MIPSfpga by Imagination

Lab 3

MIPS Assembly Programming

These materials produced in association with Imagination.
Join our University community for more resources.
community.imgtec.com/university

MIPSfpga Lab 3: MIPS Assembly Programming

1. Introduction

In this lab you will learn to program the MIPSfpga core by writing a MIPS assembly program to calculate the Fibonacci numbers.

2. MIPSfpga Assembly Tutorial

As in Lab 2, you will use Imagination's Codescape tools to compile MIPS assembly language programs and then download, debug, and run them on the MIPSfpga core. You will write the same Fibonacci program from Lab 2 that calculates and outputs the Fibonacci numbers from 1-11, but this time using MIPS assembly language. You will also learn to step through the MIPS assembly language program and debug it if you have errors.

Example MIPS Assembly Program

Before writing your own program, first let's take a look at an example MIPS assembly language program. Browse to the Lab03_Assembly\ReadSwitches directory. Open the main.S MIPS assembly file using a text editor such as Notepad++ or Wordpad. The main.S file contains the ReadSwitches program, as also shown in Figure 1.

```
# $12 = address of switches, $13 = address offset of switches
# $10 = switch values
.globl main

main:
   lui   $12, 0xbf80    # $12 = address of LEDs (0xbf800000)
   addiu $13, $12, 4     # $13 = SW address offset

readIO:
   lw    $10, 0($13)     # read switches: $10 = switch values
   sw    $10, 0($12)     # write switch values to LEDs
   beq   $0, $0, readIO # repeat
   nop                   # branch delay slot
```

Figure 1. ReadSwitches MIPS assembly program

This program performs the following steps:

1. Loads the memory-mapped addresses of the switches and LEDs into registers $12 and $13.

   ```
   readIO:
   lui   $12, 0xbf80    # $12 = address of LEDs (0xbf800000)
   addiu $13, $12, 4     # $13 = address of switches
   ```

2. Reads the value of the switches on the FPGA board:

```
lw    $10, 0($13)    # read switches: $10 = switch values
```

3. Writes that value to the LEDs

```
sw    $10, 0($12)    # write switch values to LEDs
```

4. And repeats

```
beq   $0, $0, readIO # repeat
nop                  # branch delay slot
```

The `.globl main` assembler directive at the top of the file is required so that the compiler can find the `main` function when linking the program with the boot code.

The details of how the program reads the switches and writes to the LEDs was described in the MIPSfpga Getting Started Guide. A write to memory address 0xbf800000 will output a value to the LEDs and a read to address 0xbf800008 will return the value of the switches.

Compiling, Running, and Debugging

Now you will compile, run, and debug the ReadSwitches MIPS assembly program on the MIPSfpga core. Each step is described in detail below.

Step 1. Download the MIPSfpga system onto the Nexys4 DDR board
Step 2. Compile the MIPS assembly program
Step 3. Load the program onto MIPSfpga using the Bus Blaster probe or UART interface
Step 4. Debug the program using gdb as needed (if you're using the Bus Blaster probe)

Step 1. Download the MIPSfpga system onto the Nexys4 DDR board

Refer to Lab 1 or 2 instructions if you have forgotten how to do this.

Step 2. Compile the program

Now compile the ReadSwitches example Assembly program by opening a command window (Start menu → cmd.exe) and changing to the following directory:

MIPSfpga_Labs\Labs\Part1_Intro\Xilinx\ Lab03_Assembly\ReadSwitches

For example, if MIPSfpga_Labs is located at C:\ type:

```
cd C:\MIPSfpga_Labs\Labs\Part1_Intro\Xilinx\Lab03_Assembly\ReadSwitches
```

Then compile the example MIPS assembly program by typing `make` at the prompt in the command window:

```
make
```

This runs the Makefile to compile the code found in main.S. The executable will include not only the user code found in main.S but also the boot code, found in boot.S and the other .S files. Open and view the Makefile using a text editor such as Notepad++. The Makefile is almost the

same as the one used in Lab 2 for compiling the C code, except now the user program (main.S) is listed under the ASOURCES (assembly source files) and there are no C source files (CSOURCES). A mix of C and assembly source files could be used by having files listed under both CSOURCES and ASOURCES.

```
ASOURCES= \
boot.S main.S
```

As in Lab 2 with C programs, you can clean the directory of files created during compilation by typing the following at the command prompt:

```
make clean
```

If you cleaned the directory, compile the program (main.S) again by typing `make` at the command prompt. You should now see main.o, program_dasm.txt, and program.elf. main.o is the object file and program.elf is the ELF (executable and linkable format) executable that you will use to load the program onto the MIPSfpga core. program_dasm.txt shows the disassembly of the executable (boot and program code). It is a human-readable version of the ELF executable file. Open program_dasm.txt using a text editor. As in Lab 2, the top of the file shows the boot code, starting at 0x9fc00000. This code will be placed at physical address 0x1fc00000, which is the first instruction fetched upon reset of the MIPSfpga core.

Near the bottom of the file, you can view the user code from main.S, starting at 0x80000204. This will map to physical addresses starting at 0x00000204.

Figure 2 shows the user code from program.txt (or program.dis, with file listing information between code lines deleted). This file shows the assembly code interweaved with the memory address and machine code for the instruction.

```
80000204 <main>:
80000204:  3c0cbf80   lui   t4,0xbf80
80000208:  258d0004   addiu t5,t4,4

8000020c <readIO>:
8000020c:  8daa0000   lw    t2,0(t5)
80000210:  1000fffe   b     8000020c <readIO>
80000214:  ad8a0000   sw    t2,0(t4)
```

Figure 2. program.txt: Beginning of user code starting at 0x80000204

For example, the 32-bit machine instructions for `lui $12, 0xbf80` is 0x3c0cbf80, located at memory address 0x80000204. That machine instruction is followed immediately by the next machine instruction (0x258d0004 – i.e., `addiu $13, $12, 4`), which is located 4 bytes (32 bits) later at memory address 0x80000208. The next instruction (`lw t2, 0(t5) = lw $10, 0($13)` = 0x8daa0000) is at memory address 0x8000020c, and so forth. Notice that the

compiler placed the `sw` instruction that writes to the LEDs (`sw $10, 0($12)`) into the branch delay slot at address 0x80000214, so the `nop` placed in the branch delay slot of the MIPS assembly program (at 0x80000214) is never executed.

Step 3. Load the program onto MIPSfpga using Bus Blaster

Now that the example MIPS assembly program is compiled, load it onto the MIPSfpga core using the Bus Blaster probe. Recall that you can also load the program using the existing USB cable (see the Getting Started Guide). If MIPSfpga isn't already running on the Nexys4 DDR board, load it onto the board as described in Lab 2. Then, connect the Bus Blaster probe to the Nexys4 DDR board, as also described in Lab 2.

Now open a command shell and change to the MIPSfpga_GSG\Scripts\Nexys4_DDR directory. For example, if the MIPSfpga_GSG folder is on the C drive, type the following at the shell prompt:

```
cd C:\MIPSfpga_GSG\Scripts\Nexys4_DDR
```

Now run the same script used in Lab 2 that (1) sets up a connection to the MIPSfpga core (using the Bus Blaster probe and OpenOCD), (2) loads the program onto the MIPSfpga core, and (3) allows you to debug the program using gdb.

In the command shell, type:

```
 loadMIPSfpga.bat
C:\MIPSfpga_Labs\Labs\Part1_Intro\Xilinx\Lab03_Assembly\ReadSwitches
```

After running the script, you will see the ReadSwitches program running on the MIPSfpga core. The program repeatedly reads the value of the switches and writes that value to the LEDs. Toggle some of the switches at the bottom of the Nexys4 DDR board and watch as the corresponding LEDs light up.

Step 4. Debug the program using gdb as needed

Now you can use the gdb debugger to set breakpoints, probe register values, etc. Refer to Lab 2 instructions for how to do this.

3. Assembly Code with Function Calls

You will now run MIPS assembly code with function calls. Browse to the Lab03_Assembly\ReadSwitchesFunctions directory and open main.S (also shown in Figure 3). This program uses two function calls: one to read the switches (`peek`), and one to write to the LEDs (`poke`).

```
.globl main

main:
   lui   $12, 0xbf80       # $12 = address of LEDs
   addi  $13, $12, 4       # $13 = address of switches

displaySW:
   add   $a0, $13, $0      # put address of switches in $a0
   jal   peek              # call peek function
   nop                     # branch delay slot
   add   $a0, $v0, $0      # move the return value into $a0
   add   $a1, $12, $0      # move address of LEDs in $a1
   jal   poke              # call poke function
   nop                     # branch delay slot
   beq   $0, $0, displaySW # repeat
   nop                     # branch delay slot

peek:
   lw    $v0, 0($a0)       # read I/O register value into $v0
   jr    $ra               # return to point of call
   nop                     # branch delay slot

poke:
   sw    $a0, 0($a1)       # write values to I/O register
   jr    $ra               # return to point of call
   nop                     # branch delay slot
```

Figure 3. ReadSwitchesFunctions MIPS assembly program that uses function calls

The program, which starts just after the `main` label, loads the addresses of the LEDs and switches into $12 and $13. Then it puts the address of the switches in the 0th argument register ($a0) and calls the `peek` function using `jal peek`.

The `peek` function, located at the `peek` label, reads the value of the switches and places that value in the return register ($v0). The function completes by returning to the place it was called, using `jr $ra`.

The `main` function then places the value of the switches into $a0 and the address of the LEDs into $a1 and calls the `poke` function. The `poke` function writes $a0 to the LEDs.

Compile and load the ReadSwitchesFunctions program using `make` and the `loadMIPSfpga.bat` script as described above. Remember that you can also view the program_dasm.txt file to see how the MIPS assembly code was compiled.

4. Fibonacci Numbers

Now use the ReadSwitchesFunctions example MIPS assembly program as a starting point to write your own program. As in Lab 2, you will write a program that will calculate and display the first 11 Fibonacci numbers; however, this time you will write it using MIPS assembly language. Refer to Lab 2 if you need a refresher on Fibonacci Numbers.

Make a copy of the Lab03_Assembly\ReadSwitches folder and rename the new folder Fibonacci. Write your program in the main.S file in the Fibonacci folder. Your program should compute the Fibonacci numbers for n = 1…11 and output each Fibonacci number to the LEDs. The LEDs should display the Fibonacci numbers in binary, with a delay between each number so that they are viewable.

Once you have finished writing your program, use the `make` command to compile it. If there are any errors, fix them and recompile.

After your Fibonacci program compiles without errors, load it onto the MIPSfpga core, run it, and confirm that it works as expected.