EEE 498/591: Project Part 2

February 15, 2017

[15 Points] Due at start of class Tue Feb. 28th
Summary: This part is based on using the MIPS toolchain including gcc compiler, assembler and the processor simulator.
Pair Programming: This assignment may be completed in pairs. Our expectation is that you will complete the assignment together: discussing concepts, debugging errors, and learning from each-other. You may submit your assignment individually, though you will be evaluated with the same criteria. Only one assignment needs to be submitted for each programming pair. You may select a partner for each part of the project.
Study Group: If you haven't already it is a good idea to form a study group of 5-7 students. You will find your study group to be more effective if you meet at a consistent time to discuss the course lecture, reading, and assignmentes.
Tools: The following environment variables will help. They also point you at the tool chain on the eecad machines.

```
setenv MIPS_ELF_ROOT  \
/afs/asu.edu/class/e/e/e/eee625b/MIPStools/imgtec/Toolchains/\
mips-mti-elf/2016.05-03
setenv MIPS_SIM \
/afs/asu.edu/class/e/e/e/eee625b/MIPStools/imgtec/Simulators/qemu/2.5.0.2.0
setenv MIPS_LINUX_GNU_ROOT  \
/afs/asu.edu/class/e/e/e/eee625b/MIPStools/imgtec\
/Toolchains/mips-mti-linux-gnu/2016.05-03
```

You should feel free to download the tools to you own PC. They are free on the Imagination Technologies site. You want to use the older processors (this is the 'mti' part of the path). Follow the examples in the documentation to get familiar with the tools. As we discussed in class, we will build 'bare metal' applications–stick to that. Please help each other with the tool usage. It is pretty simple but minor mistakes can be very hard to catch on your own. Also, the book carefully explained saving values on the stack and you can 'cheat' by writing the appropriate C code and looking at the assembly code generated.

1. **[0 Points] Prepare your submission:** Your should create a directory that you will submit through email. We will refer to that directory as "$DIR". It is up to you where you create your directory, but a good place might be " /eee591/ProjPart2"[1] Your

[1] You can use "mkdir -p /eee591/ProjPart2" to create all of the directories in one go

directory should contain a file to help us resolve your identity. Your file should be "$DIR/submit.txt" with a line indicating your name and posting id. It should contain your last name (as it appears in blackboard), followed by your first name, following by your posting id. There should be a single line for each of you. For example:

```
Hamm, John, 1234569
Douglas, Michael, 135711
```

2. [**0 Points**] **Project Directory** Place the programs in the appropriately named .c or .S files and show the results from the simulator by both appropriate dumps and showing the bus activity.

3. [**7 Points**] **Write a simple program in the C language:** Implement a program that sums the values in a three dimensional array (at least 3 items in each dimension (27 values). Initialize the values and then call a function to sum the values. Have the function always return 0. Compile and view the resulting assembler code. Some hints:

```
   .../bin/mips-mti-elf-gcc -S -o test2.S test.c
more test2.S
   .../bin/mips-mti-elf-gcc -S -O2 -o test3.S test.c
more test3.S
```

   Then simulate it. Then, recompile but with the optimization set using the -O2 switch. Note any differences in the assembled code. Write a paragraph or two on what you see and why you think you see it. Then, change the return value to be the sum and repeat. Explain your results (refer to the .S files) in a .txt file called 'C-program.txt'. Show snippets to demonstrate what happened. Hand in the c-programs as c3darray-add1.c, c3darray-add2.c, c3darray-add3.c clearly commenting the lines changed in each subsequent version. Include the .S produced for each with the same names.

4. [**8 Points**] **Register file dumping routine in MIPS assembly language:** For this portion implement an assembly language program that writes the values of all the register file contents (all 32 registers) to a fixed set of memory locations. The output should be in words as: 0, 0, 1, contents_of_$1, 2, contents_of_$2, ... 31, contents_of_$31. Your routine should be callable from a c-program as a c function with one parameter (the base memory word) and it should return with a logic 1 return value. No register values should be modified when this function returns. To confirm this, try it twice in a row. To put known values in all the registers, feel free to use in-line assembly code. This will be required to help the testbench in later projects so get it right.

   Link your assembled .S function to a .c program and show they function together in a text file report RFdump.txt. Hand in RFdump.S and your test c program.

   More on tool usage: Here are some useful commands and some sample output. This is a learn by doing class, so experiment and use the –help options (and sub-options to understand the usage and what you get. to compile:

$MIPS_LINUX_GNU_ROOT/bin/mips-mti-linux-gnu-gcc -g -static test.c -o test

to simulate (must go to background, so you can run gdb):

$MIPS_SIM/bin/qemu-mips -g 1234 -cpu 74Kf test &

Note that you must put it in background, since you need to do more work in the same window. to watch it in gdb:

$MIPS_LINUX_GNU_ROOT/bin/mips-mti-linux-gnu-gdb test

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
__start () at ../sysdeps/mips/start.S:83
83 ../sysdeps/mips/start.S: No such file or directory.
(gdb) b main
Breakpoint 1 at 0x400cb0: file test.c, line 4.
(gdb) continue
Continuing.

Breakpoint 1, main (argc=1) at test.c:4
4   a = 1;
(gdb) info locals
a = 0
b = 0
c = 0
d = 0
(gdb) b 10
Breakpoint 2 at 0x400ce8: file test.c, line 10.

(gdb) info registers
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 00000001 00000015 0000000d 00000001 76ffeea4 76ffeeac 00000000
            t0       t1       t2       t3       t4       t5       t6       t7
 R8   0049d570 7f4c484f ae61657e 81010100 2f2f2f2f 44533230 31345f30 312f6c69
            s0       s1       s2       s3       s4       s5       s6       s7
 R16  004013c0 00000000 00000000 00000000 00000000 00000000 00000000 00000000
            t8       t9       k0       k1       gp       sp       s8       ra
 R24  0000000f 00400ca0 00000000 00000000 004a5080 76ffedc0 76ffedc0 00400f04
            sr       lo       hi      bad    cause       pc
      21000010 999999a3 00000025 00000000 00000000 00400ce8
```

---

to get an object file dump with the code (both C and asm):

```
\   .../bin\/mips-mti-elf-objdump -S -l test \> test.objdump.txt
```

Play around with the options, to just compile but not assemble:

```
\   .../bin/mips-mti-linux-gnu-gcc -g -static test.c -S -o test.S
```

5. **Submission** Once you have completed your submission directory you should create a "tarball" and email it to the submission email. You should create your tarball with the following command:

   ```
   tar -czvf ProjPart2.tar.gz ProjPart2
   ```

   You should email your tarball to (only after the 20th and before the deadline):

   ```
   asu.eee.591ca.spring17@gmail.com
   ```

   You should use the subject line:

   ```
   ASU eee591 ProjPart2
   ```

   This will be graded by the grader, who will access each of your .tar.gz files. Be VERY CLEAR in what you include and in your discussions of the results. Make this easy for the grader and the grader may not nit-pick. Clearly show that the programs work in the .txt files