`emuARM` : A Tool for Teaching the ARM Assembly Language

Geetika Malhotra
Indian Institute of Technology, Delhi

Namita Atri
Maharaja Surajmal Institute of Technology, Delhi

Smruti R. Sarangi
Indian Institute of Technology, Delhi

Abstract—**Technology has always enhanced learning as well as the overall teaching experience. With proper tools and resources in hand, we can easily integrate educational and information technologies into the academic environment. In this paper, we present a software tool to enhance the learning of microprocessors and computer architecture for students. We have developed an ARM instruction set emulator, `emuARM` , which is a Java based software tool for duplicating the functions of an ARMv5 microprocessor. Here, we present the internal design and features of `emuARM` . We present a comparison of the features of `emuARM` with other present ARM emulators in the market. At the end, we present the results of a survey that attests the pedagogical value of our tool.**

I. Introduction

Education has now crossed the four walled traditional classroom boundary. We have started living in a new world where we can attend a virtual university from the comfort of our homes. Technology has given a new form to the conventional education system [1]. Here, we are presenting a tool to help computer architecture students in understanding and learning assembly language in a more productive manner.

The advent of the mobile era has provided people with a wide range of easily accessible services on many subject areas. In this generation of ever increasing array of mobile devices, ARM is the industry's leading supplier of mobile device microprocessor technology, offering the widest range of microprocessor cores to address the performance, power and cost requirements for almost all application markets. This influx of ARM microprocessors requires that every computer science student takes a course on ARM assembly language [2], [3]. This is because most courses in computer science include a module in computer architecture and organization, and teaching students how to write assembly language forces them to understand the computer's architecture. Assembly language teaches how a computer works at the machine (i.e. register) level. It is therefore necessary to teach assembly language to all those who might later be working in computer architecture.

In this context, we discuss a tool, `emuARM` , which is extremely useful for learning ARM assembly language. We have designed and released it as an open source tool written in Java under the Apache license. It is an ARM machine emulator, that duplicates the functions of an ARM based microprocessor on other computer systems so that the emulated behavior closely resembles the behavior of the real system. `emuARM` illustrates a wide range of features, which are fundamental to the proficiency of this emulator. It has a simple and interactive GUI. The components are presented in such a manner that the user can focus his attention on all important ones at the same time. The assembly program can be written quickly, or can be imported from text files.

We have taken great care in designing the user interface, and have adhered to the industry practice of not having more than seven elements visible at the same point of time. In concordance with this rule, the frames containing the register set, the memory table, and the jump table have been allocated on different tabs. Only one tab is visible at any point of time. This will make it simple for teachers to explain the functioning of the registers, memory and the jump table in context with the execution of the program. The status window shows the errors in syntax and execution, and the teachers can throw some light on the behavior of program execution and reasons behind the errors incurred. Probably the most useful aspect of this emulator would be the debugger. Step Into, Step Out and Step Over debug modules can simplify the task of explaining the code and execution of every instruction and statement to the students. The breakpoint facility enhances debugging by providing more flexibility with limited step by step execution of the code.

This paper examines the design and implementation of `emuARM` and its features. It also entails a survey as a part of the project which tells us about the students' satisfaction with this tool. In Section V, we compare various ARM emulator tools based on a number of criteria that mostly pertain to the user's satisfaction quotient.

II. Design of `emuARM`

The design of `emuARM` has been divided into two basic modules, the front end and the back end. The front end provides the users with a simple interface to interact with the emulator and accept assembly language programs as input. This is sent to the backend which then executes the program and indicates the results in the register set, memory table and the status window. The block diagram is shown in Figure 1. The front end has three main components : file manager, editor and the status manager. The activities performed at the frontend are delegated to the backend so that appropriate actions can be taken. The back end has six main components : parser, error reporter, branch manager, debug module, register allocation and memory manager.

File manager provides a tree view of a system's file structure in a hierarchical format. The root of the tree points to the home directory of the user. It manages all the file operations. The editor enables the user to write a new program or view an existing program. Multiple tabs can be opened and closed. The editor also contains line numbers and other text-editing features. The status manager updates the register set, memory table, jump table and the status window. Switching between number formats like binary, decimal, octal and hexadecimal is also allowed. Other than that, the status manager displays all messages relevant to various emulator actions such as building

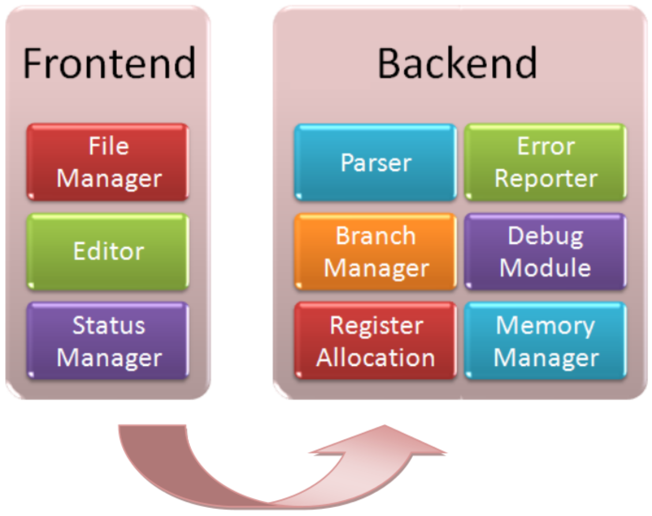


Fig. 1. Infrastructure of `emuARM`

a program, opening a file, and saving a file. All the errors listed by the error handling module in the backend are also displayed here. We have added separate macros to add print statements in the status window to display register values. These print statements can be useful in tracing the cause of the error during debugging.

Let us now consider the backend of `emuARM` . The parser takes an assembly language program as input and scans it line by line. It tokenizes the instructions and then builds the jump table. `emuARM` uses a hashmap to map the name of the instruction to an instance of the instruction executor class, whose job is to execute the instruction. The parser is integrated with the error reporter, which raises exceptions and shows the descriptive details of the error in the status window. The errors may arise due to an unknown command, an invalid addressing mode, or an invalid operand. It displays all the errors with the line number and displays the result as "Build failed".

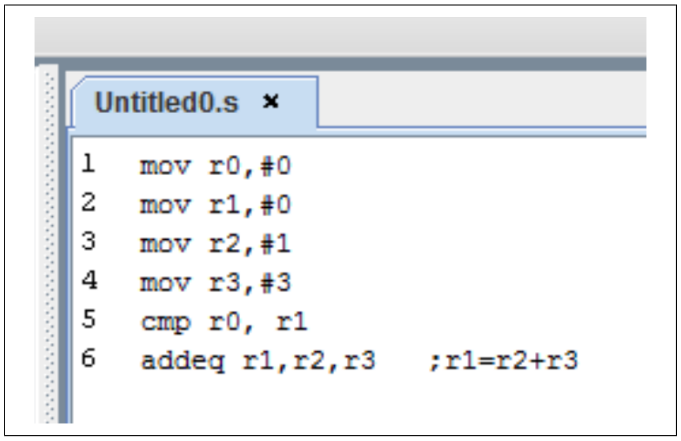
`emuARM` performs a two pass analysis of the ARM assembly code similar to most compilers. After the initial phase of lexical analysis, and parsing, the first pass builds the *jump table* from the parse tree. The jump table stores a list of all the labels in the assembly code. Each branch statement in `emuARM` uses a label as the branch target. The second pass replaces the labels with their actual addresses. The memory manager provides memory storage and retrieval capability in a clear and lucid manner. It also displays the list of memory locations used and their values in the memory table. Lastly, the debug Module helps in tracing the program execution line by line. `emuARM` supports the facilities of breakpoints, step-into, step-out and step-over.

III. Sample Session

In this section, we shall show the execution of a sample program along with the operations performed by `emuARM` in processing an instruction.

Consider the assembly language program of Figure 3.

This program may be executed completely or partially (debugging). The code is read line by line. A jump table is constructed in the first pass, and instructions are executed in the second pass after replacing the values of the labels with address from the jump table. Every scanned line can be one of the following:



```
1   mov r0,#0
2   mov r1,#0
3   mov r2,#1
4   mov r3,#3
5   cmp r0, r1
6   addeq r1,r2,r3    ;r1=r2+r3
```

Fig. 3. Sample code for execution

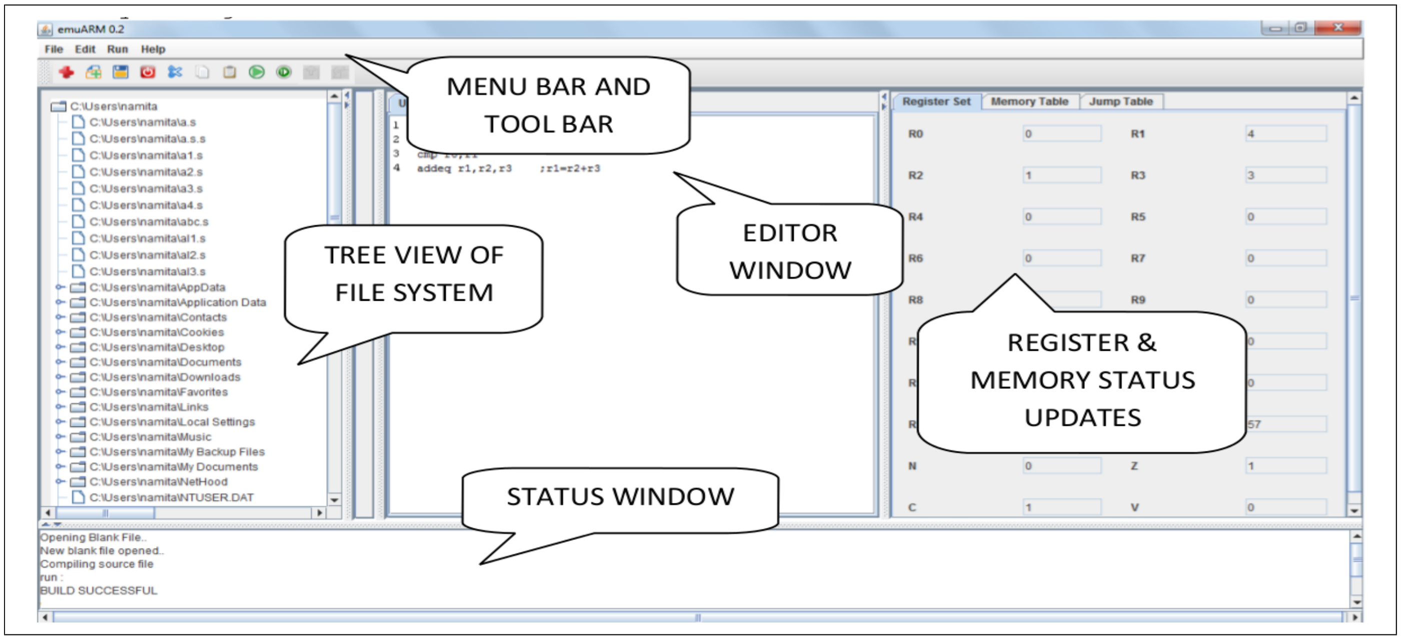
1) A blank line
2) A comment
3) An instruction
4) An instruction followed by a comment

Now, for any statement we have to ignore the first two cases and move ahead with the process of parsing the code. In our sample code, the first three statements fall in case 3 while the fourth statement falls in case 4. We recall that in assembly language, every comment starts with a ";" or "@". So, we tokenize the statement with ";" or "@" as the separator and obtain the instruction as the first token.

In the ARM instruction set, the second operand can have an optional shift argument that specifies the number of places that it needs to be shifted to the left or the right. There is a dedicated barrel shifter in the ARM processor for this purpose. Having a barrel shifter other than the ALU unit of the processor allows the programmers to perform shift operations in conjunction with other processing operations. For example, we can perform left shift as well as add in a single operation. First of all, we check whether any shift operation is to be performed on the operands or not. If there is a shift operation, then we execute the operation first.

We can conditionally execute almost all instructions irrespective of their operation such as addition, subtraction, branches, and loads. To enable conditional execution (also known as predicated execution), we need to add suffixes such as eq or ne after an instruction. For example, in line 4, we execute the $addeq$ instruction. This means that the add instruction will be performed only if the previous comparison has resulted in an equality.

After the execution of the shift operation if required, we use a hashtable (instruction table) to determine the instance of the class that needs to handle the instruction. Every instruction has a dedicated handler class that contains routines to emulate its execution on a real ARMv5 processor. The instruction along with the operands is sent to the appropriate handler class. Flags are updated if the instruction is either a compare instruction or the S bit is set in the instruction. After the execution of the instruction, the handler class updates the register file, and



```
C:\Users\namita
    C:\Users\namita\a.s
    C:\Users\namita\a.s.s
    C:\Users\namita\a1.s
    C:\Users\namita\a2.s
    C:\Users\namita\a3.s
    C:\Users\namita\a4.s
    C:\Users\namita\abc.s
    C:\Users\namita\al1.s
    C:\Users\namita\al2.s
    C:\Users\namita\al3.s
    C:\Users\namita\AppData
    C:\Users\namita\Application Data
    C:\Users\namita\Contacts
    C:\Users\namita\Cookies
    C:\Users\namita\Desktop
    C:\Users\namita\Documents
    C:\Users\namita\Downloads
    C:\Users\namita\Favorites
    C:\Users\namita\Links
    C:\Users\namita\Local Settings
    C:\Users\namita\Music
    C:\Users\namita\My Backup Files
    C:\Users\namita\My Documents
    C:\Users\namita\NetHood
    C:\Users\namita\NTUSER.DAT
```

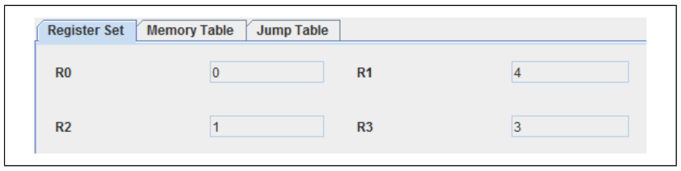
```
cmp r0,r1
addeq r1,r2,r3     ;r1=r2+r3
```

R0	0	R1	4
R2	1	R3	3
R4	0	R5	0
R6	0	R7	0
R8		R9	0
R			0
R			0
R			57
N	0	Z	1
C	1	V	0

```
Opening Blank File..
New blank file opened..
Compiling source file
run :
BUILD SUCCESSFUL
```

Fig. 2. Interface of emuARM

memory if required. The register set will reflect the values as shown in Figure 4 for our sample program.



R0	0	R1	4
R2	1	R3	3

Fig. 4. Register status after execution

The emulator window is shown in Figure 2. Along with the basic functionalities of the editor, toolbar, menubar and status window, it has the register status box, memory table, file tree view and jump table. Students were asked to comment on this layout, and the overall feedback was positive (see Section VI).

Till date emuARM implements all variants of ARM instructions in the standard unprivileged mode. There are plans to extend the simulator to model Thumb instructions, and privileged instructions.

IV. Customizing the Instruction Set and Extending emuARM

In most student projects, it is typically necessary for students to add new instructions to the instruction set. For example, students might be asked to extend the instruction set, or make changes to existing instructions. emuARM makes it fairly easy for students to make these changes. We use Java's built in object oriented features to nicely modularize the code. We have four generic instruction interfaces as shown in Figure 5. It is necessary for each instruction to implement any one of these interfaces.

Instructions can have different number of tokens in them. For example, a branch instruction has two tokens and an add instruction has four tokens. Here, a token corresponds to one operand, or the name of the instruction. Hence, we can select an appropriate interface depending on our need. For example, InsInterface2 is for a one address format instruction that generates two tokens. Note that the first token is the instruction name and the rest of the tokens are operands that follow the first token. ARM has instructions in 1, 2, 3, and 4 address formats. Hence, we have four separate interfaces (one for each format). If the user decides to add instructions that take more source operands, then she can add an appropriate interface on the lines of those shown in Figure 5.

```
public interface InsInterface2 {
     public void execute(String token1, String token2);
}

public interface InsInterface3 {
     public void execute(String token1, String token2, String
          token3);
}

public interface InsInterface4 {
    public void execute(String token1, String token2, String token3
       , String token4);
}

public interface InsInterface5 {
     public void execute(String token1, String token2, String
          token3,String token4,String token5);
}
```

Fig. 5. Interface implemented by instructions

Figure 6 shows the sample code for adding a new interface, which is easily constructed by adding a new *switch* case entry in the $execute(StringTokenizer\ st)$ method of the $Emit.java$ class. After creating an entry, we need to implement the execute function to get the desired functionality.

Lastly, we have to add an entry of the newly implemented instruction in the instruction table.

Parameter	emuARM	QEMU	SkyEye	armware
Ease of installation	✔	✔		
Easily usable	✔	✔	✔	
User manual is easy to understand	✔	✔	✔	
Graphical user interface exists	✔			
Covers all the standard ARM instructions	✔	✔	✔	✔
Error reporting facility	✔	✔	✔	✔
Breakpoints and other debugging facilities	✔		✔	

TABLE I: Comparison among tools

```
public class Emit {
  public static void execute(StringTokenizer st )
  {
     ...
     switch(ct)
     {
        //ADD YOUR NEW CASE HERE, N = number of tokens
  case N:
    if(ht.get(ins) instanceof InsInterfaceN)
                  {
                   obj=(InsInterfaceN) ht.get(ins);
                   token2=st.nextToken();
                   ...
                   tokenN=st.nextToken();
                   obj.execute(ins,token2,...,tokenN);
                  }
                  else
                  {
                       //SAME AS IN OTHER CASES
                  }
                  break;
     }
  }
}
```

Fig. 6. Adding a new interface

```
public class Emit {

  public void createHash()
  {
     ht.clear();
     ...
     //ADD THE FOLLOWING LINE
     ht.put("<NAME OF INSTRUCTION>",new <CLASS NAME>());
  }

}
```

Fig. 7. Adding an instruction's entry in the instruction table

V. Related work

A. Education Technologies

Technology driven education is becoming a major trend. Being an interactive, GUI based and easily installable tool, emuARM can be a part of the online learning package of a computer architecture course. De Bruno, Depover, and Dillenbourg[4] observe the importance of such tools in the field of education in general. They advocate the use of such assistive tools as important learning aids. [5] also makes the case for e-learning tools especially in distance education scenarios where interacting with the instructor is difficult. Note that emuARM is suitable for both assignments in conventional classroom based teaching, and also in a remote classroom based setting.

Dabbagh and Kitsantas [6] demonstrate how instructional designers and educators can provide opportunities for student self-regulation using such pedagogical tools. They note that such tools allow users to learn languages such as ARM assembly in the privacy of their homes. They can make as many mistakes as possible.

A lot of instruction set emulators such as spim (MIPS emulator) do not have expressive graphical interfaces. This makes using them difficult. As pointed out by Kasmarik [7], the learning experience is enhanced by 15-20% by using a graphical user interface.

B. ARM Emulators

There are many commercial and open source tools available for emulation of the ARM instruction set. In our opinion, none of them are tailored for educational purposes. We present a feature by feature comparison in Table I.

QEMU is a very popular dynamic binary translator (see [8]). Currently, it can emulate x86, PowerPC, ARM and Sparc processors. It is a batch mode tool, does not natively support debugging, and is meant for professionals. We have used QEMU primarily in our development process to verify the correctness of emuARM . We tried to use QEMU in a course with 135 students. Over 90% of them had reservations because it is not extremely friendly to novice users.

ARMulator [9], is a commercial tool provided by ARM to all users of ARM-based chips. It emulates the instruction sets of various ARM processors and their supporting architectures. It provides an environment for the development of ARM based software on a range of host systems. It supports the simulation of prototype ARM-based systems, ahead of the availability of real hardware, so that software and hardware development can proceed in parallel. This tool is not sold separately and hence is not suitable for a broader audience comprising mostly of students.

ARMware [10] converts ARM code snippets into blocks of x86 machine code snippets. It is an advanced emulator that offers support for dynamic compilation, and multi threading. However, it lacks a well written user manual, graphical user interface, and modules for adding extra instructions.

SkyEye [11] is another ARM based full system simulator. Our experience with SkyEye has been extremely positive from the point of view of the quality of tool. However, the reason that students do not prefer it is because it lacks a graphical user interface, and takes a long time to setup and use.

Apart from all these tools, the ARM processor family is very well supported by the GNU C/C++ toolchain. It provides tools for creating an ARM assembly file from C/C++ code and executing the assembly code, along with rich sources of documentation.

Table I lists a comparison in the features of emuARM with other tools. As we can see from this table, a list of seven fundamental characteristics have been used to compare the features of other ARM emulators. We observe that our tool scores well in all the counts, and these results are corroborated by the results of our survey (see Section VI).

VI. Survey

We recorded the number of downloads using a Google analytics framework for users from all over the world. After getting an overwhelming response, we decided to design a targeted survey to be able to get some useful feedback. From a focus group, we primarily wanted to evaluate the effectiveness of `emuARM` and its contribution to their learning. The survey was conducted through an online questionnaire system. Students who took the CSL211(Introduction to Computer Architecture) course at the Indian Institute of Technology, Delhi, were asked to voluntarily participate in this investigation, and their anonymity was ensured. A total of 30 students successfully completed the questionnaire. They were all undergraduate students in their second or third year of their engineering course. The data was collected from June 15, 2013, to July 15, 2013. The survey included ten questions for each participant with three options each, as can be seen in the Table II. These questions focus on fundamental features of `emuARM` and its usability. The participants enthusiastically agreed to participate in the survey.

Table II summarizes the information obtained from this survey. The second column gives the response statistics in the form of percentages of the number of users opting for an option with respect to the total number of users. These values have been calculated based on responses collected. Most of these students answered the questions in favor of `emuARM` on a different range of criteria.

Three quarters of the participants observed that installing `emuARM` was fairly easy. However, the rest of the participants had an issue with the version of Java. `emuARM` requires at least Java 6. Some students found it difficult to upgrade their versions of Java to 6. The next question was regarding the relevance of the tool. We asked the students if they found assembly programming difficult, and over 90% of the students answered in the affirmative. Next, we looked at the debugging feature. Roughly 86% of the participants found the debugging interface to be acceptable. A majority of them really appreciated the interface. Around 13% were not concerned about the debugging features provided by `emuARM` .

Henceforth, we asked the students about the nature of the user manual and its relevance to their sessions with `emuARM` . Ironically, 93% of the participants did not feel the need to refer to the manual. They were of the opinion that the user interface is self explanatory. From an educational point of view, this is a rather positive observation, because we would not ideally like students to be referring to the user manual frequently. Out of the rest of the respondents, 50% liked the manual, and the rest of them did not refer to the manual.

We then asked the students about issues, and bugs in the tool. In specific, we were interested to know if there were any functional errors. Three quarters of the students did not find any issues. Interestingly, 20% of the students reported incorrect executions. We investigated each of these incidents. In some cases, the students had written a wrong program; however, there were some bugs in `emuARM` , which were subsequently rectified. A concomitant question was regarding the error location facility of `emuARM` . 73% of the participants found it to be useful. 10% of the participants were not able to identify the location of the error even after using the facility, and 16.67% reported no errors in their programs.

The last three questions, were broad overview questions. We asked them about the user interface and its overall attractiveness. Two third of the participants found the user interface to be nice and user friendly. Just for the UI, 26.67% reported referring to the manual. 6% did report some problems with the user interface. We are in the process of addressing their versions, and their issues will be redressed in the next release of `emuARM` . In question 9, we asked the students if they found `emuARM` to be a helpful tool in trying to learn more about the ARM instruction set and microprocessor. Two third of the class really liked the tool, and the rest of the participants were equally divided between a negative and a neutral viewpoint. The last question (9) was the most important question. We asked the respondents if they would like to use `emuARM` as a learning aid and possibly also recommend it to others for learning and teaching purposes. 50% recommended `emuARM` and had rave comments. 30% believed that other tools such as GNU tools which they were already using, could be a possible alternative to `emuARM` . Their main concern was that they did not wish to migrate to a new tool. Issues in `emuARM` were perceived by us to be a less of a concern. 20% of the participants had not used any other tool; hence, they did not have an opinion.

To summarize, `emuARM` v1.0 has been recommended for use by more than two-third of the participants in our survey. The readers need to note that these respondents are students of a very prestigious engineering university in India, and they are well aware of a wide range of software tools. Some of the features of `emuARM` were appreciated by more than 90% of the participants, and almost all of them found it be a nice and novel effort in this space.

VII. Conclusion

In this paper, we introduced a new tool, `emuARM` , and presented its design and working. This tool enables effective and efficient teaching of ARM assembly. Through the survey, we proved students' satisfaction towards the tool.

References

[1] H. A. Latchman, C. Salzmann, D. Gillet, and H. Bouzekri, "Information technology enhanced learning in distance and conventional education," *IEEE Transactions on Education*, vol. 42, no. 4, pp. 247–254, 1999.

[2] *ARM Architecture Reference Manual* , ARM Limited, 2000.

[3] J. R. Gibson, *ARM Assembly Language an Introduction.* Lulu, 2011.

[4] B. D. Lievre, C. Depover, and P. Dillenbourg, "The relationship between tutoring mode and learners use of help tools in distance education," *Instructional Science*, vol. 34, no. 2, pp. 97–129, 2006.

[5] C. C. Ardito, M. D. Marsico, R. Lanzilotti, S. Levialdi, T. T. Roselli, V. Rossano, and M. Tersigni, "Usability of e-learning tools," in *Proceedings of the working conference on Advanced visual interfaces*, ser. AVI'04, 2004, pp. 80–84.

[6] N. Dabbagh and A. Kitsantas, "Supporting self-regulation in student-centered web-based learning environments," *International Journal on E-Learning*, vol. 3, no. 1, pp. 40–47, 2004.

[7] K. Kasmarik and J. Thurbon, "Experimental evaluation of a program visualisation tool for use in computer science education," in *Proceedings of the Asia-Pacific symposium on Information visualisation*, ser. APVis '03, vol. 24, 2003, pp. 111–116.

[8] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC'05, 2005, p. 41.

[9] The armulator. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0032f/index.html

[10] armware, this is an arm emulator. [Online]. Available: http://code.google.com/p/armware/

S. No.	QUESTION	%
1.	**Was there any problem installing `emuARM` ?**	
a)	No, not at all	76.67%
b)	Can be better	6.67%
c)	I did not do on my own	16.67%
2.	**Do you find coding in assembly difficult?**	
a)	Yes, of course.	90%
b)	No. Its fun.	3.33%
c)	Neutral.	6.67%
3.	**Did the debugging tool help in finding the bugs ?**	
a)	Yes. The breakpoint feature is nice.	53.33%
b)	It is okay.	33.33%
c)	I did not care.	13.33%
4.	**Was the manual helpful in understanding the user interface and working ?**	
a)	The GUI is very understandable. There was no need to refer to the manual.	93.33%
b)	Yes, Manual helped.	3.33%
c)	I did not refer to the manual. I asked my friend.	3.33%
5.	**Does `emuARM` give correct results ?**	**%**
a)	Yes	76.67%
b)	No.	20%
c)	Do not know. Never used.	3.33%
6.	**Does the error reporting facility help in guiding with pointing out the location and nature of error ?**	
a)	Yes, it helped every time.	73.33%
b)	No, I had no clue about the nature/location of errors.	10%
c)	I never get an error.	16.67%
7.	**Is the `emuARM` interface simple to work with and easy to understand ?**	
a)	Yes, the interface is easy and attractive.	66.67%
b)	I had to refer to the manual to understand it.	26.67%
c)	It is very difficult to understand the interface.	6.67%
8.	**Do you think its a helpful tool for learning ARM microprocessor ?**	
a)	Yes, I loved it	66.67%
b)	No, It is not very good.	16.67%
c)	It is okay.	16.67%
9.	**Do you prefer `emuARM` as a tool for learning over other tools ?**	
a)	Yes, I do	50%
b)	No, I do not	30%
c)	I haven't used any other tool.	20%

TABLE II: SURVEY QUESTIONS

[11] Skyeye, a very fast full system simulator. [Online]. Available: http://sourceforge.net/apps/trac/skyeye/wiki/UM1