# IASSim: A Programmable Emulator for the Princeton IAS/Von Neumann Machine

Barry Fagin
Dept. of Computer Science, US Air Force Academy
2354 Fairchild Drive
USAFA, CO 80840
1-719-333-7377
barry.fagin@usafa.edu

Dale Skrien
Dept. of Computer Science, Colby College
5851 Mayflower Hill
Waterville, ME 04901-8858
1-207-859-5851
djskrien@colby.edu

## ABSTRACT
In this paper, we describe a programmable emulator for the Princeton IAS/Von Neumann machine. The emulator is historically accurate, preserving the quirks and eccentricities of the machine. It is also user-friendly and robust, suitable for undergraduate architecture and programming classes as a teaching tool. Users can write non-trivial programs in IAS assembly code or machine code. We present some examples here, and discuss assignments from its first use in two undergraduate classes. IASSim is a Java application publicly available at no cost.

## Categories and Subject Descriptors
C.0 [**Computer Systems Organization**]: General – *instruction set design, modeling of computer architecture.*

## General Terms
Measurement, Design, Standardization, Languages, Verification.

## Keywords
Emulation, IAS Machine, Von Neumann Architecture, Courseware, Computer Architecture.

## 1. INTRODUCTION
In 2009, the first author began development of a "Great Ideas in Computer Science" course, designed for a special academic program at the US Air Force Academy. Courses in this program were intended to engage students directly with the fundamental ideas and fundamental figures in the discipline under study. The "Great Ideas" chosen by the author, presented in historical order, were 1) Computation as calculation, 2) Computation and its limits as articulated by Turing, 3) Computation as a physically realizable idea, 4) Programming languages and software, 5) Computational complexity, 6) Cryptography, and 7) The future of computing. Reading materials chosen included Babbage's work on the Difference Engine [3], Turing's original paper on computation [12], Rivest, Shamir and Adelman's paper on public key cryptosystems [8] and so forth. All students read and discuss the original papers, and do "hands-on" assignments intended to acquaint them with the associated ideas as directly as possible.


*SIGCSE'11,* March 9–12, 2011, Dallas, Texas, USA.


For block (3), the obvious choice for reading material was Burks, Goldstine and Von Neumann's technical report [4]. So for the corresponding assignment a close encounter with the Princeton Institute for Advanced Study (IAS) computer seemed appropriate. Preliminary inquiries, however, indicated no emulators or comparable tools existed. Fortunately, the second author had developed an extendible microprogrammed simulator "CPU Sim" [10] that could implement a wide variety of architectures and instruction sets. This led to collaborative development of IASSim, a "frozen" offshoot of CPU Sim that faithfully models assembly language programming on the original IAS machine. This tool is described here.

## 2. PREVIOUS WORK
A large variety of computer simulators have been developed for educational purposes [6,7,11,14]. Such simulators have value in environments where resources are limited and the actual computers are not available for study. They are also valuable in introductory courses, in which the study and programming of real architectures would be too complex and time consuming.

Most simulators imitate a specific architecture. Some simulate very simple ones [9] that give beginning students a feel for how CPUs work. Others simulate real modern architectures, such as the MIPS machine [13]. Many such simulators are discussed in two issues of JERIC [1,2]. Simulators allow you to study the architecture of and write programs for computers that no longer physically exist, except possibly in museums. For example, there are simulators for the PDP-8 [2] and the EDSAC [5]. Unfortunately, no simple simulator could be found by the authors for the IAS architecture. This led to the creation of IASSim.

## 3. THE IAS MACHINE
The IAS machine is named after the Princeton Institute for Advanced Study, where it was developed by a team headed by the mathematician John Von Neumann. Von Neumann had met Alan Turing during the latter's short stay at Princeton, and was very interested in bringing Turing's ideas to life by building an actual computer. While there will always be controversy regarding who built the "first" computer, the technical report [4] describing the IAS machine is generally regarded as the first and most authoritative publication of the general ideas used in computers today. We therefore chose it and the machine it describes as the "Great Idea" that would best illustrate humanity's attempts to

**Table 1: The IAS Instruction Set**

| inst name | opcode | description |
|---|---|---|
| S(x)->Ac+ | 1 | copy the number in Selectron location x into AC |
| S(x)->Ac- | 2 | same as #1 but copy the negative of the number |
| S(x)->AcM | 3 | same as #1 but copy the absolute value |
| S(x)->Ac-M | 4 | same as #1 but subtract the absolute value |
| S(x)->Ah+ | 5 | add the number in Selectron location x into AC |
| S(x)->Ah- | 6 | subtract the number in Selectron location x from AC |
| S(x)->AhM | 7 | same as #5, but add the absolute value |
| S(x)->Ah-M | 8 | same as #7, but subtract the absolute value |
| S(x)->R | 9 | copy the number in Selectron location x into AR |
| R->A | 10 | copy the number in AR to AC |
| S(x)*R->A | 11 | Multiply the number in Selectron location x by the number in AR. Place the left half of the result in AC and the right half in AR. |
| A/S(x)->R | 12 | Divide the number in AC by the number in Selectron location x. Place the quotient in AR and the remainder in AC. |
| Cu->S(x) | 13 | Continue execution at the left-hand instruction of the pair at Selectron location x |
| Cu`->S(x) | 14 | Continue execution at the right-hand instruction of the pair at Selectron location x |
| Cc->S(x) | 15 | If the number in AC is >= 0, continue as in #13. Otherwise, continue normally. |
| Cc`->S(x) | 16 | If the number in AC is >= 0, continue as in #14. Otherwise, continue normally. |
| At->S(x) | 17 | Copy the number in AC to Selectron location x |
| Ap->S(x) | 18 | Replace the right-hand 12 bits of the left-hand instruction at Selectron location x by the right-hand 12 bits of AC |
| Ap`->S(x) | 19 | Same as above, but modifies right-hand instruction |
| L | 20 | Shift the number in AC to the left 1 bit (new bit on the right is 0) |
| R | 21 | Shift the number in AC to the right 1 bit (leftmost bit is copied) |
| halt | 0 | Halt the program (see paragraph 6.8.5 of IAS report) |

AC: Accumulator
AR: Arithmetic Register

first realize computation with actual machines. We used it as the guideline in designing IASSim.

Table 1 shows the instruction set of the IAS machine, as originally described in [4]. This is also the instruction set of IASSim. To recreate the "look and feel" of the original, and to emphasize the origins of programming in mathematics, IASSim preserves the opcode notation as described by the original authors. Original terminology is also preserved, for example, in the description of a memory cell as a "Selectron location". Opcodes are identical to those originally described in [4]. Contrary to present practice, the source of an instruction is on the left, destination on the right.

The IAS machine organizes memory into 4K words of 40 bits each. In a quote worth pointing out to students, the authors of the IAS architecture noted "this memory capacity exceeds the capacities required for most problems that one deals with at present by a factor of about 10" [4]. They chose a 40-bit word size since it gave about 12 decimal places of accuracy, "higher than what is required for the great majority of present day problems" [4]. In this machine, 40-bit integers are stored in 2's complement notation.

Floating point numbers were not supported in the IAS machine. It is worth pointing out and discussing Von Neumann's views on this topic [4] in the classroom. In a nutshell, Von Neumann believed that the costs were not worth the benefits. If you could not keep track of where the binary point belonged during your calculations, you had no business using a computer.

Instructions use 20 bits and so two instructions fit in one word. Halfwords that are used as instructions store the opcode in the left 8 bits, and use the remaining 12 as an address or instruction operand. Instructions and data are co-resident in memory, one of the fundamental contributions of this machine to computer science. Instructions are fetched one pair at a time, and executed from left to right within the word and from lower addresses to higher, unless branching interrupts the normal flow.

The IAS machine has two programmer-visible registers: The Accumulator (AC or A in the table) and the Arithmetic Register (AR or R in the table). Both are 40 bits wide. The AR is set only by a special load instruction or one of two arithmetic instructions; the AC is the more common target. Instructions 13 and 14 are unconditional branches, while 15 and 16 provide conditional branching. Instructions 18 and 19 are used for indexing, which the IAS machine achieves through self-modifying code.

The IAS machine does not include any instructions for handling I/O and so all data must be loaded into memory directly (either manually or via assembly pseudoinstructions) before the program is executed. IASSim includes an option for the programmer to specify the target load address.

The original IAS report does not specify a halt instruction. However, paragraph 6.8.5 states "There is one further order that the control needs to execute … which will tell the computer to stop and to flash a light or ring a bell." In that spirit, we have provided a HALT instruction with a zero opcode that will stop computation and ring a bell. Mercifully, the auditory component of this instruction can be disabled by the user.

```
; adds up the values n+...+3+2+1(+0) in a loop and stores
; the sum in memory at the location labeled "sum"

loop:
              S(x)->Ac+     n       ;load n into AC
              Cc->S(x)      pos     ;if AC >= 0, jump to pos
              halt                  ;otherwise done
              .empty                ;a 20-bit 0
pos:          S(x)->Ah+     sum     ;add n to the sum
              At->S(x)      sum     ;put total back at sum
              S(x)->Ac+     n       ;load n into AC
              S(x)->Ah-     one     ;decrement n
              At->S(x)      n       ;store decremented n
              Cu->S(x)      loop    ;go back and do it again

n:            .data 5               ;will loop 6 times total
one:          .data 1               ;constant for decrementing n
sum:          .data 0               ;where the running/final total is kept
```

**Figure 1: Summing numbers from 1 to n on the IAS Machine**

## 4. A SAMPLE IAS PROGRAM

The IAS machine had no assembly language or assembler, and so coding needed to be done in machine language. However, to simplify programming and to make it more useful as a teaching tool, we have provided a basic assembly language and assembler as part of IASSim.

Figure 1 shows how the IAS machine could be programmed to sum the numbers from 1 to *n*, where *n* is stored in a memory location. IASSim supports labeling of memory locations. Labels are terminated with a colon. Each instruction and operand corresponds to a 20-bit halfword. Comments are indicated with a semicolon.

The `.data` pseudoinstruction converts its operand to a 40-bit binary number and stores it at the current memory location. The `.empty` pseudoinstruction stores a 20-bit zero at the indicated halfword. Because all instruction fetches are 40 bits at a time, and because alignment errors are difficult for beginning students to catch and debug, the IASSim assembler enforces labels and branch targets to align with word boundaries. The `.empty` pseudoinstruction enables programs to comply with this requirement.

## 5. RUNNING PROGRAMS WITH IASSim

The opening screen of IASSim is shown in Figure 2. Memory contents appear in the large window on the left. Only a small number of 40-bit words are shown below, but the size of memory is user-configurable.

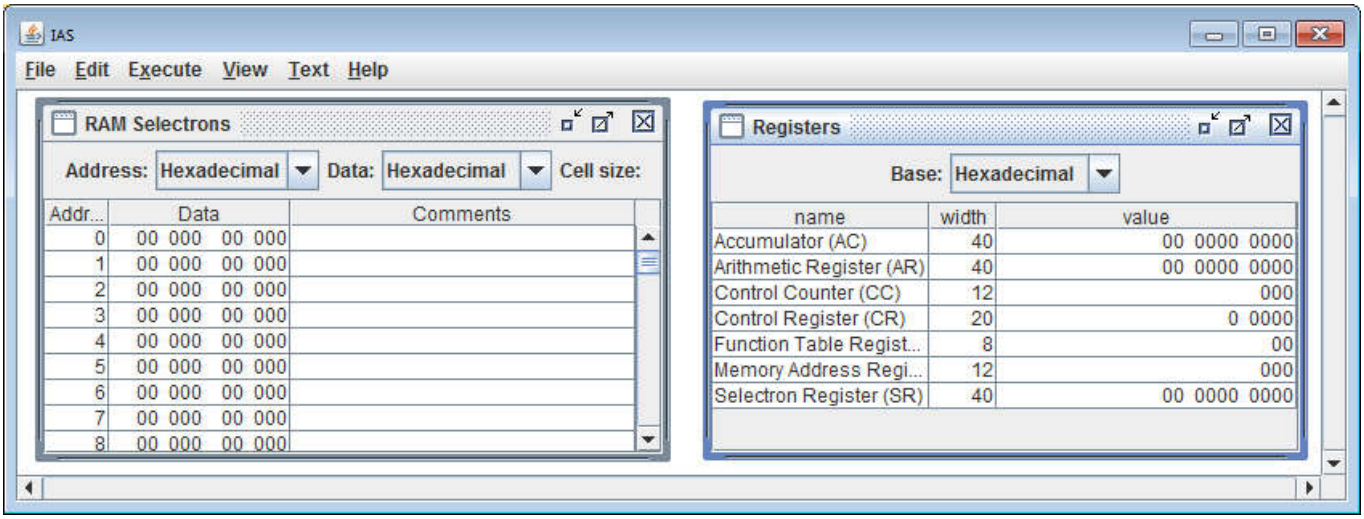



**Figure 2: Opening Screen of IASSim**

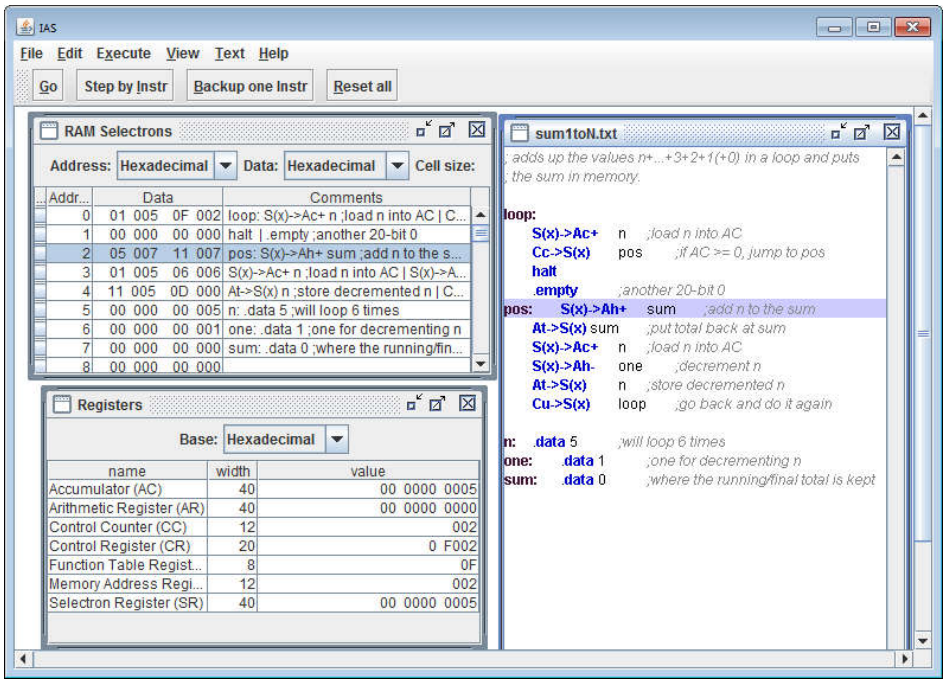



**Figure 3: After Executing the First Two Instructions**

In addition to the previously mentioned AC and AR registers, the Registers window also permits students to view some internals of the machine, as an aid to understanding and debugging. Consistent with our emphasis on historical accuracy, we employ the authors' original terminology. The Control Counter is what we now call the Program Counter. The Control Register holds the currently executing instruction. The Function Table Register holds the current opcode, the Memory Address Register the current memory address, and the Selectron Register the current data value being read from or written to memory.

Assembly language programs like that of Figure 1 can be created using either the built-in text editor or an external editor and stored as text files. The built-in editor has syntax highlighting specifically designed for IAS assembly language.

Program windows can be opened from the "File" menu, and then assembled and loaded from the "Execute" menu. The RAM Selectrons window shows each instruction halfword loaded into memory, with the upper eight bits of opcode visually separated as an aid to learning.

Placing the simulator into "Debug" mode enables setting break points and allows single step execution of the program forward or backward. After two instructions are executed, the resulting screen is shown in Figure 3. Note that the value of *n* (5) is now loaded into the accumulator. Since the contents of the accumulator are positive, execution will continue at the instruction labeled "pos:" This can be verified by examining the value of the Control Counter. The instruction at the branch target has not yet been fetched, so the Control Register still holds the previous instruction executed. When in debug mode, the next instruction to be executed is always highlighted in the program window and the Rams Selectron Window

Programs may be executed to completion at any time by pressing "Go". At the end of execution, 15 (the sum of the numbers from 1 to 5) is stored in the memory location corresponding to the label "sum:", the Control Counter holds the address of the halt instruction, and the simulator highlights the empty halfword after it.

## 6. IASSim TECHNICAL DETAILS

IASSim is based on CPU Sim [10], a Java application that the second author created for use in introductory computer organization classes. It was modeled on the STARTLE simulator developed by J. Kerridge and N. Willis [7]. CPU Sim was designed to simulate many different CPU architectures, including simple accumulator-based architectures, RISC architectures, and stack-based architectures such as the JVM.

When using CPU Sim, the user can specify a previously saved architecture or can create a new one. In the latter case, the user specifies the CPU features such as the number and width of registers, the size of main memory, the instruction set, and the layout and implementation of the instruction set using microinstructions. Once a CPU architecture has been specified, the user can write and run assembly language and machine language programs for that architecture within the CPU Sim environment.

CPU Sim provides standard debugging tools such as break points and stepping forward and backward through the code to inspect and optionally change the values in registers or memory at any point.

IASSim was created by modifying CPU Sim to simulate only the IAS architecture. Quite a few modifications were necessary because of the IAS's non-standard features, such as 40-bit words containing either a 40-bit data value or two 20-bit instructions. To keep IASSim as simple as possible, many menus and menu items used in CPU Sim to allow the user to modify the architecture were removed from IASSim.

# 7. TEACHING INSIGHTS

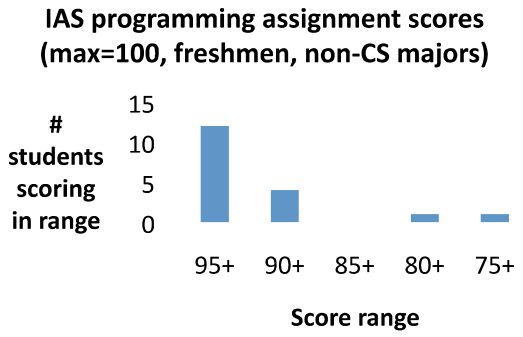
## 7.1 Using IASSim in a "Great Ideas" Course

As mentioned previously, the first author used IASSim in a "Great Ideas" computer science course, taught to freshman cadets at the US Air Force Academy. Although most of these students had some computer programming in high school, the course had no formal prerequisites other than admission into the USAFA Scholars Program, a program designed to identify the top forty or so cadets in each class based on their academic potential. The course is not required for the computer science major, and although any student enrolled may declare the CS major in their sophomore year (the typical time for cadets to declare), it is expected that most will not.

Students in this course read the original IAS tech report [4], and then used IASSim as a vehicle to explore humanity's first attempts to build actual prototypes of Turing's Universal Computing Machine [12]. Students had written their own Turing machines using a tool developed at USAFA, designed to follow Turing's notation and style as much as possible. Although many students had taken some computer programming in high school, for some their construction of Turing Machines was their only programming background prior to tackling this assignment.

Nonetheless, all students successfully completed ten IAS assembly language programming assignments. These included summing two or three numbers, finding the maximum of two or three numbers, polynomial evaluation, adding a scalar to a vector, multiplying a vector by a scalar, primality testing, and using Von Neumann's algorithm in [4] to calculate reciprocals.

Students reported a range of 3 to 15 hours of effort to complete all 10 problems, with an average of around 7 hours. The class average was 95%, with three perfect submissions. A histogram of their performance on the IAS programming task is shown below.

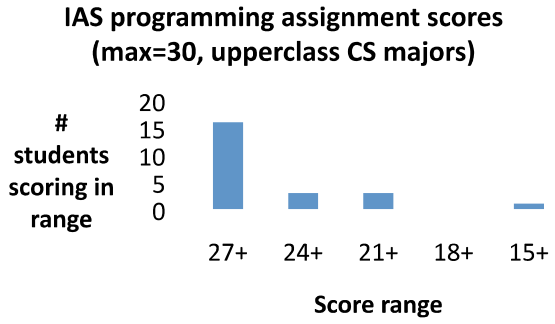



These results should be interpreted with caution. Students in the Academy Scholars program are an extremely bright and motivated group. Nonetheless, the fact that a group of college freshmen with a wide variety of academic interests were able to perform so well in writing assembly language programs for a 60-year-old computer suggests that IASSim is a useful and effective teaching tool.

## 7.2 Using IASSim in a CS Course

The second author used IASSim in his computer organization class in the spring of 2010. The 23 students had already been using CPU Sim for a month before the IASSim assignment was given, and so they felt very comfortable with the IASSim package. A brief (6 paragraph) introduction to the IAS architecture was given and a sample program was provided that found the larger of two integers. Then the students were given assignments to find the largest of 3 integers, evaluate a cubic polynomial, and copy a value into a series of consecutive locations in memory.

Scores on this assignment are shown below:




The students were asked to keep track of the amount of time it took them to complete these assignments. The average was about 4 hours for all three assignments, with the times ranging from 1.5 hours to 7 hours. These numbers were not significantly out of line with previous and later CPU Sim assignments. It appears that, because of previous exposure to the CPU Sim package, simple accumulator based architectures, and self-modifying code, the students took the IASSim assignments in stride with no complaints and no significant difficulties.

## 7.3 Educational Impact

Although the tool is still fairly new and has only been used in two courses at two institutions, we have some evidence of its educational utility.

We have, for example, performance data from IASSim's use on graded tasks. The results clearly show that students (albeit in some cases very bright and highly motivated ones) can use a 60-year-old instruction set architecture to accomplish programming tasks, and can do so with a reasonable amount of effort.

Qualitatively, we have also seen some evidence of pedagogical value in using IASSim. Students who work with a "real" architecture that is nonetheless limited to 12-bit addresses gain an immediate and practical insight into the relationship between address space and computational power. The awkwardness of the "left" and "right" instructions in the IAS also helps students appreciate modern architectures and provides an appreciation for simplicity and regularity in engineering design.

One particularly unconventional feature warrants special mention. Memory addresses in the IAS architecture can only come from instruction fields (it has no index register). Since addresses for array accesses are not normally known at compile

time, array indexing requires the address fields of instructions to change while the program runs. Hence if you are working with vectors on the IAS machine (as the original designers did), you must employ self-modifying code.

We believe challenging students with a programming problem for which the only answer is self-modifying code provides a very useful historical and technical perspective. The historical perspective illustrates that even towering intellectual figures don't always get the right answer the first time around. Sometimes, only the experience of working with a system will show a better way to solve a problem.

The technical perspective is relevant today. Self-modifying code is so frowned-upon as a programming practice that it is seldom, if ever taught, and never to beginning programmers. However, virtually all of the most sophisticated viruses work by modifying their own instructions. Students who have written at least some simple self-modifying assembly language routines will have a much better understanding of how viruses work once they encounter them. Particularly given the profession's increased interest in teaching computer security issues more vigorously and earlier in the computing curriculum, we believe this insight to be particularly useful.

Finally, the IAS ISA can start a good discussion about hardware/software tradeoffs in computer architecture. What did the designers do at the gate level, and what did they leave for software? What technology was available to them, and how did that influence their decision? The answers are different today, but the questions, as always, remain the same.

We hope other faculty will use IASSim and report their experiences to us.

# 8. CONCLUSIONS AND FUTURE WORK

For faculty and students with experience in assembly language programming, programming with IASSim is a trip back in time. You find yourself bemoaning the existence of a single register, grimacing at the extra memory traffic, and wistfully longing for an index register. Writing self-modifying code to accomplish vector processing definitely takes some getting used to.

But for students with no background in assembly, the small instruction set and simplicity of the architecture make it an attractive first platform, independent of its historic interest. The behavior of each instruction is clearly defined, and there is seldom more than one way to accomplish a task, so even if the program seems inefficient by modern standards the correct solution is discoverable by a sufficiently bright and motivated student.

In the future, IASSim could also serve as a useful tool to illustrate numerous issues in high-level language programming. Shortly after submission of this paper, the first author developed JVNTRAN, a very simple high-level language, and an accompanying compiler that generates IAS assembly code. This compiler is being integrated into IASSim, with the eventual goal of supporting JVNTRAN/IASSim classroom assignments.

Beginning programmers who must solve programming problems in both assembly language and a higher level language could learn first-hand about the forces that led to the emergence of high level languages in the first place, an important goal in a "Great Ideas" course. More advanced students could be assigned the task of writing their own compiler for JVNTRAN. Speaking from experience, the authors can attest that they will learn in dramatic fashion the impact of architectural decisions on compiler design!

IASSim is available for download at http://www.cs.colby.edu/djskrien/IASSim/. Source code will be freely provided to anyone interested in inspecting or improving it. We hope a user community will develop that will find bugs, report experiences, develop assignments, and suggest features for future releases.

# 9. ACKNOWLEDGMENTS

The authors wish to thank our students in "Computer Organization" (Colby) and "Introduction to Computing, Scholars Section" (USAFA) for their service as beta testers.

# 10. REFERENCES

[1] *ACM Journal of Educational Resources in Computing (JERIC),* Vol 1, No. 4, December 2001.

[2] *ACM Journal of Educational Resources in Computing (JERIC),* Vol. 2, No. 1, March 2002.

[3] Lardner, D., "Babbage's Calculating Engine", *The Edinburgh Review,* Vol. 59, 1834, pp 263-327.

[4] Burks, A. et. al., "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", Report to US Army Ordnance Department, 1946.

[5] Campbell-Kelly, M. "The EDSAC Simulator", http://www.dcs.warwick.ac.uk/~edsac/ [accessed July, '10]

[6] Grunbacher, H., "Teaching Computer Architecture/ Organization Using Simulators," *IEEE Frontiers in Education Conference (FIE),* 1998, pp 1107-1112.

[7] Kerridge, J.M. and Willis, N., "A Simulator for Teaching Computer Architecture," *ACM SIGCSE Bulletin,* Vol. 12, No. 2, July 1980, pp 65-71.

[8] Rivest, R. et. al., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM* **21** (2): pp 120-126.

[9] Schweitzer, D. and Boleng, J., "A Simple Machine Simulator for Teaching Stack Frames", *Proceedings of the 41st SIGCSE Technical Symposium on Computer Science Education,* September 2010, pp 361-365.

[10] Skrien, D., "CPU Sim 3.1: A Tool for Simulating Computer Architectures for Computer Organization Classes", *ACM Journal of Educational Resources in Computing,* Vol. 1, No. 4, December 2001, pp 46-59.

[11] Tangorra F., "The Role of the Computer Architecture Simulator in the Laboratory," *ACM SIGCSE Bulletin,* Vol. 22, No. 2, June 1990, pp. 5-10.

[12] Turing, A, "On Computable Numbers, With an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society* **42**: pp 230–65. 1937.

[13] Vollmar, K. and Sanderson, P., "MARS: An Education-Oriented MIPS Assembly Language Simulator," *ACM SIGCSE Bulletin*, Vol. 38, No. 1, March 2006, pp 239-243.

[14] Wolffe, G. S. et. al., "Teaching Computer Organization/Architecture with Limited Resources Using Simulators", *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, September 2002, pp 176-180.