data cache uses one-word cache blocks, hence only a single word is read from DRAM on data cache misses. This difference in cache block size is useful in distinguishing between instruction cache misses and data cache misses (write-throughs are obvious from the assertion of the write signal). The benefit of caching is clearly evident in the reduced execution time for the second call. Only write-throughs are needed, and these occur in parallel with continued execution of other instructions. The execution time between the markers labelled ‘1’ and ‘2’ is reported as the “Delta Time” of  $2.32 \mu\text{sec}$ , and this value can also be confirmed by assuming one instruction fetched every cycle for the loop code in Figure 4.

The behavior during cache misses is then inspected more closely by using the *zoom* feature of the logic analyzer. Figure 6 shows the diagrams in documentation that is provided to students in order to complement the in-lecture demonstration. A reduced version of the overview from Figure 5 is manually annotated with a box to indicate the region of interest in the full-size image in Figure 6. During the in-lecture demonstration, the display is scrolled left/right and zoomed in/out while the behavior is explained, but students are provided with printouts of timing diagrams with explanations in order to prepare for in-lecture demonstration and to review afterwards.

The correspondence between the captured waveforms and the code in Figure 4 is explored by examining the fast page mode access in Figure 6. Four words beginning at address `0x?????1B0` are accessed, and these words correspond to the four instructions beginning with the `addiu` instruction at `0x800201B0` in Figure 4 (as confirmed by comparing D31-D24 to the most significant byte of each instruction). The effective column address makes it easy to identify the original address. Other regions in Figure 5 are examined similarly in detail.

A useful feature of the RC36100 evaluation board is the ability to force uncached instruction execution, where each instruction must be read from DRAM, even if it is reused in a loop. Figure 7(a) shows an acquisition from uncached execution, with unlabelled markers marking the end of the first call of the test function and the end of the second call. For comparison, the cached execution in Fig-

---

```

000000080020194 <TestFunction>:
 80020194: 3c02a5a5    lui    $v0,0xa5a5      # build constant
 80020198: 3442a5a5    ori    $v0,$v0,0xa5a5  # for write_marker
 8002019c: af828000    sw     $v0,-32768($gp) # perform the write
 800201a0: 00002821    move  $a1,$zero       # a1 = i (loop counter)
 800201a4: 3c048002    lui    $a0,0x8002     # build pointer to a
 800201a8: 24841ba0    addiu $a0,$a0,7072    # 7072 = 0x1ba0
 800201ac: 3c038002    lui    $v1,0x8002     # build pointer to b
 800201b0: 24631b80    addiu $v1,$v1,7040    # 7040 = 0x1b80
 800201b4: 8c620000    lw     $v0,0($v1)     # loop: load b[i]
 800201b8: 24630004    addiu $v1,$v1,4      # advance ptr to b
 800201bc: 24a50001    addiu $a1,$a1,1      # i = i + 1
 800201c0: ac820000    sw     $v0,0($a0)     # store to a[i]
 800201c4: 28a20008    slti  $v0,$a1,8      # check: i < 8 ?
 800201c8: 1440fffa    bnez  $v0,800201b4   # if yes, goto loop
 800201cc: 24840004    addiu $a0,$a0,4      # advance ptr to a
 800201d0: 03e00008    jr    $ra             # return from subroutine
 800201d4: 00000000    nop                    # <jump delay slot>

```

Figure 4: Output from gcc compiler for test code with annotations to aid understanding

---

ure 7(b) has markers positioned for cached execution ('1' and '2') and uncached execution (unlabelled markers), relative to the trigger point. The importance of caches is reinforced by observing the marked difference in execution times and behavior with the logic analyzer.

Two final aspects of DRAM behavior can be also demonstrated with acquisitions from the logic analyzer. The first is row access optimization where the  $\overline{RAS}$  signal is held asserted for extended periods of time in anticipation of successive column addresses in the same row. Figure 7(b) shows instances of this optimization. The second aspect is that DRAM memory must be refreshed periodically. The  $\overline{CAS}$ -before- $\overline{RAS}$  refresh protocol means that  $\overline{RAS}$  must be deasserted. Figure 7(b) shows two occurrences of this deassertion near the two unlabelled markers. The analyzer display can then be used to closely examine the signal timing at these points. The interval between refreshes can also be measured and related to the DRAM specifications.

Altogether, there are a variety of important concepts that can be effectively demonstrated in lectures using the combination of the evaluation board and logic analyzer. With appropriate examples that are kept simple enough to clearly illustrate aspects of pipelining and the memory hierarchy, student understanding can be enhanced.

## 6 Student Feedback

As part of a broader effort to collect student feedback, a survey was conducted near the end of the course. Among other things, the survey sought feedback on the documentation for the demonstrations, as well as the value of the demonstrations themselves. Out of 60 students, 29 survey forms were returned. The responses relevant to this paper are given below:

- Was the documentation on memory timing useful?
 

yes:	28%	(8 /29)	*****
somewhat:	55%	(16 /29)	*****
not really:	17%	(5 /29)	****
- Were the RC36100 demonstrations educational?
 

yes:	41%	(12 /29)	*****
somewhat:	45%	(13 /29)	*****
not really:	14%	(4 /29)	****

Note that the feedback for the first question is on the instructor-supplied custom documentation for the in-lecture demonstration, not on the vendor-supplied commercial documentation. Overall, the feedback may be viewed as being relatively positive. Although a minority of responding students did not feel there was a benefit for them, the majority of students did indicate that there was value in using in-lecture hardware demonstrations.

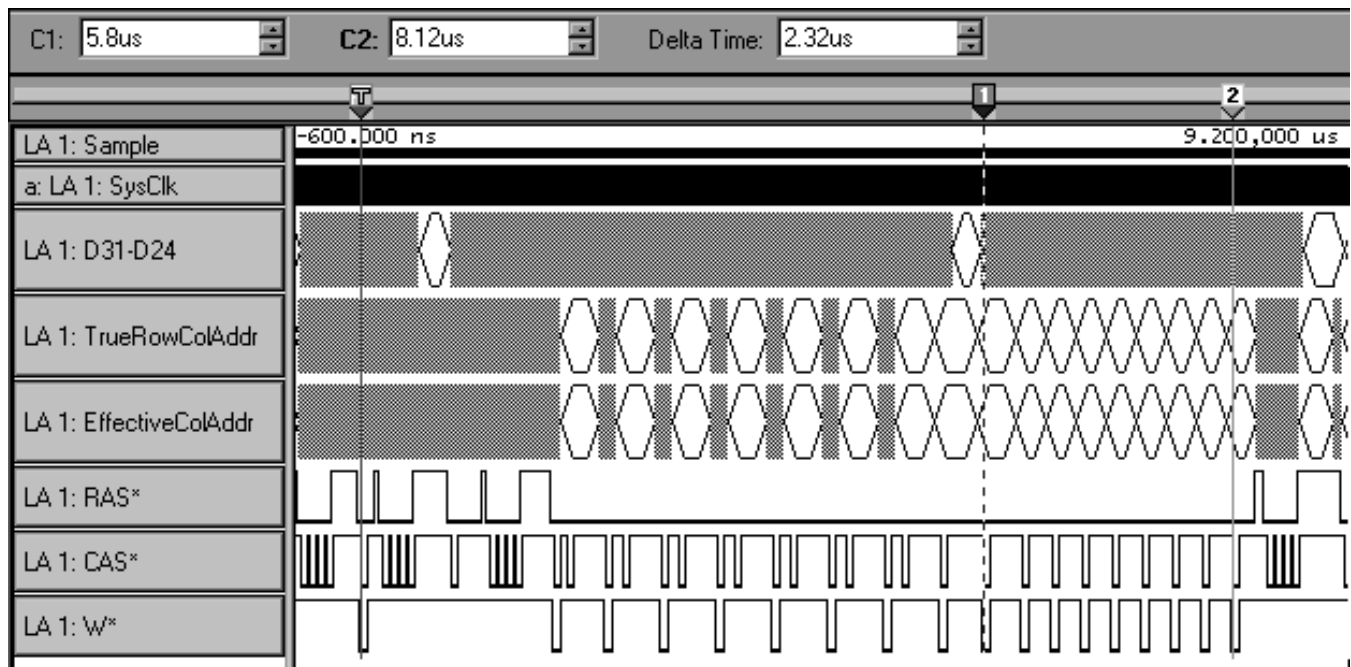


Figure 5: Timing acquisition for cached execution of sample code

## 7 Conclusion and Future Plans

The use of in-lecture demonstrations for the ELEC470 course using a logic analyzer and representative hardware to illustrate pipeline behavior and memory timing has value in aiding student understanding and increasing interest in the course material. Hardware-based examples can complement the simulator-based exercises that the students already study in the course, and student feedback is generally supportive of demonstrations.

In considering future plans for continued use of in-lecture demonstrations, an appropriate balance must be maintained between exposition of fundamental concepts and demonstration of those concepts in operation. Currently, less than 10% of the 36 lecture hours in the term are dedicated to in-lecture demonstration time. There is, however, the opportunity to increase that figure somewhat. Possibilities include comparison of the pipelined performance of unoptimized and optimized code from the compiler, observations and predictions of performance for sample code that forces the write buffer to become full and stall the processor, and demonstration of the impact of cache mapping conflicts with suitable examples that show the sensitivity of performance to array start-

ing addresses. These examples can effectively utilize the ability to observe cache misses with the logic analyzer and the ability to rapidly identify full addresses from the effective column address in order to relate the observed behavior to assembly or C source code.

There is also opportunity to extend the benefit of viewing output from the logic analyzer not only in a static form on paper documentation, but also in a more dynamic form outside of lectures. Tektronix makes available a software package for stand-alone PC computers that has precisely the same viewing features (i.e., scrolling and zooming) as the full software on the logic analyzer hardware. Observations captured by the logic analyzer can be saved to a data file, and this data file may then be examined and manipulated on any other computer with the viewing software. With this ability, students may review any aspect of the in-lecture demonstration at their leisure.

Finally, the relatively low cost of the RC36100 evaluation board allows the consideration of acquiring additional boards for allowing students to experiment hands-on with the hardware. The Department of Electrical and Computer Engineering at Queen's University has more than one TLA704 analyzer, as well as other PC-based an-

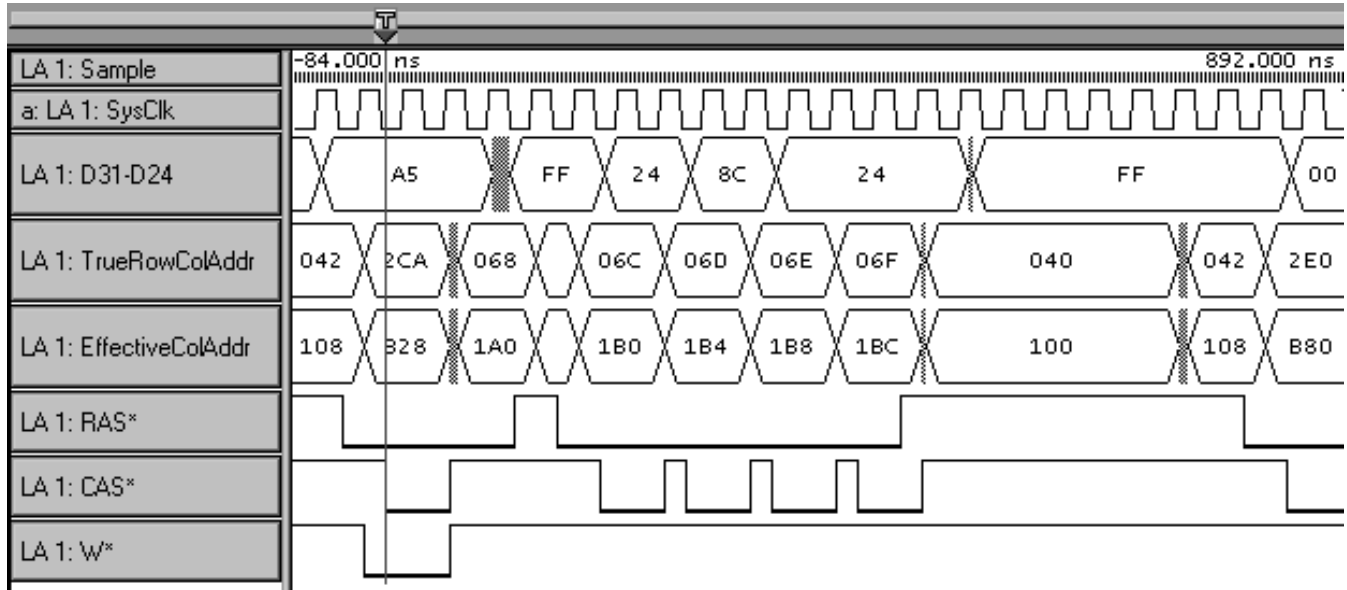
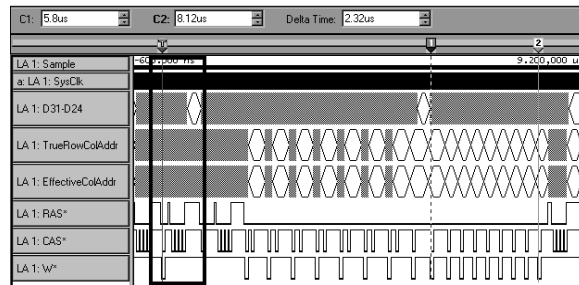


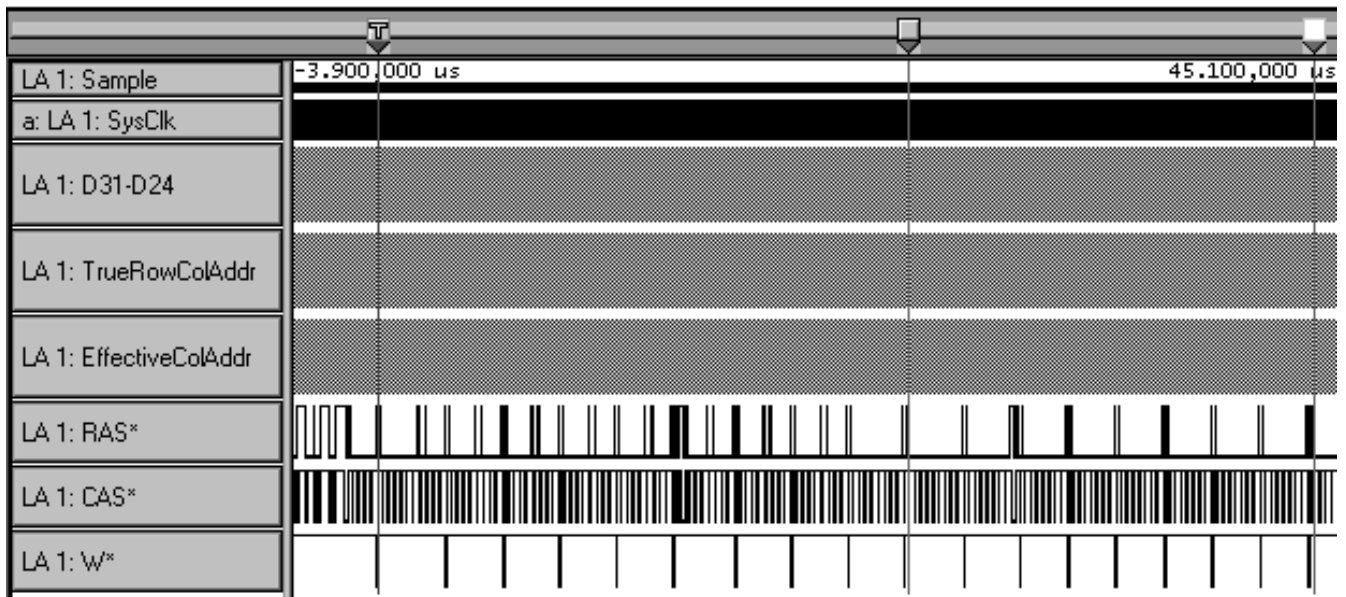
Figure 6: Closer inspection of first instruction cache miss

alyzers. Although ELEC470 is not officially a laboratory course at this time, there is an opportunity to introduce a practical component to the course and further reinforce important architectural and implementation concepts.

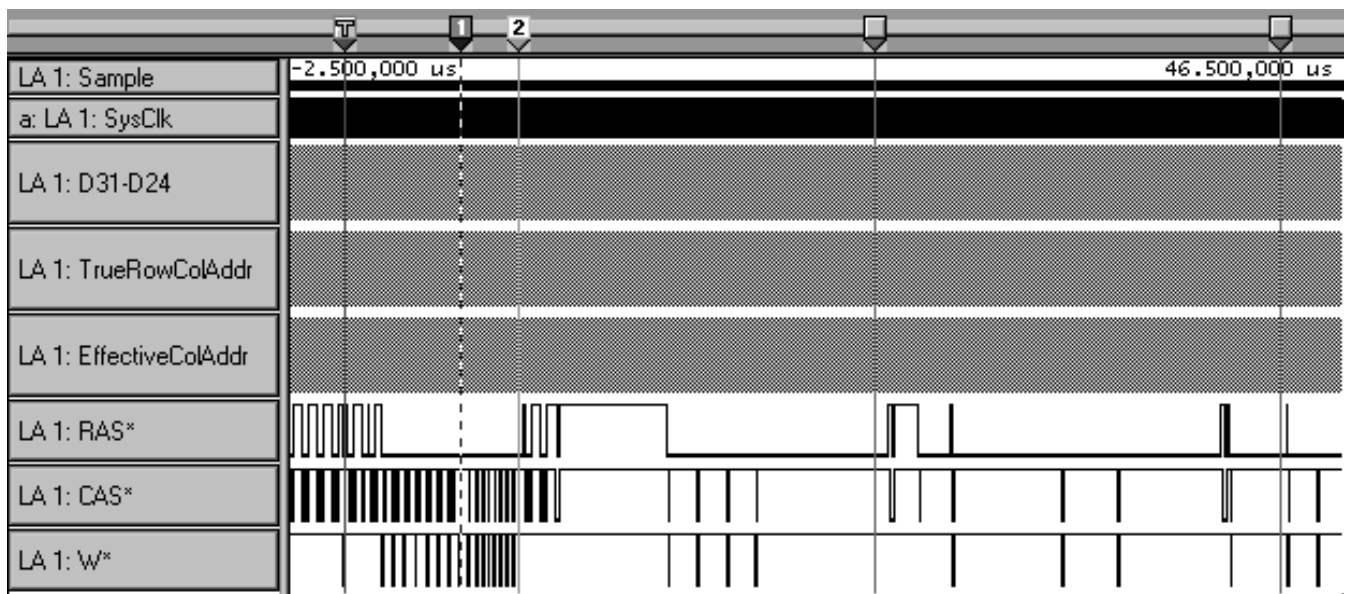
Links from the author's WWW site (<http://www.ece.queensu.ca/hpages/faculty/manjikian>) provide further information on the ELEC470 course.

## References

- [1] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Tech. Report 1342, Computer Sciences Department, University of Wisconsin-Madison, June 1997.
- [2] J. O. Hamblen. Using large CPLDs and FPGAs for prototyping and VGA video display generation in computer architecture design laboratories. *Newsletter of the IEEE Computer Society Technical Committee on Computer Architecture*, July 1999. Available at <http://computer.org/tab/tcca/NEWS/feb99/index.html>. Appeared in the 1998 Workshop on Computer Architecture Education.
- [3] J. L. Hennessy and D. A. Patterson. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, San Francisco, CA., 2nd edition, 1998.
- [4] Integrated Device Technology, Inc. *79S361ST "MIPS-Mini" Evaluation Board Hardware User's Manual — Version 1.0*. Santa Clara, CA., December 1997.
- [5] Integrated Device Technology, Inc. *IDT79RC36100 Highly Integrated RISController Hardware User's Manual — Version 2.1*. Santa Clara, CA., August 1998.
- [6] N. Manjikian and S. Simmons. Evolution and enhancements of a microprocessor systems course. *IEEE Transactions on Education*, 42(4), November 1999. Appears in special CD-ROM supplement with summary on p. 360.



(a) Acquisition for uncached execution



(b) Cached execution with markers positioned for comparison with uncached execution

Figure 7: Comparing uncached execution with cached execution

[7] A. K. Uht. The integrated computer engineering design (ICED) curriculum. *Newsletter of the IEEE Computer Society Technical Committee on Computer Architecture*, July 1999. Available at <http://computer.org/tab/tcca/NEWS/feb99/index.html>.

Appeared in the 1998 Workshop on Computer Architecture Education.