University of Colorado
Colorado Springs

CS 4200, Spring 2022: Computer Architecture

Instructor: Mong Sim

Homework Five

Andrew Schmidt

2022/04/01

Project Overview

This project we were given a CPP file containing an iterative and recursive function for factorials, along with a short main section that would call the functions and output a test value to ensure the expected result was returned from the function. I then added my own integrative and reclusive functions for 2^p, along with some code in main that would output a test case to ensure the correct result was being returned from the functions. These additions can be seen in the screenshots below.

```
//------------------------------------------------------------------------
// Performs a math operation 2^num in a iterative method
//------------------------------------------------------------------------
uint32_t    Iterative_Power(uint32_t  num)
{
    uint32_t x;
    uint32_t retVal = 2;
    for (x = num; x > 1; x--)
    {
        retVal *= 2;
    }

    return retVal;;
}

//------------------------------------------------------------------------
// Performs a math operation 2^num in a recursive method pass by value
//------------------------------------------------------------------------
uint32_t    RPower_Value(uint32_t  num)
{
    if (num > 1)
    {
        return 2 * RPower_Value(num - 1);
    }

    return 2;
}
```

```
fVal = 0;
num = 6;
fVal = Iterative_Power(num);
printf("Iterative Power of 2^%d = %d\n", num, fVal);

fVal = 0;
num = 6;
fVal = RPower_Value(num);
printf("Recursive Power of 2^%d = %d\n", num, fVal);
```

As you can see by the screenshots, they are very similar to the functions provided and only needed a few adjustments to change the functionality of them and ensure the correct print statements were being outputted. This was the easy part of the assignment as implementing these features are trivial and with most the code being able to be copied from the other provided examples, a quick test and a could changed operations allowed for easy code building. I made sure to run a couple values through to make sure that inputs give the correct outputs assuming that the values are positive and valid, then moved on to step 2 of the Homework.

Creating a Build Environment

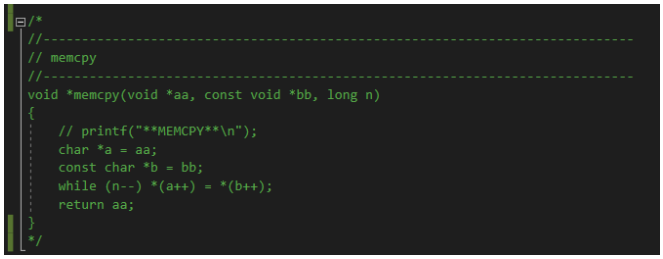
This process was also very similar to the process we saw in the bootloader project, where we take our dhrystone files and copy them into the new folder, we can then use ./d.bat to remove a majority of the Dhrystone specific files and manually remove that remaining few. We then open the c.bat file and take out all the dhry files that would normally be compiled and change the main.c to be our HW05.cpp file, along with

```
cls
del *.o
del *.disass
del *.srec
del *.ihex
del *.rom

riscv64-unknown-elf-g++ -c -mabi=ilp32 -march=rv32im boot.s -g -o boot.o
riscv64-unknown-elf-g++ -c -mabi=ilp32 -march=rv32im -DUSE_MYSTDLIB -DTIME -DRISCV -g stdlib.c -o stdlib.o
riscv64-unknown-elf-g++ -c -mabi=ilp32 -march=rv32im -DUSE_MYSTDLIB -DTIME -DRISCV -g HW05.cpp -o HW05.o
riscv64-unknown-elf-ld -nostdlib -T rv32im.ls boot.o stdlib.o HW05.o -o HW05.elf
riscv64-unknown-elf-objcopy -O binary HW05.elf HW05.bin
riscv64-unknown-elf-objcopy -O srec HW05.elf HW05.srec
riscv64-unknown-elf-objcopy -O ihex HW05.elf HW05.ihex
riscv64-unknown-elf-objcopy -R rom HW05.elf HW05.rom
riscv64-unknown-elf-objdump -D HW05.elf > HW05.disass
```

updating all the other file names to go along with our new HW05 naming scheme. We also update the compiler from gcc to g++ as noted by the project description and save the file. Now that its set up and ready to go, we can run pp.bat to set out path, then run the newly created c.bat file.

When I first ran this I got an error message back from the stdlib.c for the memcpy function as it was throwing errors about casting void * to char*, but since this code was supplied to us and it wasn't needed for our project I belive that this can be commented out. This might pose a problem for our next project if I need to run the program and implement memory copying to insert and remove break points, however there are ways to do these operations without the memcpy function and I can always work it back in later. I have added comments to indacte this change to the file for future reference if needed.



```
/*
//------------------------------------------------------------------------------
// memcpy
//------------------------------------------------------------------------------
void *memcpy(void *aa, const void *bb, long n)
{
    // printf("**MEMCPY**\n");
    char *a = aa;
    const char *b = bb;
    while (n--) *(a++) = *(b++);
    return aa;
}
*/
```

Running in Softcore

Once this was commented out the c.bat file ran perfectly and create the bin and elf files that will be needed for this and future projects. I could then take the path of the bin file and pass it as the debugging argument in the softcore provided for the class as we have done in all the past assignments to run the compiled program and got my output, which I could then save to a file called output.txt. Here it outputs our main code that shows the functions all running with test values to ensure they work correctly and give the correct output, along with some data that is printed by the softcore itself.

Files Submitted

1. Homework05_Andrew_Schmidt.pdf — This file (2 Page write up)
2. Output.txt — Program output from softcore
3. HW05.cpp — Iterative and Recursive code functions
4. Zip Folder — Contains all build files used for the homework