General Processor Emulator [GenProEmu]

Bindu B, Shravani K L, Sinchana Hegde
Guide: Mr.Aditya Koundinya B,
Asst. Prof
Computer Science Department
Jyothy Institute Of Technology
Tatguni,Bangalore-82

Abstract **- GenProEmu is an interactive computer emulation package in which the user specifies the details of the CPU to be emulated, including the register set, memory, the microinstruction set, machine instruction set, and assembly language instructions. Users can write machine or assembly language programs and run them on the CPU they've created.**

GenProEmu simulates computer architectures at the register-transfer level. That is, the basic hardware units from which a hypothetical CPU is constructed consist of registers and memory (RAM). The user does not need to deal with individual transistors or gates on the digital logic level of a machine.

1. INTRODUCTION

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical
parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded
systems control many devices in common use today.

Embedded systems are dedicated to handle a particular task. Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. One of the very first recognizably modern embedded systems was the Apollo Guidance Computer.

1.1 Processor

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

A microprocessor is a computer processor that incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits.The microprocessor is a multipurpose, programmable device that accepts digital data as input, processes it according to instructions stored in its memory, and provides results as output. It is an example of sequential digital logic, as it has internal memory. Microprocessors operate on numbers and symbols represented in the binary numeral system.

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions called a program. The instructions are kept in some kind of computer memory. There are three steps that nearly all CPUs use in their operation: fetch, decode, and execute.

Fetch-
The first step, fetch, involves retrieving an instruction (which is represented by a number or sequence of numbers) from program memory. The instruction's location (address) in program memory is determined by a program counter (PC), which stores a number that identifies the address of the next instruction to be fetched.

Decode-
The instruction that the CPU fetches from memory determines what the CPU has to do. In the decode step, the instruction is broken up into parts that have significance to other portions of the CPU.

Execute-
After the fetch and decode steps, the execute step is performed. Depending on the CPU architecture, this may consist of a single action or a sequence of actions.

1.2 Emulator

Emulator is hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest). An emulator typically enables the host system to run software or use peripheral devices designed for the guest system.Emulation refers to the ability of a computer program in an electronic device to emulate (imitate) another program or device.

Developers of software for embedded systems or video

game consoles often design their software on especially accurate emulators called simulators before trying it on the real hardware. This is so that software can be produced and tested before the final hardware exists in large quantities, so that it can be tested without taking the time to copy the program to be debugged at a low level and without introducing the side effects of a debugger. In many cases, the simulator is actually produced by the company providing the hardware, which theoretically increases its accuracy.

CPU simulator-
The CPU simulator is often the most complicated part of an emulator. Many emulators are written using "pre-packaged" CPU simulators, in order to concentrate on good and efficient emulation of a specific machine.

The simplest form of a CPU simulator is an interpreter, which is a computer program that follows the execution flow of the emulated program code and, for every machine code instruction encountered, executes operations on the host processor that are semantically equivalent to the original instructions.

Functional simulators-
Functional simulation is the use of a computer program to simulate the execution of a second computer program written in symbolic assembly language or compiler language, rather than in binary machine code. By using a functional simulator, programmers can execute and trace selected sections of source code to search for programming errors (bugs), without generating binary code. This is distinct from simulating execution of binary code, which is software emulation.

1.3 Need for GenProEmu

Embedded processors can be broken into two broad categories. Ordinary microprocessors (µP) use separate integrated circuits for memory and peripherals. Microcontrollers (µC) have on-chip peripherals, thus reducing power consumption, size and cost. In contrast to the personal computer market, many different basic CPU architectures are used, since software is custom-developed for an application and is not a commodity product installed by the end user. Embedded debugging may be performed at different levels, depending on the facilities available.
Debugging can be done using emulators because building a microprocessor and then testing is an expensive process and weste of resources.

An in-circuit emulator (ICE) replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor. A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified, and allowing debugging on a normal PC machine or assembly language programs and run them on the CPU they've created.

GenProEmu simulates computer architectures at the register-transfer level. That is, the basic hardware units from which a hypothetical CPU is constructed consist of registers and memory (RAM). The user does not need to deal with individual transistors or gates on the digital logic level of a machine.

2. LITERATURE SURVEY

2.1 Eclipse

Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications.

The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.Eclipse uses plug-ins to provide all the functionality within and on top of the runtime system.

The Eclipse SDK includes the Eclipse Java development tools (JDT), offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards.

Eclipse implements uses the graphical control elements of the Java toolkit called SWT, whereas most Java applications use the Java standard Abstract Window Toolkit (AWT) or Swing.

2.1.1 WindowBuilder

WindowBuilder is a powerful and easy to use bi-directional Java GUI designer.
WindowBuilder is composed of SWT Designer and Swing Designer and makes it very easy to create Java GUI applications without spending a lot of time writing code. Use the WYSIWYG visual designer and layout tools to create simple forms to complex windows; the Java code will be generated for you. Easily add controls using drag-and-drop, add event handlers to your controls, change various properties of controls using a property editor, internationalize your app and much more.

WindowBuilder is built as a plug-in to Eclipse and the various Eclipse-based IDEs (RAD, RSA, MyEclipse, JBuilder, etc.). The plug-in builds an abstract syntax tree (AST) to navigate the source code and uses GEF to display and manage the visual presentation.

Generated code doesn't require any additional custom libraries to compile and run: all of the generated code can be used without having WindowBuilder Pro installed. WindowBuilder Pro can read and write almost any format and reverse-engineer most hand-written Java GUI code. It also supports free-form code editing (make changes anywhere...not just in special areas) and most user re-factorings (you can move, rename and subdivide methods without a problem).

2.2 Design Pattern

Design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply object-orientation or, more generally, mutable state, are not as applicable in functional programming languages.
Patterns originated as an architectural concept by Christopher Alexander (1977/79). In 1987, Kent Beck and Ward Cunningham began experimenting with the idea of applying patterns to programming and presented their results at the OOPSLA conference that year. In the following years, Beck, Cunningham and others followed up on this work.

2.2.1 History

Design patterns gained popularity in computer science after the book Design Patterns: Elements of Reusable Object-Oriented Software was published in 1994 by the so-called "Gang of Four" (Gamma et al.), which is frequently abbreviated as "GoF". That same year, the first Pattern Languages of Programming Conference was held and the following year, the Portland Pattern Repository was set up for documentation of design patterns.

Practice-
Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns.

In order to achieve flexibility, design patterns usually introduce additional levels of indirection, which in some cases may complicate the resulting designs and hurt application performance.

By definition, a pattern must be programmed anew into each application that uses it. Since some authors see this as a step backward from software reuse as provided by components, researchers have worked to turn patterns into components. Meyer and Arnout were able to provide full or partial componentization of two-thirds of the patterns they attempted.

Software design techniques are difficult to apply to a broader range of problems.[citation needed] Design pattern provide general solutions, documented in a format that does not require specifics tied to a particular problem.
Abstract Factory is a creational design pattern.

2.2.2 Abstract Factory

Modularization is a big issue in today's programming. Programmers all over the world are trying to avoid the idea of adding code to existing classes in order to make them support encapsulating more general information. Take the case of a information manager which manages phone number. Phone numbers have a particular rule on which they get generated depending on areas and countries. If at some point the application should be changed in order to support adding numbers form a new country, the code of the application would have to be changed and it would become more and more complicated.

In order to prevent it, the Abstract Factory design pattern is used. Using this pattern a framework is defined, which produces objects that follow a general pattern and at runtime this factory is paired with any concrete factory to produce objects that follow the pattern of a certain country. In other words, the Abstract Factory is a super-factory which creates other factories (Factory of factories).

Abstract Factory offers the interface for creating a family of related objects, without explicitly specifying their classes.
The classes that participate to the Abstract Factory pattern are:

AbstractFactory - declares a interface for operations that create abstract products.

ConcreteFactory - implements operations to create concrete products.

AbstractProduct - declares an interface for a type of product objects.

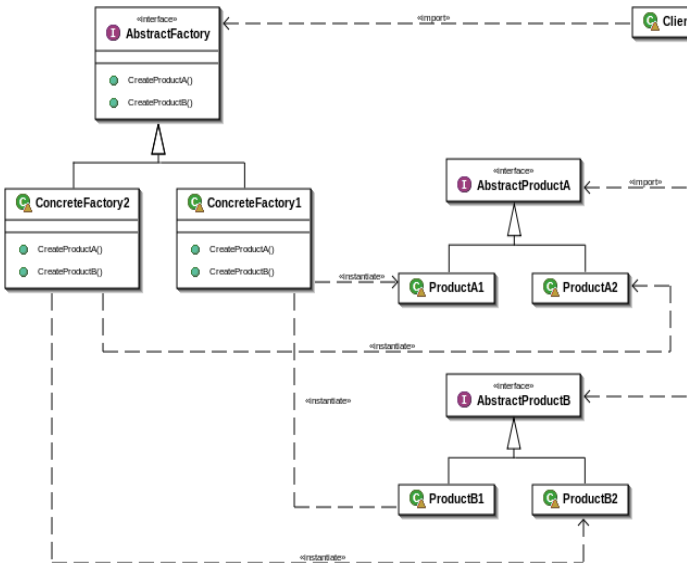
Product - defines a product to be created by the corresponding ConcreteFactory; it implements the AbstractProduct interface.

Client - uses the interfaces declared by the AbstractFactory and AbstractProduct classes.

The AbstractFactory class is the one that determines the actual type of the concrete object and creates it, but it

returns an abstract pointer to the concrete object just created. This determines the behavior of the client that asks the factory to create an object of a certain abstract type and to return the abstract pointer to it, keeping the client from knowing anything about the actual creation of the object.

UML Class Diagram



3. CONCLUSION

Testing and debugging Embedded system hardware and software is a costly affair. Using emulators or simulators will reduce this cost considerably and also increase the debugging capability .

GenProEmu is an emulator of the same kind which can be used to design and test different kinds of processors by running assembly language programs on the user designed machine. Hence GenProEmu aims to makes designing and testing of processors easier.

4. REFERENCES

[1]http://howtodoinjava.com/2012/10/29/abstract-factory-pattern-in-java/ http://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm

[2]Java Foundation Classes in a Nutshell by David Flanagan, copyright (c) 1999 by O'Reilly & Associates.

[3] "Programming XML in Java" by Mark Johnson, JavaWorld, April, 2000.

[4]"The Java Tutorial" at http://java.sun.com/docs/books/tutorial.