

UAV Communications: Integrating a Real-World Scenario with Computer Architecture

Chris Augeri¹, Danial Neebel², Leemon Baird³, and Adrian de Freitas⁴

Abstract – A challenge facing many educators is providing assignments in a realistic context that achieve the specified learning objectives. Integrating real-world scenarios in one’s curriculum can be challenging. We present a new integrated exercise sequence using unmanned aerial vehicles (UAVs) involving both assembly language and high-level language software development. During this sequence, our sophomore students at the U.S. Air Force Academy (USFA) implement a communications packet-based protocol for a simulated UAV system.

The first exercise is an introduction to assembly language programming, involving user input/output and integer-based instructions. The second exercise adds the use of assembly language floating point instructions. To simulate radio transmission of data from the UAV to a ground control station (GCS), the third exercise introduces command-line pipes. The GCS is implemented in a high-level-language and consists primarily of an IEEE 754 software multiplier. Memory traces from these three exercises are used during their final team project, implementing a visual cache simulator.

This new UAV-based computer architecture assignment meets an institutional goal of having career-related assignments in each course. Our institution has a strong inter-disciplinary UAV research group, which a member of our department directs and from which this sequence is derived. This sequence prepares students for the senior-year UAV-based software engineering capstone.

Index Terms - Unmanned Aerial Vehicles, UAVs, Computer Architecture, Education

INTRODUCTION

Our students study computer architecture (CS 351) in the spring semester during their sophomore year. These students have declared the computer science major and have typically completed three major-relevant courses – *Introduction to Computer Science* (CS 110) – taken by all incoming students, *Computer Programming I* (CS 210), and *Introductory Digital Systems* (EE 281). EE 281 is offered by the Electrical Engineering department. Our students concurrently enroll in a data structures course, CS 220. Two years ago, the CS 351 course was re-sequenced from being taken by our fall semester juniors to our spring semester sophomores.

CS 351 has two prerequisite courses, the CS 110 and EE 281 courses identified previously. The CS 110 course provides instruction in developing algorithmic thinking and information system technology concepts [20]. Beginning with the Summer 2003 offering, students are first introduced to programming using the department’s newly developed tool, RAPTOR [5]. CS 110 closely supports one of the institution’s seven educational outcomes, the ability to “frame and resolve ill-defined problems” [10]. EE 281 introduces Boolean logic, two’s complement arithmetic, and digital circuits.

The CS 210/220 course sequence introduces students to Ada, our core programming language, data structures, object-oriented design, and graphical user interfaces (GUIs). Subsequent courses building on computer architecture include operating systems, networks, and compilers.

Our material is designed to meet ACM 2001 curriculum guidelines [8]. The computer architecture requirements are approximately split 30-50-10-5-5 (%) between the digital systems, computer architecture, operating systems, networks and compiler courses respectively. We use three textbooks in CS 351: a MIPS-based computer architecture text, a MIPS assembly handbook, and a computer ethics text [2, 24, 26].

Table I identifies the course topic timing and coverage. The assembly language and integer operation lessons are interleaved; the remaining lessons are introduced in the order shown. Material from the ethics text is woven throughout the four-year curriculum of the major. Ethics topics introduced in our computer architecture course include manufacturer recalls, benchmark relevance, and environmental impacts of computer hardware. The ethics material is designed to support the most important goal of the institution, character development.

TABLE I
COURSE TOPICS IN INTRODUCED ORDER

Topic	Coverage Hours
MIPS Assembly Language	6
Integer Operations	4
Performance Analysis	3
Floating Point Operations	3
Single/Multi-Cycle Data Paths	3
Pipelines	4
Cache Memory	4
High-Performance Computing	3
*Ethics	1
*Security	1
*Interrupts/Exceptions	1

*Material introduced in segments throughout course.

¹ Christopher J. Augeri, Air Force Institute of Technology, chris.augeri@afit.edu

² Danial J. Neebel, Loras College, danial.neebel@loras.edu

³ Leemon C. Baird III, United States Air Force Academy, leemon.baird@usafa.edu

⁴ Adrian de Freitas, United States Air Force Academy, c06adrian.defreitas@usafa.edu

Course assignments consist of five problem sets, four programming exercises (PEXes), two in-class exams and a comprehensive multiple-choice final. Each assignment is individual effort except for the final programming exercise (PEX) which is a 3-person team project. Students are permitted to compare answers and discuss approaches on the homework and programming exercises, but must submit their own work and identify any help received.

To introduce the MIPS assembly language, we provide five in-class programming exercise laboratory lessons. For these lessons, students bring their personal laptops to class and we introduce the primary software tools used, SPIM and MIPster [12, 16]. Students may also use a preferred text editor in lieu of the MIPster package for development. Students will use both the command-line and graphical versions of SPIM during the course. Laboratory exercises include exposure to instruction formats, procedure calls, addressing modes, interrupt/exception handling, and pseudo-instructions. To prepare students for their required senior-year compiler course, we also demonstrate cross-compilation with a simple C program compiled to Intel, MIPS, and PowerPC assembly language source code.

INTEGRATING UNMANNED AERIAL VEHICLES (UAVS)

Given our student population, we have a keen interest in developing assignments related to the primary mission of the U.S. Air Force. Many departments are involved in research applications related to UAVs and the UAV research group (UAVRG) director is a computer science department member. While UAVs have military applications, they also have civilian applications, e.g., monitoring crop growth and city traffic, and similar research can be found at other institutions [14, 15, 18]. Thus, we believe this sequence can be modified for use at both military and civilian academic institutions.

The UAVRG spans all four of our educational divisions (science, engineering, humanities, social sciences) at USAFA. We actively collaborate with and receive funding from the U.S. Air Force, Department of Defense, other government agencies and a local company developing UAV technology. As part of the institution's new system engineering program, we also established a course centered on UAVs. A recent offering's emphasis for this new course was developing a tree beetle infestation tracking system for the U.S. Forestry Service. We also have a growing fleet of actively flown radio-control aircraft and UAVs. Other projects include developing ground control station software, studying aeronautical/engine performance, and improving swarming algorithms [11, 19].

Integrating real-world projects is a challenging, yet rewarding concept noted by other educators [23, 25]. Our challenge was finding a relevant UAV application that we could use in computer architecture [7, 17]. We settled on UAV and ground control station (GCS) telemetry communications. This permitted us to focus on integer and floating point operations along with data manipulation issues. While the concepts were drawn from an active UAV communications project, for security reasons we created a fictitious UAV for the two offerings we have used this sequence, the Hummingbird and the Raven.

Our computer architecture course historically has four assignments presented in the following order: integer operations, stack usage, IEEE 754 operations, and cache performance [1, 2]. The assignments were independent and involved programs such as a recursive primality tester, and developing IEEE 754 operations in software with a high-level language. The cache analysis exercise was a command-line only program. Previously, extra credit on these assignments was via early submission. For this sequence, PEX 1 and 2 have early submission only. PEX 3 has early submission and an extra implementation requirement. PEX 4, given its team approach, only has an implementation option.

We had to create functionally similar assignments in the UAV realm to compare results with historical data. Given this constraint, we created a sequence based on integer operations, floating point operations, IEEE 754 multiplication, and a cache simulator with a graphical user interface (GUI). This is an integrated sequence – a student must use previous assignments as building blocks for subsequent ones. While students were permitted to assist others on PEXes 2-4, each student worked independently on PEX 1. This facilitated early instructor assessment of each student's progress.

Each assignment had a solution template or a key from the previous assignment available. Solution templates were provided electronically, but keys were distributed only by appointment and then only on paper. We conjectured student learning increased by transcribing the solution, especially given the subtleties of assembly language development.

For programs requiring a MIPS-only assembly language solution a spreadsheet-based simulator was provided so students could verify various input/output combinations. We provided a pre-compiled executable for assignments requiring development in Ada. In addition, one-on-one assistance from the instructor was available and various tips were offered in class during the course of an individual assignment. Table II provides a comparison of the programming assignments.

TABLE II
PROGRAMMING EXERCISE COMPARISON

PEX	1	2	3	4
Independent Effort	Yes	Yes	Yes	No
Primary Focus	Integer Ops	Floating Point (FP) Ops	IEEE 754 FP Format	Cache Memory
Key Concepts	Text I/O Byte-Level Fields Data Integrity	Integer – FP Conversions Bounds Checking Stack Usage	FP Multiplication Array Manipulation Command-Line Filters	Data Collection Performance Analysis GUI Handling
Programming Tasks	Parse user input XOR compute loop Byte Shifting	Pass data to coprocessor FP math {*, +, /, -} Stack pointer (\$sp) usage	Ada records Unchecked Conversion Filtering {<, >, >>, !}	Ada bit manipulation GUI widget updates LRU history tracking
Extra Credit	Early Turn-In	Early Turn-In	Early Turn-In / Add'l Req't	Additional Requirement

October 19 – 22, 2005, Indianapolis, IN

With this sequence, we exposed students to a number of assembly language and interface issues. Students were able to use nearly all of the integer-based operations, including byte, half-word, and full-word variants. In addition to integer instructions, students handled alignment and data field manipulation issues. Input and output were done using SPIM simulated system calls, array data and command-line pipes.

ASSIGNMENT DESIGN

For PEX 1 and 2, students must prompt the user for various data elements. They then develop assembly language code to merge this data into packets. The hexadecimal packet display function is provided to them as part of the source code template. Students must also implement a byte-level XOR checksum in the telemetry packet. While SPIM assumes the underlying endian order of the system it is executing on, this protocol forces students to recognize the endian order, especially in the XOR function. Tables III and IV provide the 16-byte packet format specified for the Hummingbird UAV.

TABLE III
PACKET FORMAT (PEX 1), SPRING 2004

Packet Offset	00		
Field Name	Mask	Reserved	Altitude (m)
Range	[0, 255]	N/A	[0, 500]
# Bytes	1	1	2

Packet Offset	01		
Field Name	Lat (deg)	Lat (min)	Lat (sec)
Range	[-90, 90]	[0, 59]	[0, 59]
# Bytes	2	1	1

Packet Offset	10		
Field Name	Long (deg)	Long (min)	Long (sec)
Range	[-180, 180]	[0, 59]	[0, 59]
# Bytes	2	1	1

Packet Offset	11		
Field Name	Fuel (mL)	Reserved	Checksum
Range	[0, 250]	N/A	[0, 255]
# Bytes	1	2	1

TABLE IV
PACKET FORMAT (PEX 2-4), SPRING 2004

Packet Offset	00		
Field Name	IFF ID	IFF Code	Altitude (m)
Range	[0, 255]	[0, 255]	[0, 500]
# Bytes	1	1	2

Packet Offset	01		
Field Name	Latitude (degrees)		
Range	[-90.0, 90.0]		
# Bytes	4 (IEEE 754)		

Packet Offset	10		
Field Name	Longitude (degrees)		
Range	[-180.0, 180.0]		
# Bytes	4 (IEEE 754)		

Packet Offset	11			
Field Name	Fuel (mL)	TAS (m/s)	RTB (mL)	Checksum
Range	[0, 250]	[0, 127]	[0, 250]	[0, 255]
# Bytes	1	1	1	1

We intended on retaining a similar packet structure across the quarter and changing only the latitude / longitude fields from integer to IEEE 754 format. However, students felt the field mask requirement in PEX 1 made the assignment overly challenging. This issue may be due to inadequate classroom coverage. We also added an Identification Friend or Foe (IFF) code to reinforce the data integrity issue. Both changes were integrated with PEX 2 of the first offering. Each packet consists of sixteen bytes, or four 32-bit MIPS words.

Danial Neebel joined the department faculty in 2004 through the Distinguished Visiting Professor program and directed the Spring 2005 offering. Based on student and teacher feedback, some changes were made to the packet formats. Most notably, the field mask in PEX 1 was removed and the IFF ID is now an octal 4-digit value, as used for real IFF beacons [21]. Tables V and VI show these changes. The TAS field gives the true air speed of the UAV and the RTB field is a calculated excess fuel level if an immediate return-to-base command were to be issued. The value is cross-checked when received by the software IEEE 754 multiplier.

TABLE V
UAV PACKET FORMAT, PEX 1, SPRING 2005

Packet Offset	00		
Field Name	Reserved	IFF	Altitude (m)
Range	N/A	[0000, 7777] ₈	[0, 500]
# Bits	4	12	16

Packet Offset	01		
Field Name	Lat (deg)	Lat (min)	Lat (sec)
Range	[-90, 90]	[0, 59]	[0, 59]
# Bytes	2	1	1

Packet Offset	10		
Field Name	Long (deg)	Long (min)	Long (sec)
Range	[-180, 180]	[0, 59]	[0, 59]
# Bytes	2	1	1

Packet Offset	11		
Field Name	Fuel (mL)	Reserved	Checksum
Range	[0, 250]	N/A	[0, 255]
# Bytes	1	2	1

TABLE VI
UAV PACKET FORMAT, PEX 2-4, SPRING 2005

Packet Offset	00		
Field Name	Reserved	IFF	Altitude (m)
Range	N/A	[0000, 7777] ₈	[0, 500]
# Bits	4	8	16

Packet Offset	01		
Field Name	Latitude (degrees)		
Range	[-90.0, 90.0]		
# Bytes	4 (IEEE 754)		

Packet Offset	10		
Field Name	Longitude (degrees)		
Range	[-180.0, 180.0]		
# Bytes	4 (IEEE 754)		

Packet Offset	11			
Field Name	Fuel (mL)	TAS (m/s)	RTB (mL)	Checksum
Range	[0, 250]	[0, 127]	[0, 250]	[0, 255]
# Bytes	1	1	1	1

A thematic element retained across the sequence was for students to be adept in their ability to conduct bit-level data field manipulations. Throughout the semester, we reinforce that two's complement, IEEE 754 and instruction formats are just unique ways of assigning fields to a given bit string.

As Tables III – VI show, our packet fields were simply one or more bits (bytes) in a 32-bit MIPS word. Students will later be introduced to higher-level applications requiring bit manipulations, e.g., shifts and field masks. Applications include the formation of TCP/IP packets and image file reading/writing. Thus, it is crucial that students have a solid understanding of bit manipulation techniques, both for computer architecture and for future courses.

The department also encourages inclusion of computer security concepts within each course. Adding the XOR and IFF ID functions provided the opportunity to directly include security material in the course. We also integrated a reading and homework assignment later in the course relating to buffer overflow issues related to stack management.

Figures 1 and 2 depict user interaction for PEX 1 and 2 respectively. Please note the latitude and longitude values are integers in PEX 1 and IEEE 754 values in PEX 2 when displayed in their hexadecimal form. Thus, the two center packet words are different, though user input is identical. This change is a warm-up for PEX 3's IEEE 754 implementation. The hexadecimal display function is provided to the students.

```
(E)ncode, (D) display format, (Q)uit? E
Field mask value: 11111111
UAV altitude (m): 57
UAV latitude (degrees): 89
UAV latitude (minutes): 13
UAV latitude (seconds): 44
UAV longitude (degrees): 127
UAV longitude (minutes): 34
UAV longitude (seconds): 22
UAV remaining fuel (mL): 50
Packet: FF000039 00590D2C 007F2216 320000C7
```

FIGURE 1

PEX 1: MIPS ASSEMBLY LANGUAGE USER INTERFACE

During PEX 2, students are introduced to MIPS assembly language floating point conversions, e.g., mtc1, lwc1, cvt.s.w and l.s. They must convert the latitude and longitude entered as degrees, minutes, seconds integer values to floating point degree values. This better prepares students for PEX 3, where they must implement a software IEEE 754 multiplier in Ada.

As part of PEX 3 students also further modify their assembly language program from PEX 2 [4, 12, 16]. In previous offerings, students implemented a software IEEE 754 adder and multiplier during PEX 3. We replaced the adder requirement with a modification to PEX 2 that has students use the command-line version of SPIM via operating system pipes. This exposes students to pipes and filters at an earlier stage in the curriculum than done previously.

The only other major change was modifying the extra credit option from being an early submission requirement for PEX 1/2 to a simple GUI showing UAV movement for PEX 3 [6, 9]. Students are being introduced to GUI development in their data structures course, CS 220, taken concurrently.

Due to advances in (GUIs), today's students are often not familiar with command-line pipes and filter techniques. We sought to introduce this potentially challenging concept earlier in our program and this PEX sequence provided us the opportunity. In PEX 3, students modify their PEX 2 solution to generate packets via the SPIM command-line version. These packets are then piped to a concurrently executing student-modified Ada shell to verify the incoming data using an IEEE 754 software multiplier that students develop. The command-line pipe simulates air-to-ground RF transmissions and the Ada program represents a simple GCS monitor.

Using the provided latitude and longitude, students develop a routine to compute the distance to the GCS. Based on the current fuel load on the UAV and assuming it travels at maximum speed, they estimate the excess fuel the UAV will have after executing an immediate return-to-base command. This result is transmitted as part of the UAV telemetry packet and is cross-checked by the GCS via the IEEE 754 software multiplier. The RTB function is common to many UAVs.

```
(E)ncode, (D)isplay format, (Q)uit? E
UAV IFF transponder ID: 17
UAV IFF transponder code: 217
UAV altitude: 57
UAV latitude (degrees): 89
UAV latitude (minutes): 13
UAV latitude (seconds): 44
UAV degrees of longitude: 127
UAV minutes of longitude: 34
UAV seconds of longitude: 22
UAV remaining fuel (mL): 99
UAV TAS (Km/hr): 33
UAV distance from the GCS (m): 31
Packet: 11D90039 42B27530 42FF2543 3221FF73
```

FIGURE 2

PEX 2: MIPS ASSEMBLY LANGUAGE USER INTERFACE

Students were permitted to use standard multiplication for development and debugging, but the submitted solution must implement IEEE 754 multiplication in software for full credit. To motivate the software implementation of an IEEE 754 multiplier, we cited a need for fault tolerance and redundancy if the hardware floating point unit were to fail while the UAV is airborne.

Note, when distributing a new PEX, ample class time was allocated to review the "big picture" – what students had implemented thus far and what they would be implementing in the upcoming PEX. This provided several opportunities during the course of the semester to tie the PEXes together. While only the first three PEXes were directly UAV-related, we will briefly note how the fourth team-based exercise was indirectly connected – additional details are in [22].

October 19 – 22, 2005, Indianapolis, IN

Previously, students only analyzed instructor-provided traces. We provided both these traces and scripts for students to collect their own traces from the MIPS and Ada programs of the earlier PEXes. The provided scripts made use of both SPIM's command-line tracing capability and a debugger for the Ada compiler. In addition to increasing student motivation, it also introduced the concept of profiling and data collection to the students. Students used the "best" PEX 3 program from their team during PEX 4 to collect these traces. This assignment task further increased students' exposure to command-line filter facilities early in their academic career. In addition, the Ada trace facility introduces them to debugger usage at a relatively early stage of their development.

For PEX 4, students worked in groups of three to develop a GUI-based cache simulator. As with previous PEXes, we provided a template that each team modified. The template was based on the same Ada packages are exposed to in their concurrently-taken data structures course [4, 7, 10]. While the cache analysis was similar to previous offerings, the GUI component was new, and hopefully more instructional. The GUI allowed students to view the contents of the tag, set, block offset and word offset fields, visually display an active cache (hits, misses, or collisions), and the overall performance. Figure 3 depicts a sample of the main window for this assignment. Again, PEX 4 is discussed further in [22].

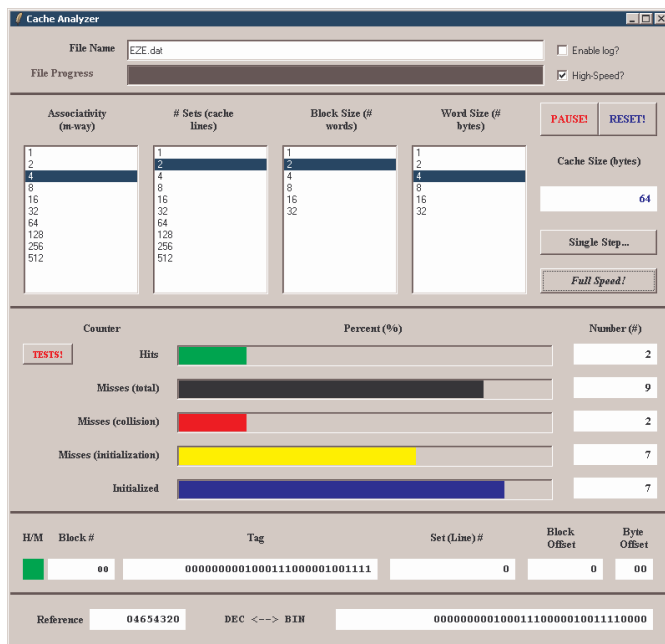


FIGURE 3
PEX 4: ADA-BASED CACHE SIMULATOR

RESULTS

We have used this sequence for two offerings, Spring 2004 and Spring 2005. While the assignments have changed, we include results with archival data as the general tempo, assignment load and point value has remained the same. As Table VII indicates, we average 23 students who take this course in a given offering. Each offering is taught by 1-2 instructors with 1-2 sections per offering. It is worth noting

nearly all classes at our institution are restricted to 24 students per section, providing an excellent student-to-teacher ratio. This greatly increases the opportunity for in-class interaction.

TABLE VII
GENERAL COURSE INFORMATION

Measure	N	GPA _{avg}	GPA _{major}	Avg	Profs	Sections
Spring 05*	30	3.020	3.084	75.0	2	2
Spring 04	18	2.844	3.329	77.8	1	1
Spring 03*	25	2.910	3.195	74.5	2	2
Fall 02**	22	2.959	3.120	74.6	2	2
Cumulative	23	2.941	3.170	75.3	2	2

* Change in course director
** Last offering to fall juniors, subsequently fall sophomores

Table VIII lists the student averages over the last four offerings. The combined programming exercise score forms 25% (250 / 1000 points) of the final grade. Table IX quantifies PEX complexity by lines of code. Students were given either a template when an assignment is distributed or have access to a hard-copy key from previous PEXes (for the UAV sequence). Previously, only a template was available. Scores are based on students who complete the course.

TABLE VIII
PEX AVERAGE SCORES (%)

PEX	1	2	3	Cumulative
Spring 2005	90	93	70	84
Spring 2004	88	86	72	82
Spring 2003	85	89	70	81
Fall 2002	85	83	83	84
Cumulative	87	88	73	83

Scores on PEXes 1-3 are comparable or better in Spring 05 than in previous years. However, comparing the scores on PEXes from Spring 2003 and Spring 2004 to those of Spring 2005 is problematic due to a complete instructor and course director change-over. The increase in scores between Spring 2003 and Spring 2004 offerings of computer architecture is more meaningful than the differences seen in the Spring 2005 because the course director and primary instructor was the same for each offering. The small change in scores does indicate that the idea of linked assignments using UAV communication and computer architecture as a theme can be transferred to another instructor.

TABLE IX
PEX COMPLEXITY (LINES OF CODE)

PEX	1	2	3	4
UAV (key)	637	767	878 / 720	2,337
UAV (shell)	223	0*	0* / 512**	1,963
UAV (goal)	Integer Encoder	IEEE 754 Encoder	IEEE 754 Multiplier	Cache (GUI)
Language(s)	MIPS	MIPS	MIPS / Ada	Ada

Previous (key)	320	444	464	550
Previous (shell)	51	107	25	405
Previous (goal)	Caesar Cipher	Hi-Lo Game	IEEE 754 Adder/Multiplier	Cache (text)
Language(s)	MIPS	MIPS	Ada	Ada

* PEX builds on previous PEX, hard-copy key available
** Ada template

Table X shows the end-of-course survey questions – the thirteen instructor-oriented questions are not shown. Figure 4 shows the student responses to these questions (converted from a 0 to 6 scale) across the last three offerings for the same instructor. For the first UAV PEX offering in Spring 2004, we noted an improvement in 10 / 10 questions (vs. Spring 2003) and 8 of 10 questions (vs. Fall 2002). Compared with Fall 2002 and Spring 2003, there is a significant ($\geq 10\%$) improvement for three – 6 (relevance), 7 (reasonableness), and 10 independent thought). Spring 2005 survey results were not available at the time of this writing.

TABLE X
COURSE SURVEY QUESTIONS

Question # (paraphrased for space)
1. Course Organization
2. Assessment techniques
3. Usefulness of course text(s)
4. Overall course
5. Clarity of objectives / requirements
6. Relevance / Usefulness
7. Reasonableness of assigned work
8. Course met stated objectives
9. Amount learned
10. Independent and intellectual thought

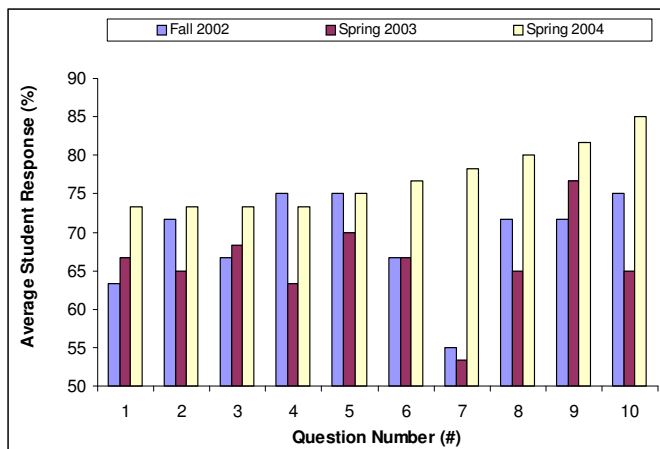


FIGURE 4
AVERAGE STUDENT RESPONSE ON END-OF-COURSE SURVEY

CONCLUSION

We have developed a real-world scenario involving UAVs in computer architecture. There are other real-world applications that also can be integrated, while keeping computer architecture relevant to both scientist and engineer and in the context of ACM 2001 curriculum guidelines [8].

Future work includes continuing to improve the exercise sequence. A key challenge is determining how robust the provided shell should be – balancing the amount of work required from the student versus the amount of work done for the student. Students report that while a shell is helpful to get started, too robust a shell is not, as the time required to comprehend and interface with it exceeds the amount of time required to write it themselves. A possible modifications include reversing the information flow – this changes the focus from telemetry to control, e.g., uploading flight plans.

Danial Neebel will be using linked assignments based on autonomous vehicle communications when teaching computer architecture at his home institution in Fall 2005.

ACKNOWLEDGMENT

We thank Al White and Rick Sward for their crucial work in developing USAFA’s UAV program – their efforts made integrating real-world problems in the classroom practical for many courses. We also thank Barry Fagin and Steve Hadfield for fruitful pedagogical discussions while developing this UAV exercise sequence and suggesting this paper be written. Finally, we gratefully thank our students for the last two years.

REFERENCES

- [1] 754-1985, *Standard for Binary Floating-Point Arithmetic*, IEEE, 1985.
- [2] Britton, R, *MIPS Assembly Language Programming*, Prentice-Hall 2004.
- [3] Carlisle, M, and Watkinson, B, *RAPID*, USAFA, ftp://ftp.usafa.af.mil/pub/dfcs/carlisle/usafa/rapid/index.html
- [4] Carlisle, M, Chamillard, T, and Michel, D, *AdaGIDE*, USAFA, http://www.usafa.af.mil/dfcs/bios/mcc_html/adagide.html, 2004.
- [5] Carlisle, M, et al, “RAPTOR: visual programming environment for teaching algorithmic problem solving”, *Proceedings of the 36th SIGCSE Symposium on Computer Science Education*, Feb. 2005.
- [6] Carlisle, M, et al, *AdaGraph*, USAFA, http://www.usafa.af.mil/dfcs/bios/mcc_html/ada_stuff.html, 2001.
- [7] Clements, A, “CSIDC: Competing students design real-world systems”, *IEEE Micro*, vol. 23, Jul 2003, pp. 78-80.
- [8] Computing Curricula 2001 Task Force, *ACM Computing Curricula Guidelines*, ACM, http://www.computer.org/education/cc2001/, 2001.
- [9] Dijk, J, V, *AdaGraph*, http://users.nrcvnet.nl/gmvdijk/adagraph.html.
- [10] Educational Outcomes, http://www.usafa.af.mil/df/outcomes.htm.
- [11] Fredell, R, *2003 USAFA Annual Research Report*, USAFA, http://www.usafa.af.mil/dfc/dfcr/report2003.pdf, 2003.
- [12] Haynes, D, *MIPster*, http://www.downcastsystems.com/mipster/, 2002.
- [13] Head L, et al, “Real-world design as a one-semester undergraduate project: example of a robust and low-cost solar lantern”, *IEEE Transactions on Education*, vol. 45, Nov. 2002, pp. 357-363.
- [14] Herwitz, S, *Directed Research: NASA UAV Project*, Clark University, http://www.clarku.edu/academiccatalog/course.cfm?id=1148.
- [15] Johnson, E, *UAV Research Facility*, Georgia Institute of Technology, http://controls.ae.gatech.edu/uavrfl.
- [16] Larus, J, *SPIM*, http://www.cs.wisc.edu/~larus/spim.html.
- [17] Leva, A, “A hands-on experimental laboratory for undergraduate courses in automatic control”, *IEEE Transactions on Education*, vol. 46, May 2003, pp. 263-272.
- [18] Long, L, *Unmanned Air Vehicles*, Penn State University, http://www.personal.psu.edu/faculty/l/n/lnl/uav1/, 2004.
- [19] Mandeville, W., *2004 USAFA Annual Research Report*, USAFA, http://www.usafa.af.mil/dfc/dfcr/report2004.pdf, 2004.
- [20] Mingus, T, and Grassl, R, “Algorithmic and Recursive Thinking”, 1998 Yearbook, edited by L. Morrow, NCTM, 1998, pp. 32-43.
- [21] Mode 3/AC Transponder, FAA, http://www.nas-architecture.faa.gov/cats/mechanism/mech_data.cfm?mid=483&RBGSY=2011
- [22] Neebel, D, Augeri C, MacMillan, G, Baird, L, and de Freitas A, “Work-in-Progress: A Visual Cache Memory Simulator”, *Frontiers in Education Conference*, Oct. 2005.
- [23] Nelson, V, Theys, M, and Clements, A, “Computer architecture and organization in the model computer engineering curriculum”, 33rd ASEE/IEEE Frontiers in Education Conference, Nov. 2003.
- [24] Patterson, D, and Hennessy, J, *Computer Organization and Design: The Hardware / Software Interface*, Morgan Kaufmann, 3rd ed., 2004.
- [25] Rotithor, H, “On the structure of a multipurpose introductory course in computer architecture at the Worcester Polytechnic Institute”, *IEEE Transactions on Education*, vol. 38, Aug. 1995, pp. 230-236.
- [26] Spinello, R, and Tavani, H, (eds.), *Readings in CyberEthics*, Jones and Bartlett, 2001.