University of Colorado Colorado Springs

CS 4200, Spring 2022: Computer Architecture

Instructor: Mong Sim

Project Two

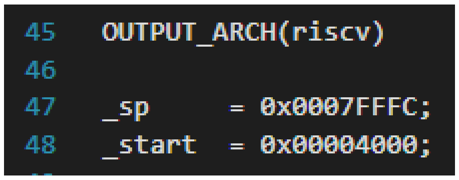
Andrew Schmidt

2022/03/15

Through this project there have been many issues that have come up. Through working with Dr. Mong Sim and lots of troubleshooting errors, I have come up with a working bootloader that compiles the original bin file into an header file starting at 0x4000, then reads the array or binary information, stores it into main memory at 0x4000, then runs the bootloader, finally jumping to the dhrystone program. Bellow I have included screenshots, explanations, and the problems and solutions I have found along the way.

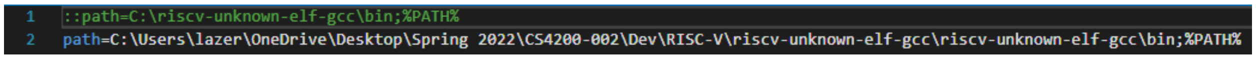
Recompile Dhrystone starting at 16KiB.

This was relatively straight forward. Open the rv32im.ls file and look for _start = 0x00000000; To set the start to 16KiB, 0x4000, we update this value to be _start = 0x00004000; and run the following commands to recompile Dhrystone (Done in HW2). d.bat, pp.bat, c.bat. The only changes are that pp.bat still points the riscv-unknown-elf-gcc bin folder as the path for compiling the code.



```
OUTPUT_ARCH(riscv)

_sp      = 0x0007FFFC;
_start   = 0x00004000;
```

Figure 1 rv32im.ls

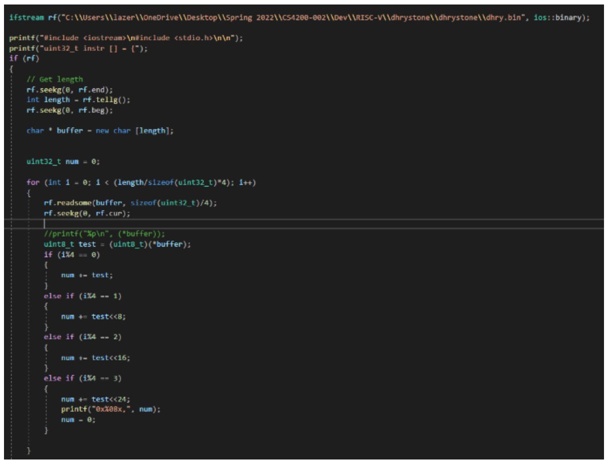


```
::path=C:\riscv-unknown-elf-gcc\bin;%PATH%
path=C:\Users\lazer\OneDrive\Desktop\Spring 2022\CS4200-002\Dev\RISC-V\riscv-unknown-elf-gcc\riscv-unknown-elf-gcc\bin;%PATH%
```

Figure 2 pp.bat

Write an application to convert the dhry.bin into dhry_bin.h

This step is still straight forward; however, it took a lot of research about how Cpp reads from files, specifically trying to read in the binary information instead of strings and other objects like I'm used to. After hours of tinkering I came up with the code I have which outputs the code to the terminal with normal printf and std::cout statements, then by compiling and running the executable pointing to an output file, I was able to capture the terminal output in the form of a file which I called dhry_bin.h. (.\binToHeader > dhry_bin.h). This program takes information 2 bytes at a time, shifts it left as needed to compile with the endianness that I wanted, then after it reads 8 bytes in it prints instruction to the terminal. There's file protection incase I can't open the original bin file correctly and dynamic length checking so if the bin file length was to be changed, the program would still work appropriately.

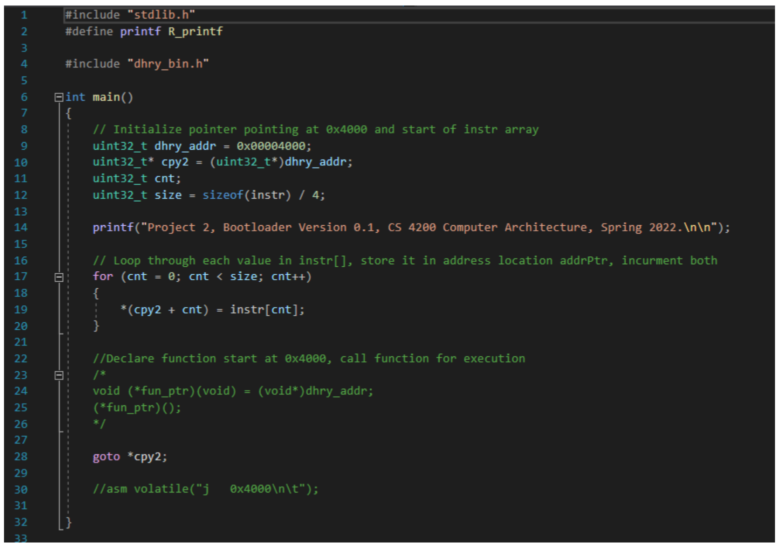


```
ifstream rf("C:\\Users\\lazer\\OneDrive\\Desktop\\Spring 2022\\CS4200-002\\Dev\\RISC-V\\dhrystone\\dhrystone\\dhry.bin", ios::binary);

printf("#include <iostream>\n#include <stdio.h>\n\n");
printf("uint32_t instr [] = {");
if (rf)
{
    // Get length
    rf.seekg(0, rf.end);
    int length = rf.tellg();
    rf.seekg(0, rf.beg);

    char * buffer = new char [length];


    uint32_t num = 0;

    for (int i = 0; i < (length/sizeof(uint32_t)*4); i++)
    {
        rf.readsome(buffer, sizeof(uint32_t)/4);
        rf.seekg(0, rf.cur);

        //printf("%p\n", (*buffer));
        uint8_t test = (uint8_t)(*buffer);
        if (i%4 == 0)
        {
            num += test;
        }
        else if (i%4 == 1)
        {
            num += test<<8;
        }
        else if (i%4 == 2)
        {
            num += test<<16;
        }
        else if (i%4 == 3)
        {
            num += test<<24;
            printf("0x%08x,", num);
            num = 0;
        }

    }
```

Figure 3 binToHeader.cpp

Write a bootloader to include dhry_bin.h

The bootloader is relatively simple once the main components are understood and the concept behind them become clear. Here I have a reworked version of the code that was reviewed, and I got help writing by Dr. Sim to make it clear and concise to aid the readability and efficiency. I have several different functions commented in that can be interchanged for different portions of the code, however using the goto method was the simplest and most clear way to show the program jumping to another memory location and continuing the execution there. When using goto, it's important to be carful with jumping to the correct location in order to get the right result, which is why we use the variable cpy2 which is set to the starting location and not changed.



```c
#include "stdlib.h"
#define printf R_printf

#include "dhry_bin.h"

int main()
{
    // Initialize pointer pointing at 0x4000 and start of instr array
    uint32_t dhry_addr = 0x00004000;
    uint32_t* cpy2 = (uint32_t*)dhry_addr;
    uint32_t cnt;
    uint32_t size = sizeof(instr) / 4;

    printf("Project 2, Bootloader Version 0.1, CS 4200 Computer Architecture, Spring 2022.\n\n");

    // Loop through each value in instr[], store it in address location addrPtr, incurment both
    for (cnt = 0; cnt < size; cnt++)
    {
        *(cpy2 + cnt) = instr[cnt];
    }

    //Declare function start at 0x4000, call function for execution
    /*
    void (*fun_ptr)(void) = (void*)dhry_addr;
    (*fun_ptr)();
    */

    goto *cpy2;

    //asm volatile("j   0x4000\n\t");

}
```

Figure 4 main.c

Write a linker file for your bootloader with the start address at 0x0

At the beginning I was confused on what the need for a linker file was, but having it set to 0x4000 for the dhrystone program makes it compile starting at 0x4000 so jump addresses are correct when converted to binary, then in the bootloader keeping the start address to 0x0000 so that the main.c will begin at the very beginning of memory before jumping to the dhrystone code that was put at location 0x4000 by the main.c program.

Finally I set the debugging path to the new bin file, and ran it in the softcore program to get the dhrystone program to run with my print statements first.

In the submission

1. main.c | Converts dhry_bin.h to memory address 0x4000
2. binToHeader.cpp | Output redirection used to store to header file
3. dhry_bin.h | Header file generated by binToHeader.cpp
4. Project02_Andrew_Schmidt.pdf
5. All other files needed in a bootloader folder