University of Colorado Colorado Springs

CS 4200, Spring 2022: Computer Architecture

Instructor: Mong Sim

Project Three

Andrew Schmidt

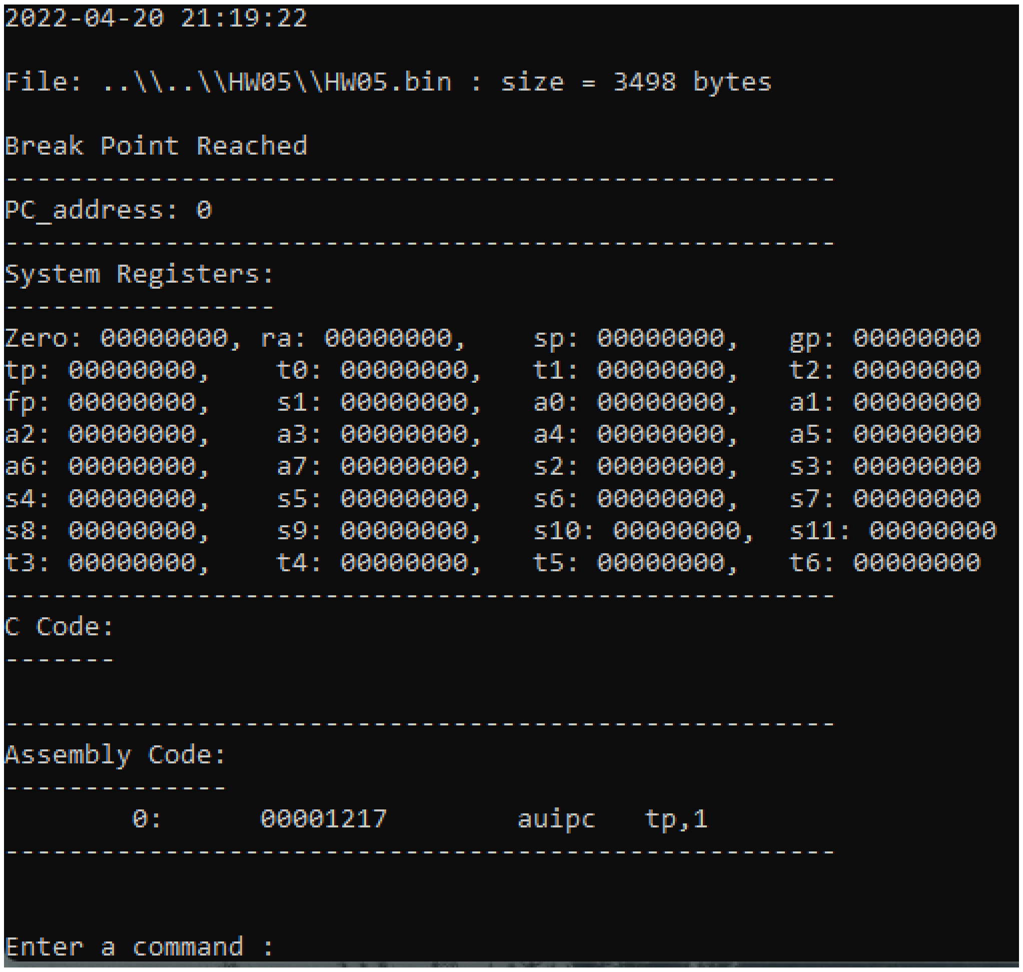
2022/04/28

Project Overview:

For this final project, project 3, we were tasked with designing a program debugger that would allow for breakpoints to be set, removed, and single step feature to work on top of the provided Softcore system that we used on our other 2 projects. Making sure to follow a given directory layout and relative paths provides a file system that could be transferred to any device and have the expectation of running given the Softcore was set up correctly. We test our debugger program with our HW05.cpp file that we compiled with our given Dhrystone and passing in the bin file to correctly run the program, along with using the given readelf function and our HW05.elf file to get some more decodedLine.txt files that we need for extracting the C-code and various other operations.

Step 1: The setup

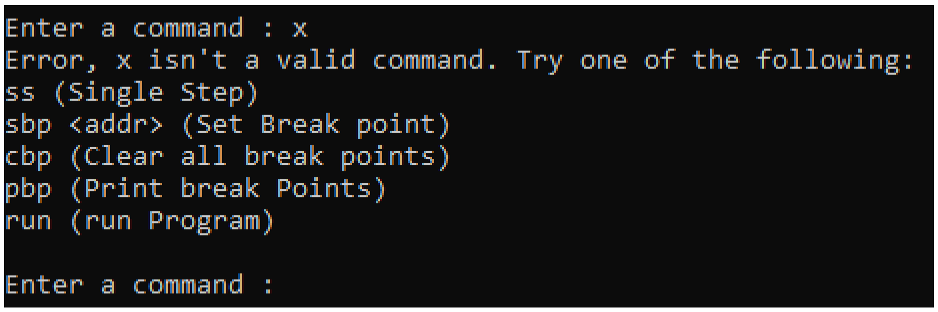
When we launch our program in the softcore, we want it to jump to our program so that any break points can be set before the program runs. Even though there is a _kbhit function that would detect a keyboard hit while running and begin debugging, because the HW05 file is so small and runs so fast this would be nearly impossible to test without incorporating an infinite loop in the HW file or adding some kind of delay between instruction. By simply setting a flag at the start of the program we can detect when it's the first instruction through and send it to our code before it executes the instruction. When this is run for the first time, the following output is provided as seen to the right. Since it's the first time through all the registers are still at 0, and the first line of assembly code is printed out, along with the PC address before prompting for a command to be entered. As you can see some of the output is missing, which is discussed later on in currently missing and needs improvements section.



```
2022-04-20 21:19:22

File: ..\\..\\HW05\\HW05.bin : size = 3498 bytes

Break Point Reached
----------------------------------------------------
PC_address: 0
----------------------------------------------------
System Registers:
-----------------
Zero: 00000000, ra: 00000000,    sp: 00000000,   gp: 00000000
tp: 00000000,    t0: 00000000,   t1: 00000000,   t2: 00000000
fp: 00000000,    s1: 00000000,   a0: 00000000,   a1: 00000000
a2: 00000000,    a3: 00000000,   a4: 00000000,   a5: 00000000
a6: 00000000,    a7: 00000000,   s2: 00000000,   s3: 00000000
s4: 00000000,    s5: 00000000,   s6: 00000000,   s7: 00000000
s8: 00000000,    s9: 00000000,   s10: 00000000,  s11: 00000000
t3: 00000000,    t4: 00000000,   t5: 00000000,   t6: 00000000
----------------------------------------------------
C Code:
-------

----------------------------------------------------
Assembly Code:
--------------
        0:      00001217        auipc   tp,1
----------------------------------------------------


Enter a command :
```

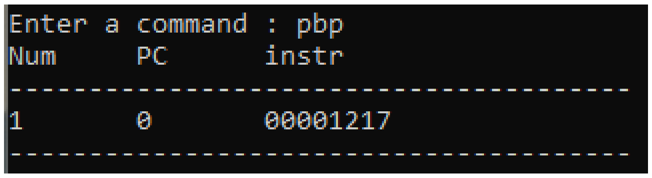
Step 2: Command detection

Next is reading in commands for the user to tell the program what they would like done, here is an example of a given bad command where it prompts the user to enter a valid command and circles back to asking for a command to be entered.

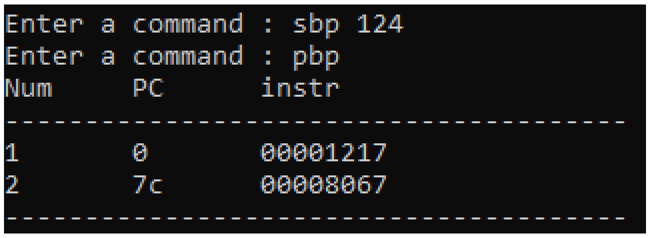


```
Enter a command : x
Error, x isn't a valid command. Try one of the following:
ss (Single Step)
sbp <addr> (Set Break point)
cbp (Clear all break points)
pbp (Print break Points)
run (run Program)

Enter a command :
```

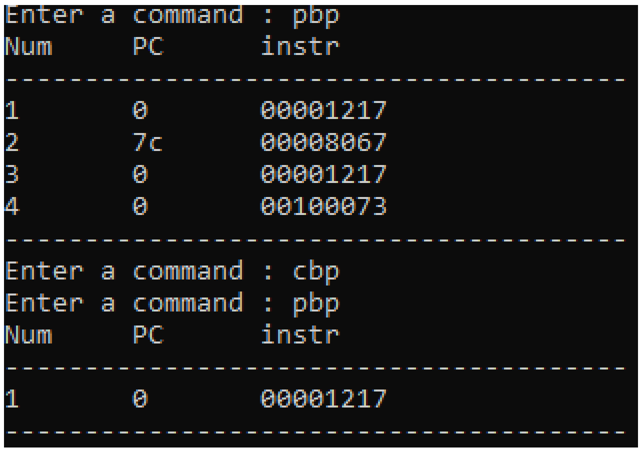
If a given good command is entered, then it performs the expected task. As of now, the cpp code is very messy and will need some more time to clean up, however, to reduce the complexity I originally started by creating individual functions for all the different commands, however after some headaches with passing values around between functions I opted to instead include them all in the main program. This is something I would like to spend some more time on cleaning up.



```
Enter a command : pbp
Num     PC      instr
---------------------------------------
1       0       00001217
---------------------------------------
```



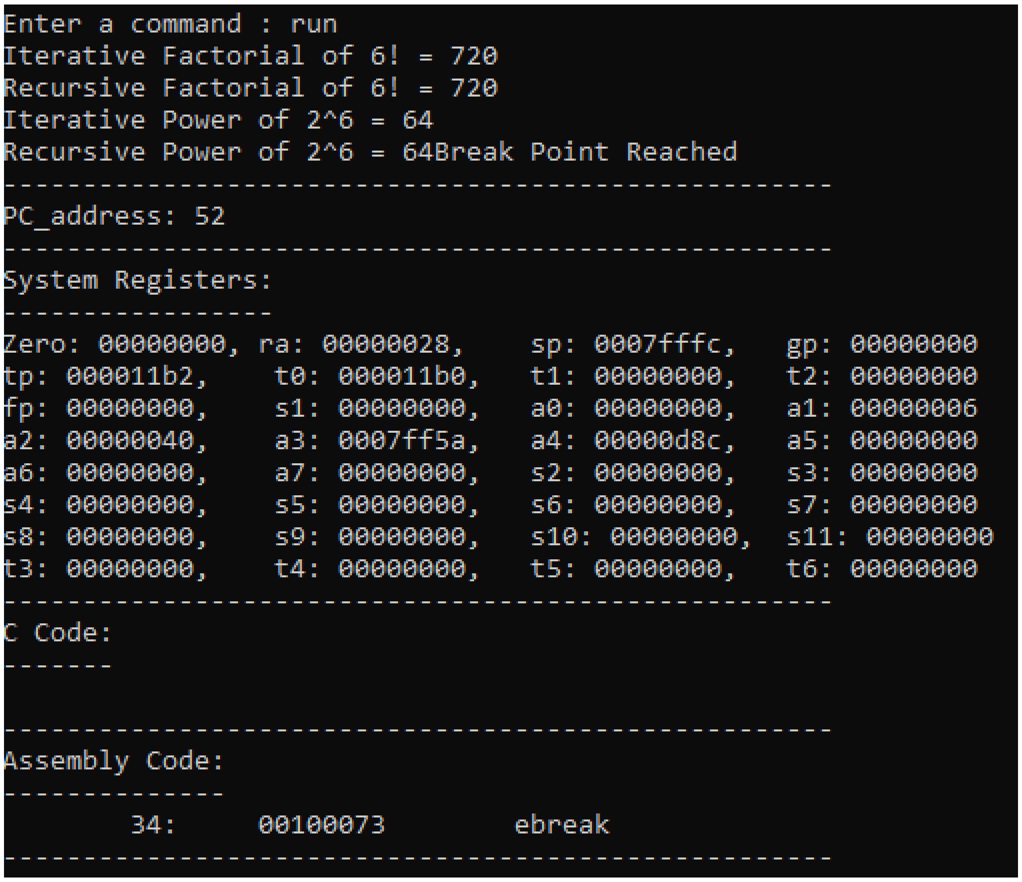
```
Enter a command : sbp 124
Enter a command : pbp
Num     PC      instr
---------------------------------------
1       0       00001217
2       7c      00008067
---------------------------------------
```



```
Enter a command : pbp
Num     PC      instr
---------------------------------------
1       0       00001217
2       7c      00008067
3       0       00001217
4       0       00100073
---------------------------------------
Enter a command : cbp
Enter a command : pbp
Num     PC      instr
---------------------------------------
1       0       00001217
---------------------------------------
```

When Clear Break Point is called you can see that I leave the first break point at the beginning. This is a bug that I have been attempting to solve, but clearly haven't been able to work out yet.

The next command shown below is the run command, here you can see it running until it hits an EBrake in the mainMemory array, this example is the end of the program, seen by the pc address 34 which when compared to our dissas file, it indicates it is the very last instruction of the program. Also included in the screenshot is the HW05.cpp file output printed before it hits the last break point showing both mine and the given function executing.



```
Enter a command : run
Iterative Factorial of 6! = 720
Recursive Factorial of 6! = 720
Iterative Power of 2^6 = 64
Recursive Power of 2^6 = 64Break Point Reached
---------------------------------------------------
PC_address: 52
---------------------------------------------------
System Registers:
-----------------
Zero: 00000000, ra: 00000028,    sp: 0007fffc,   gp: 00000000
tp: 000011b2,    t0: 000011b0,   t1: 00000000,   t2: 00000000
fp: 00000000,    s1: 00000000,   a0: 00000000,   a1: 00000006
a2: 00000040,    a3: 0007ff5a,   a4: 00000d8c,   a5: 00000000
a6: 00000000,    a7: 00000000,   s2: 00000000,   s3: 00000000
s4: 00000000,    s5: 00000000,   s6: 00000000,   s7: 00000000
s8: 00000000,    s9: 00000000,   s10: 00000000,  s11: 00000000
t3: 00000000,    t4: 00000000,   t5: 00000000,   t6: 00000000
---------------------------------------------------
C Code:
-------


---------------------------------------------------
Assembly Code:
--------------
        34:     00100073        ebreak
---------------------------------------------------
```

The problems that still exist:

Obviously, this isn't fully completed yet. No C code is outputted when break points are hit, along with only 1 line of assembly code being printed each time. In order to implement this feature, I would need to open the decodedline file that I generated with the readelf function provided and search through to find the address, then using the file and line number listed open a different cpp file and read and print that line of code. For multiple disassembled lines I would then read the next line of the decodedline file and compare the difference in addresses. For every 0x4 difference it is 1 line of assembly code, which then can be pulled from the mainMemory array and passed through my disassembly code from project 1.

I also found some bugs when testing my code about setting break points and ensuring that it was setting for the next C line instead of the next assembly instruction, which has led to some issues with properly setting the break points to be hit later. All these are logical errors in the code that will take some time to sort through, and with the short deadline I was unable to fix them. I hope with a little more time I might be able to sort through these problems.