ARM7 Processor Simulator with Graphical User Interface

Khaw Boon Chai*, Muhammad Mun'im bin Ahmad Zabidi **
Department of Microelectronics and Computer Engineering
Universiti Teknologi Malaysia
81310 UTM Skudai
Johor, Malaysia
*boonchai83@gmail.com, munim@utm.my***

Abstract

The paper describes an ARM7 processor simulator with graphicak user interface (GUI) which is easy and suitable for the beginner. This software is written in Java language and the graphic user interface is written using Eclipse Standard Widget Toolkit (SWT). GNU toolchain is using as the compiler and assembler for this simulator to produce binary machine code for ARM. The simulator decodes the 32-bit ARM machine instructions and simulate the execution of every instruction.

1. Introduction

ARM is a 32-bit processor architecture that is widely used in a number of embedded designs. Because of its power saving features, this architecture is dominant in the mobile electronics market, where low power consumption is a critical design goal [1].

ARM processors come in several variations. The simplest version is the ARM7 which implement Version 4 of the ARM instruction set. ARM7 is also the version with the lowest power consumption and widely used in cost-sensitive applications.

To create software for the ARM, developers use commercially available toolchain or the open-source GNU toolchain. To run ARM software before hardware is available, simulators such as GDB (GNU Debugger), SimIt-ARM, and Green Hills Multi for ARM. However, these simulators were designed for professionals and too complex for educational use.

The objective project is to develop a simulator to simulate the ARM7 processor which is suitable for beginner, so it can ease the learning process for them.

2. ARM Architecture

ARM is a typical Reduced Instruction Set Computer (RISC) with the following extra features:

- usage of both the Arithmetic Logic Unit and shifter in every data-processing instruction
- auto-increment and auto-decrement addressing modes
- load and store multiple instructions
- conditional execution of all instructions

ARM has 31 general-purpose 32-bit registers; at any one time only 16 of these registers are visible. The other registers are used to speed up exception processing. Two of the 16 visible registers have special roles which are Register 14 (Link Register or LR) and Register 15 (Program Counter or PC). The remaining 14 registers have no special hardware purpose. Their uses are defined purely by software. Software normally uses Register 13 as a Stack Pointer (SP). Current operating processor status is stored in Current Program Status Register (CPSR). Each exception mode also has a Saved Program Status Register (SPSR) which holds the CPSR immediately before the exception occurred. CPSR and SPSR are accessed with special instructions. Information that is held in the status register are 4 bits of condition code flags (Negative, Zero, Carry and Overflow), 2 interrupt disable bits, 5 bits current processor mode and 1 bit of ARM or Thumb instruction set [2].

3. Simulator Design

This simulator is designed as multi-tabbed software; the user may open many files with simulator program running. It is also designed as multi-threaded program, so it can run many simulations simultaneously.

There are 3 main Java classes developed for this simulator. They are the GUI class, tabbedComponent class, and ARM7Core class.

GUI class is the main class for this simulator. Its function is to create and display the main graphic user interface for the simulator.

The tabbedComponent class creates the tabs when the user opens a file. Depending on the type of file opened it creates a simulation tab or editor tab. If a user open .bin (binary) file, a “simulation tab” will be created. If a user opens a .c (C language) file, a .s (ARM assembly language) file or other filetypes, an “editor tab” will be created. Refer to Figures 1 and 2 for screenshots.

Compilation functions are only available to the editor tab. When the user click on a button to start compiler, the software activates the GNU toolchain located in the system path. Simulation of ARM7 instruction execution functions is only available to the simulation tab.

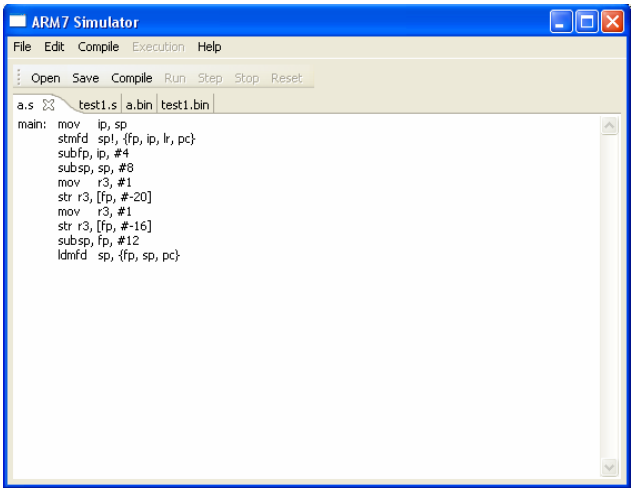

Figure 1: Editor Tab

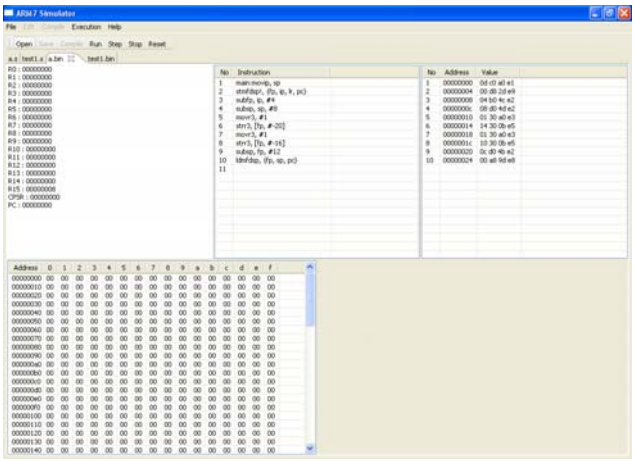

Figure 2: Simulator Tab

4. Instruction Simulation

The simulation process is handled by the ARM7Core class. Its functions are to simulate the execuiton of ARM7 machine instructions and store the machine state. Memory is declared as an 8-bit wide array, while registers are declared as a 64-bit wide array. Each instruction is simulated by a separate function.

ARM7Core instruction executing processes are:

- fetch 32-bits instruction from memory according to the address in Register 15 (Program Counter)
- decode fetched instruction and determine which function to call
- increment the Program Counter by 4

Every simulator tab will instantiate an ARM7Core object and the GUI is updated after executing each instruction.

Each instruction of ARM is a 32-bit binary code. Bits 28-31 is always the condition field. This field determines if the instruction is to be executed depending on the value of N, Z, C and V flags in the CPSR. If the flags satisfy the condition specified in this field, the instruction will be executed. Otherwise, the instruction acts as a NOP and execution advances to the next instruction.

The remaining bits represent the operation code and operand addressing modes. The instruction formats are shown in Figure 3. To efficiently decode these bits, the decoding strategy shown in Figures 4 and 5 is used. Clearly, most instructions have the bits 27-26 equal to 00 so the decoding tree checks these two bits first.

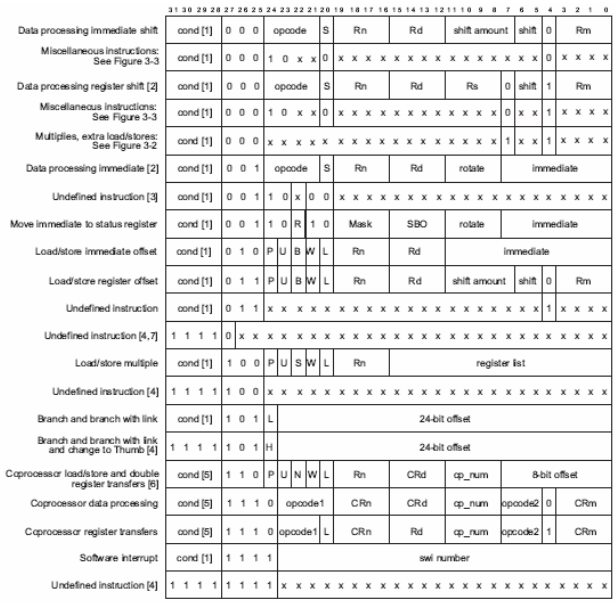

Figure 3. ARM Instruction Formats

5. Results and Future Works

As tested, the simulator enables the beginner is edit ARM assembly language source files and simulates the machine instructions. Further work that can be done to include syntax-coloring editor and simulation of simple input/output devices. The simulator can also be extended into an emulator by connecting to an physical ARM board through serial or JTAG ports, so that ARM chips can be programmed directly and the software debugged from the same software.

References

[1] Stever Furber, *ARM System-on-Chip Architecture, 2e*, Addison-Wesley, 2000.

[2] ARM Limited, *ARM Architecture Reference Manual (ARM DDI 0100E)*, 2000.

[3] Standard Widget Toolkit. Available at http://www.eclipse.org/swt/.

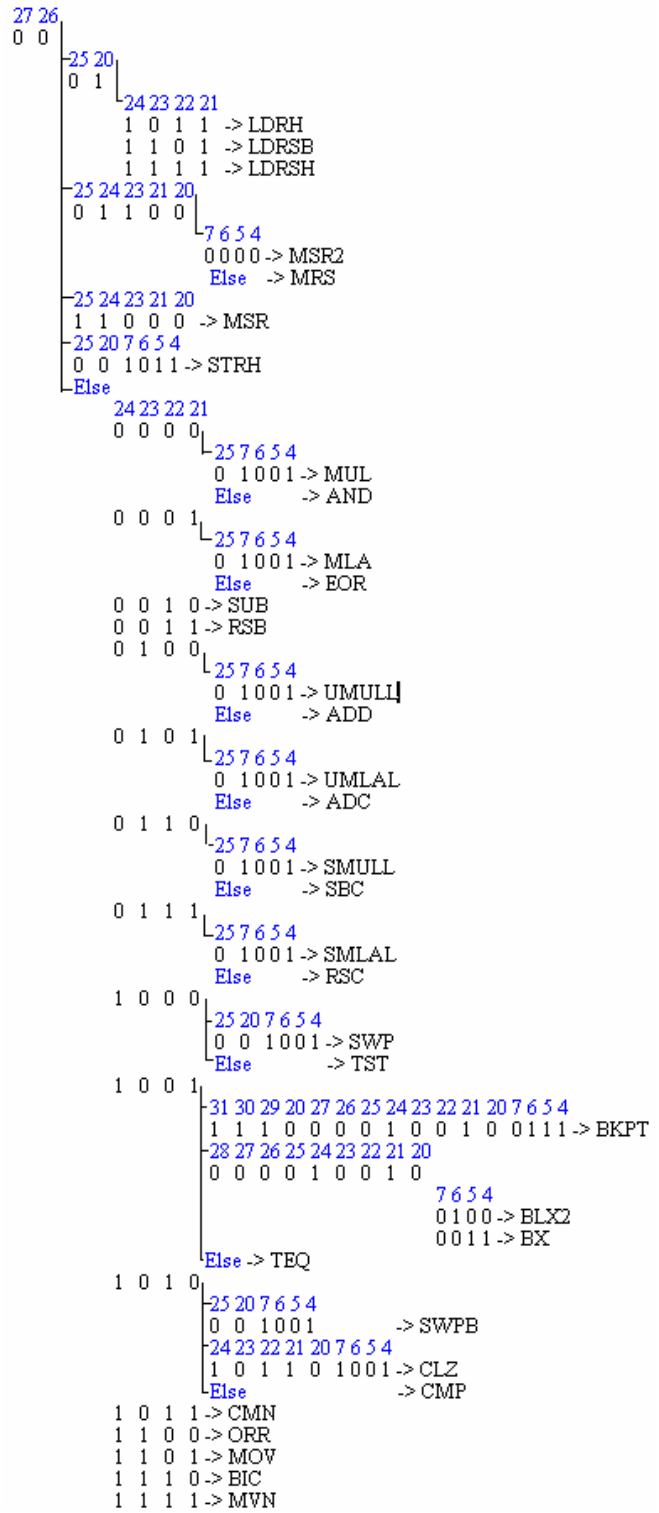

Figure 4: Machine Code Decode Table for Bits 27-26 = 00

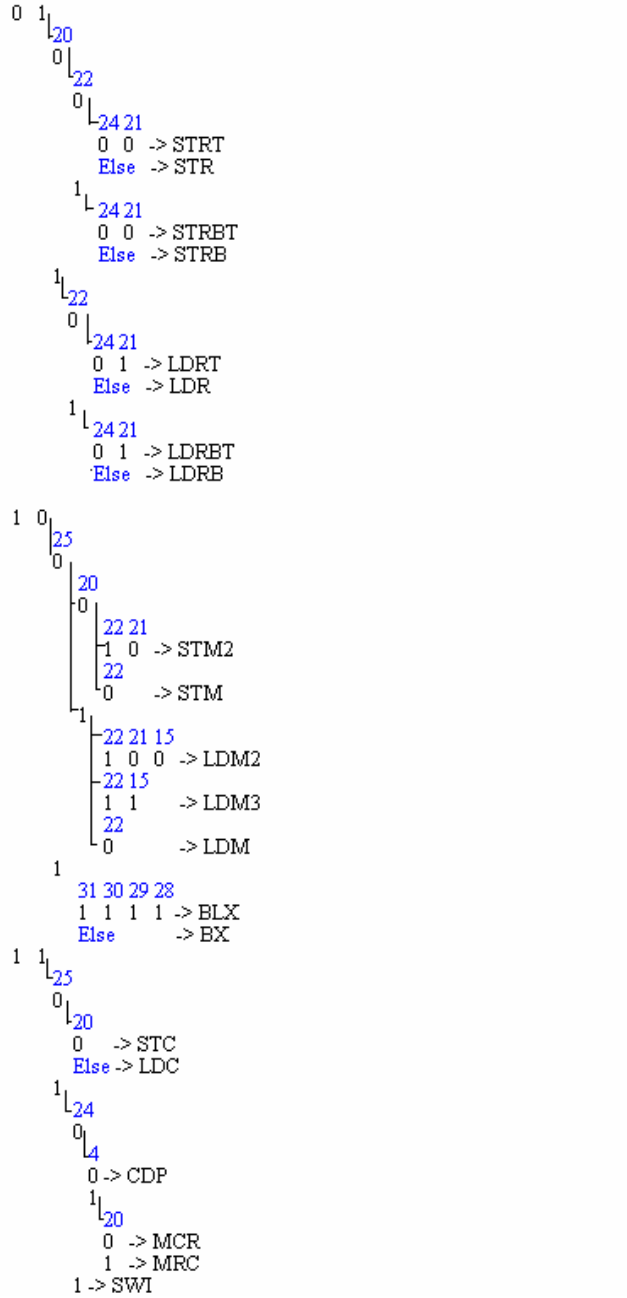

Figure 5: Machine Code Decode Table for Bits 27-26 = 01, 10 and 11