Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department
CS 401 – Graduation Project

Graduation Project
General-Purpose Computer and Game Console
Design of Emulation Software

Student:

- Mostafa Abd El-Aziz (65)

Supervisor:

- Prof. Dr. Layla Abo Hadid

Abstract:

First step of the project is to build an emulator for the proposed MIPS machine. It is important to write the emulator before working on VHDL code so that I make sure that I understand the instruction set correctly, and to make any needed modifications to the design before sketching it on the FPGA device.

This document describes the design of the emulator, and shows how its components interact with each other to properly simulate the proposed machine. Pre-compiled firmware (or operating system) is loaded to a simulated ROM in host machine's RAM, and executed by the simulated CPU.

Components:

Figure 1 shows the structural design for the emulator. The emulator consists of the following components:

- Clock pulse generator (clock.c)
- Central Processing Unit (cpu.c)
- Memory Interface, ROM, and RAM (mem.c)
- VGA Interface (vga.c)
- Keyboard Interface (kbd.c)

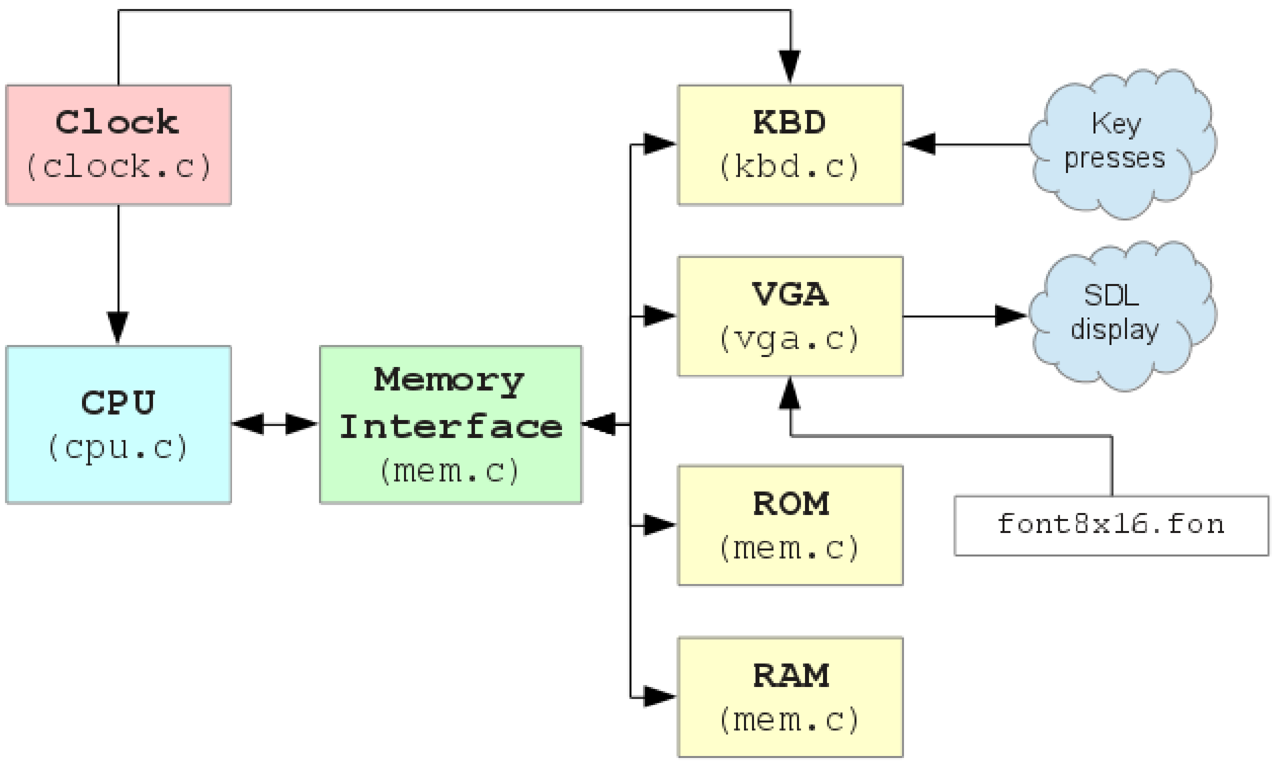

Figure 1. Emulator components.

1. Clock Pulse Generator (clock.c):

The function of this module is to provide synchronization between other components. It generates clock pulses (i.e, calls cpu_clk() and kbd_clk() frequently in a loop) and listens to external events from SDL (Simple DirectMedia Layer), like button press and exit events.

2. Central Processing Unit (cpu.c):

This module is responsible for fetching instructions from memory and executing them. It simulates the CPU module of the real machine. The CPU implements MIPS-1 instruction set architecture. The source code in cpu.c is organized such that it is very similar to the VHDL code of the processor, this implies that cpu.c implements the pipeline as it is.

Figure 2 shows the pipeline design of the processor. It is based on the original pipeline design of a subset of MIPS instruction set described in *Computer Organization and Design*.

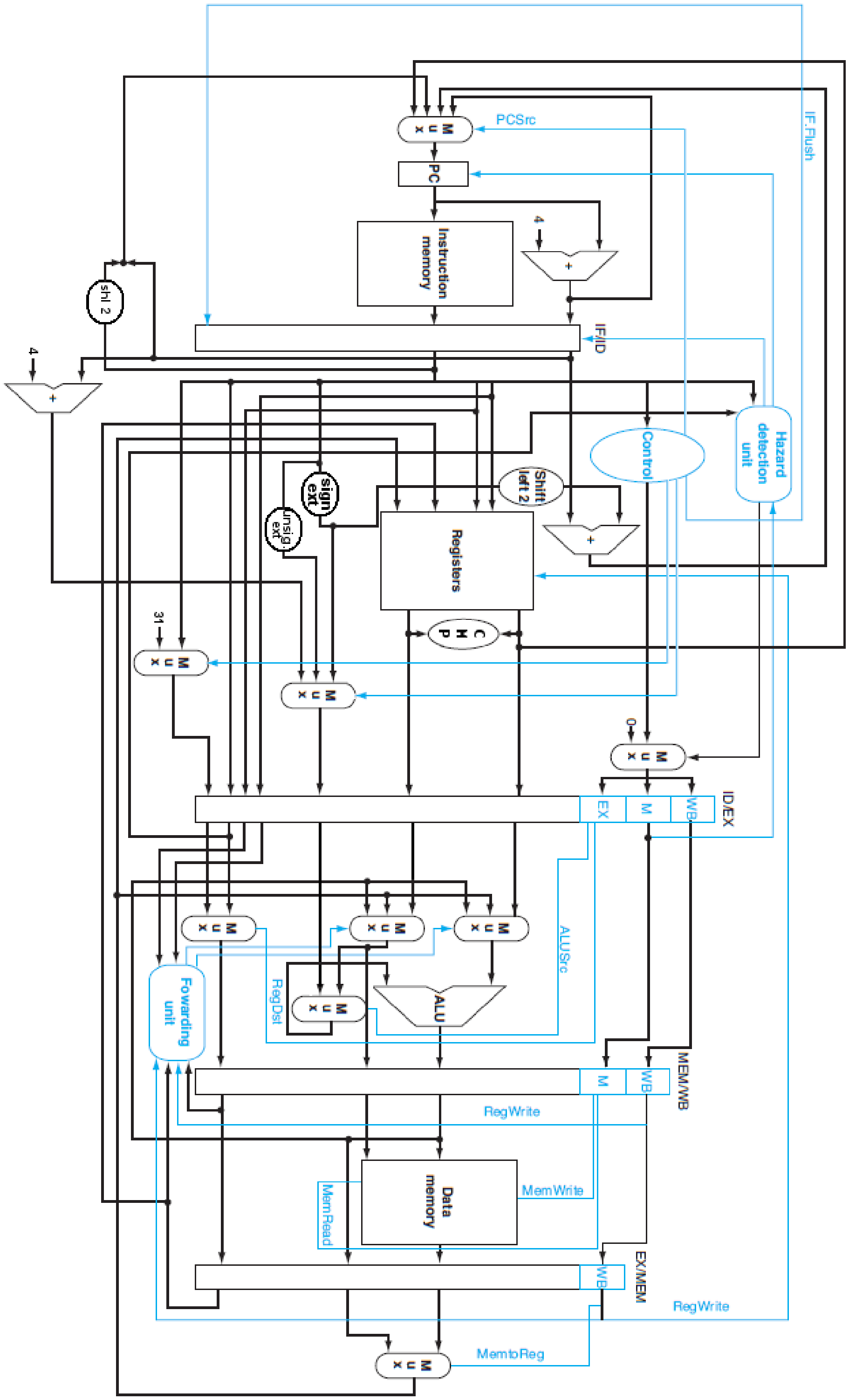

Figure 2. Generalized pipeline design for MIPS

3. Memory Interface (mem.c):

This module provides an interface between CPU and other components in the system (ROM, RAM, Keyboard, and VGA). ROM and RAM are two arrays in host memory, allocated when the program starts.

MEMREAD and MEMWRITE signals of the processor are translated into mem_read() and mem_write() calls. These functions in turn translate the request into read/write from/to ROM or RAM arrays in memory, or translate the request into calls to keyboard and VGA modules.

Figure 3 shows the memory map implemented by mem.c. Address passed to mem_read() and mem_write() is checked to know which module is repsponsible to handle the request.

First Address	Last Address	Region Name	Size
0x00000000	0x0000FFFF	ROM	64K
0x00010000	0x00017FFF	RAM	32K
0x00018000	0x0001FFFF	VGA	32K
0xFFF00000	0xFFFFFFFF	KBD	1M

Figure 3. Memory map.

4. VGA Interface (vga.c):

This module simulates the video graphics interface of the real machine. The real interface allows software to print text on screen. Currently, it supports resolution of 640x480 and screen is divided into 80 columns and 30 rows. Every character is plotted in 8x16 pixel square.

The VGA appears to software as a memory buffer (in 0x00018000 to 0x0001FFFF region). A word write to 0x18000 sets the ASCII code of character at row 0, col 0. Word write to 0x18004 sets its attribute. Word write to 0x18008 sets the ASCII code of character at row 0, col 1, and so on.

The code in vga.c catches memory writes to VGA buffer, and draws appropriate content on SDL screen, using an 8x16 font file (font8x16.fon).

5. Keyboard Interface (kbd.c):

This module catches keypresses on SDL screen, and translates them into “scancodes”, then sends the scancodes to software (just like the real PS/2 keyboard and PS/2 interface do). Figure 4 shows scancode map assumed by the running software.

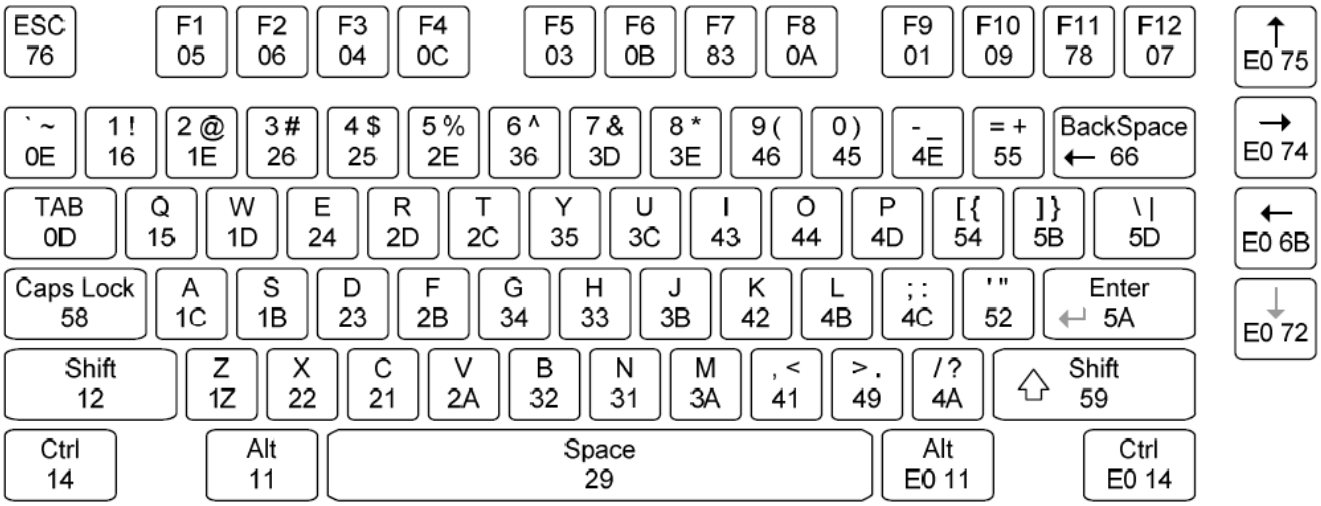

Figure 4. PS/2 keyboard scancode map.

Compilation:

Emulator source code is written in C and compiled using gcc, and uses SDL for GUI. A Makefile is distributed with software to automate compilation process. Just type “make” in terminal while shell's current working directory is the directory that contains source code (mipsemu directory) as shown in figure 5.

Figure 5. Compiling mipsemu.

If “make” completes successfully, the compiler has generated “./mipsemu” executable file that is ready to use.

Usage:

./mipsemu command takes one parameter, which is the path to firmware image that will be loaded to ROM when the emulator starts. Figure 6 shows how mipsemu can be started using a pre-compiled firmware image.

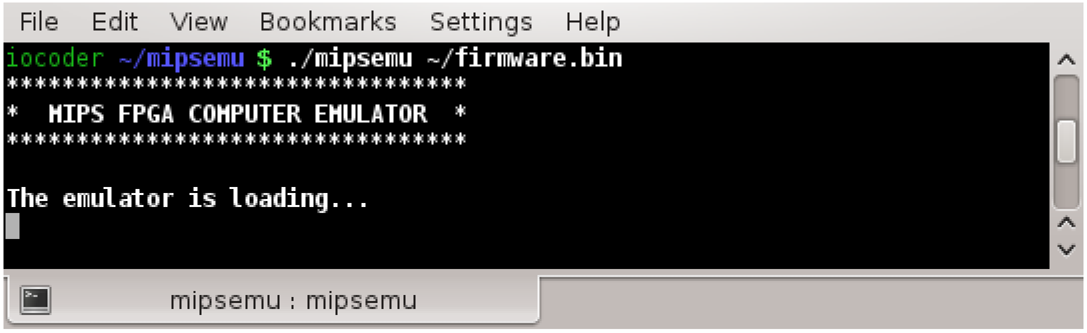

Figure 6. Running mipsemu.

Screenshots:

Here are some snapshots for the emulator with firmware.bin loaded into its ROM:

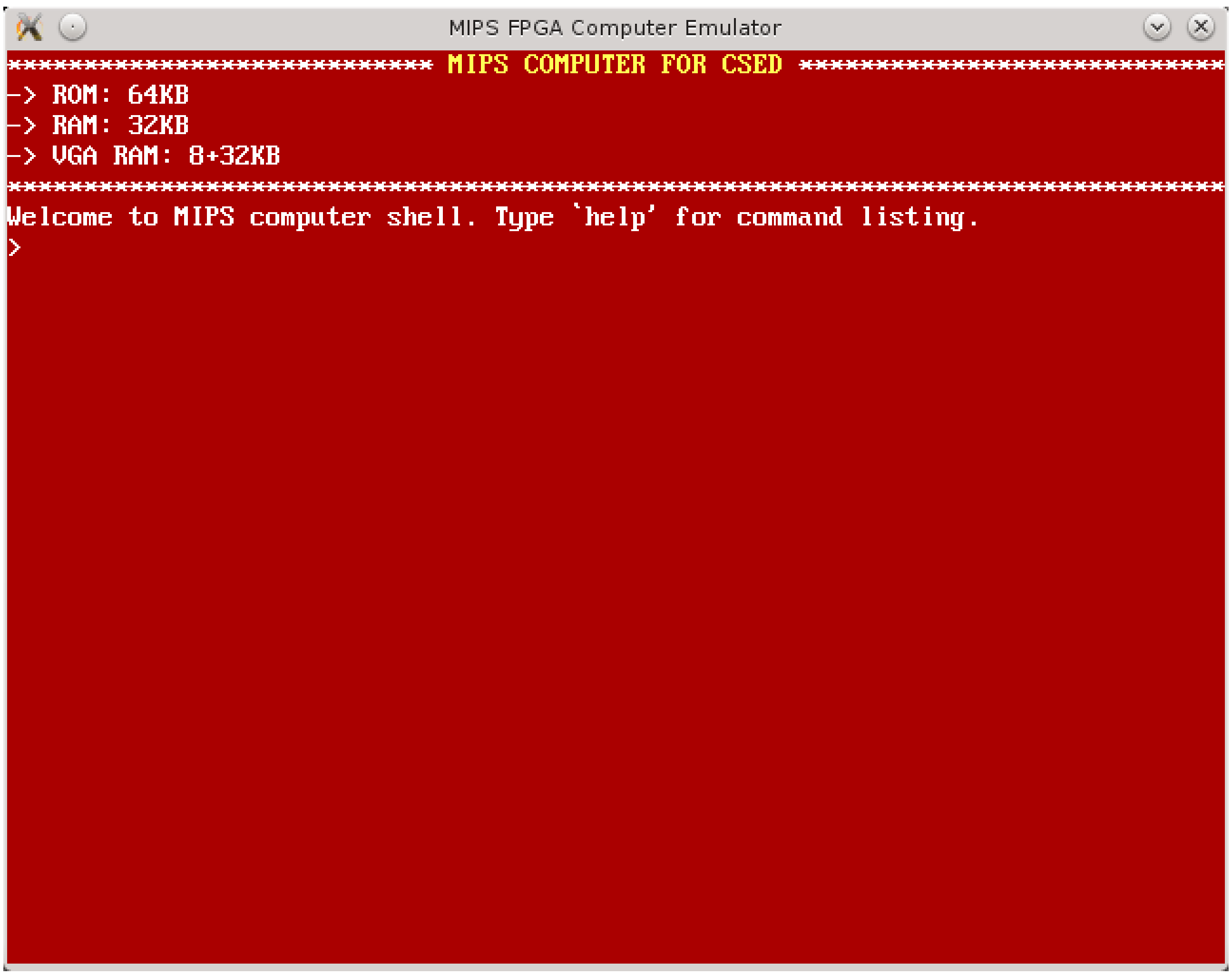

Figure 7. Firmware shell prompt

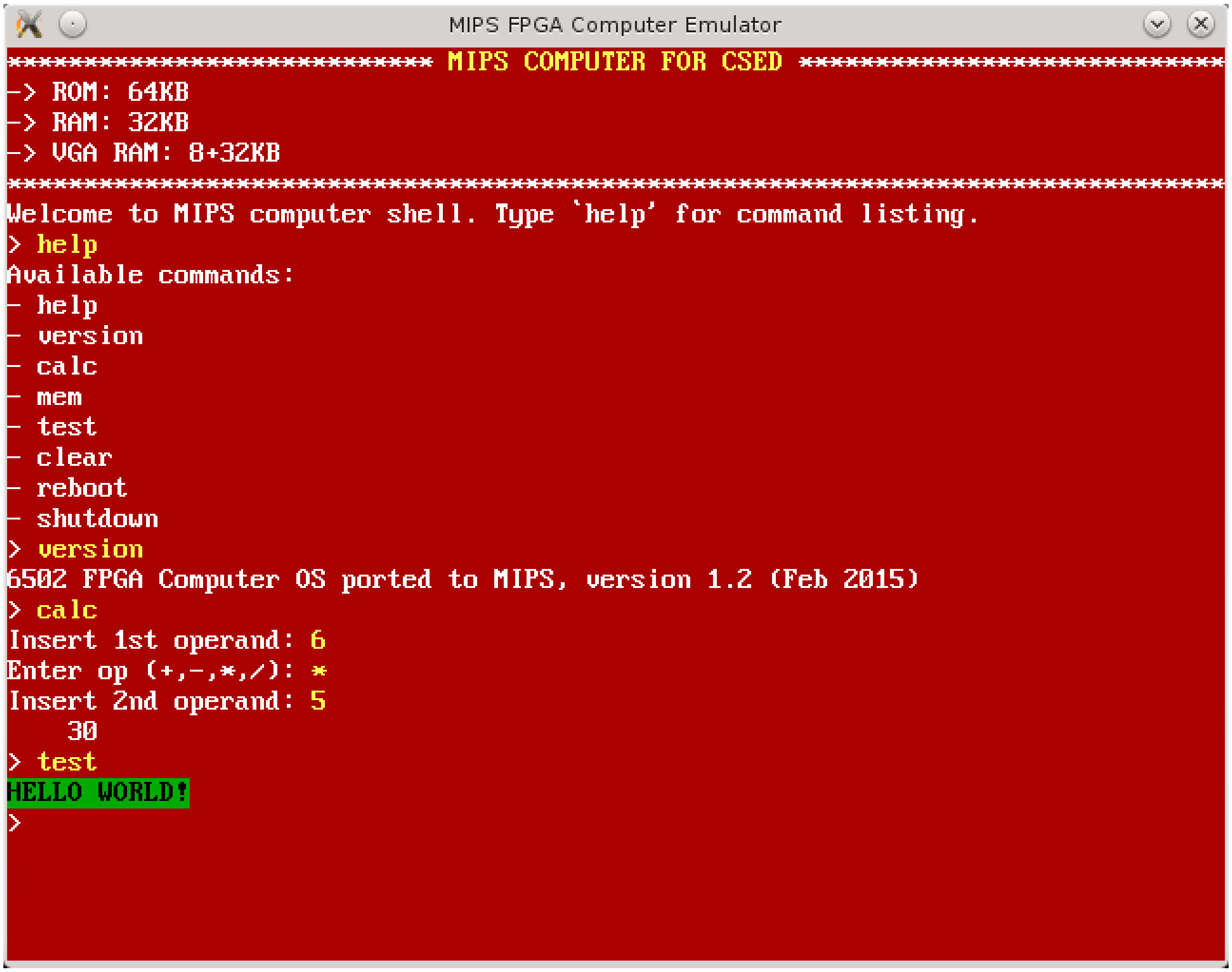

Figure 8. Commands are written by keyboard and caught by firmware