Teaching Computer Concepts Using Virtual Machines

Ola Ågren
Department of Computing Science
Umeå University
SE-901 87 Umeå, SWEDEN
ola@cs.umu.se

Abstract
A set of virtual assemblers and a virtual machine are used as a teaching tool in order to teach students differences and similarities between architectural styles of computer processors. Programs written by the students in the virtual assemblers are compiled using the corresponding virtual assembler and then executed on the virtual machine so that the students can follow the execution of the programs step by step or at full speed.

1. Introduction
The course in Computer Architecture is one of the most popular courses in Computing Science at Umeå University. It is a graduate level course, mandatory for students of the MSC/E program and optional for students of the MSC and BSC/E programs. The main text of the course is the very popular textbook by Patterson and Hennessy [1].

1.1 Problem
The course has grown over the years and last fall had an enrollment of over 100 students, which meant that the assignments had to be changed because of resource limitations. The first assignment has been to build an instruction tracer that executed other programs while collecting vital statistics about it (i.e., size of data used in the operations, register use, length of basic blocks and calls, bits used in instructions, etc.). This assignment unfortunately required that all students had access to machines with MIPS processors. This assignment had to be scrapped since there are only a total of 28 SGI workstations and two SGI servers available and test runs of the assignment led to an unacceptably high load level on the machines running it.

Since the author wanted to highlight the issue of different architectural styles a new assignment was planned. A virtual machine [2] running on all UNIX platforms available (SGI Irix, SUN Solaris and IBM AIX) had to be created together with virtual assemblers that converted assembler code of the different architectural styles to the virtual target machine assembler.

1.2 Goals
There were three goals for the new assignment:
1. It had to give deeper understanding of how different computer architectures behaves,
2. It had to be easy to understand and use, and
3. It had to be portable to all platforms available at the department (except Windows-based PCs).

All of the goals were deemed imperative for the actual implementation to be successful.

2. Solution
New hardware is expensive both to acquire and maintain, especially if it makes the machine park heterogeneous. Moreover, very few machines with unorthodox system architectures are currently in production, accumulator and stack machines have become very rare lately. This indicated that another route had to be followed: The Virtual Machine.

The basis of virtual machinery is the actual hardware that is supposed to build up a machine. This is constructed in such a way that it "executes" the program given in the same way as an actual machine would do, but in software. Virtual machines have been used for a multitude of reasons. One use of virtual machines has been to test hardware systems before actual construction (simulation, more recently done in VHDL or Verilog). Another use is to give system developers the availability of their own machine without blocking it for all others (providing the user with the same interface as the underlying hardware) [2].
3. The Different Types of Virtual Machines
There are four basic types of architectures ([1]): Accumulator, Stack, Memory-Memory and Load-Store. Instead of
developing a virtual machine (with user interface) for each style we opted for a system of programs. This system is
built up of one general-purpose register architecture in a virtual machine (the only one with interactive user interface)
and four independent virtual assemblers that convert their input to the assembler of the actual virtual machine.

The virtual assemblers were created with small instruction sets (between 20 and 30 for all machines) to make
the learning as fast as possible. The different assembler languages had some parts that were similar (e.g., name of the
normal operations) and some that were not (i.e., branching/jumping, how to create variables, and architecture specific
instructions).

3.1 Accumulator
The accumulator machine is one of the most basic processor architectures created. This type of machine has one
register (called the accumulator) that is implicitly used as one of the operands in all instruction and is the destination
of all calculations. It is the target of all loads and the source for all stores.

3.2 Stack
A strict stack machine does not have any general purpose registers at all. All data is handled in a last-in, first-out
stack. Operations take their operands from the stack and the result is pushed back onto the stack, except for
operations that move data from memory to stack or vice versa.

3.3 Memory-Memory
A strict memory-memory machine does not have any general purpose registers. The main difference from stack
machines is that operations in a memory-memory machine use the values in memory cells as operands and the result
is stored in a memory cell.

3.4 Load-Store
The load-store machine has a fixed number of registers (in our case 32) that are used as operands and the destination
of the result for operations, except for operations that transfer data between registers and memory or vice versa.

4. The System of Scripts and Machines
All parts of the system had to be portable and preferably using only command line or terminal-based interfaces in
order to maximize usability of the system. There had to be one (and only one) basic starting point for the system and
a consistent interactive user interface of the virtual machines.

To accommodate these needs a small system of scripts was devised. The first one checked the suffix of the
given input file to run the correct virtual assembler for the given assembler program file (ac for accumulator, ls for
load-store, mm for memory-memory and st for stack). The output from the virtual assembler was then piped into the
virtual machine that changed its behaviour slightly depending on the command line arguments set by the main script.
The main script handles the difference in underlying architectures by having sets of the programs in directories with
the same name as the standard UNIX arch command gives.

The virtual machine does not only execute the code but gives a terminal based debugging environment where
the current line (as given in the input file) is highlighted and all variables are shown together with their values.
Moreover, the main script tells the virtual machine what other things to show depending on the suffix of the given
source file (the accumulator, the stack, the lowest addresses or the registers, respectively). Some of the commands
available in dbx/xdb/sdb are functional, i.e., s for step, n for next, g for go. When either a stop operation is executed
or an uninitialized address is reached the execution stops and the total number of instructions executed (both
the symbolic ones in the input file and those in the general purpose virtual machine) is presented to the user.

All scripts are Bourne shell scripts and the virtual machines are written in ANSI C (using curses to
implement the interface), with some parts in yacc (memory-memory assembler only) and lex.
5. Sample Screenshot
The following screenshot shows the virtual machine/debugger in operation. The program being tested is a stack machine representation of the C statement \( A=B+C*D; \) and shows the screen when all the values have been pushed. The declarations of variables are off screen.

```
6
7 ; C statement "A=B+C*D;" on a stack machine
8 start push B
9 push C
10 push D
> 11 mul
12 add
13 pop A
14 stop
15 end start
```

Variables:
- \( D: 3 \)
- \( C: 2 \)
- \( B: 1 \)
- \( A: 0 \)

Stack:
- 3
- 2
- 1

s)tep or g)o?

6. Conclusions
The students have found it much easier to understand how different architectural styles affect code (generality, parameter passing, layout, execution, porting, etc.) after this assignment was added to the curriculum. Since this was one of the main goals for the assignment it has to be regarded as a success so far.

The system is easy to use for anyone that has experience with terminal based software, thereby fulfilling the second goal. Some of the students that were used to graphical user interfaces felt that the system was out of date because of the lack of such.

The third major goal of this assignment was that any workstation or server could be used as platform for programming and debugging of the students' assignments, thereby reducing the burden of the SGI machines. This goal was clearly achieved. There were actually students working with their assignments using the terminal program in their home PC.

7. Future Updates and Additions
We are currently developing an index machine (an accumulator machine with up to two levels of indirection using two index registers) that is supposed to be added for the next course (fall 1999).

There have been ideas about translating the virtual machine/debugger to Java (a programming language designed by Sun Microsystems) in order to preserve the portability while giving an opportunity to add a graphical user interface in an easy and structured manner. This would mean that the virtual machine would be running in another virtual machine...

References