COMPUTER ORGANIZATION & ARCHITECTURE

B17 SOFTWARE DEVELOPERS MANUAL

May 1, 2015

Dylan Geyer, Grant Hill, Johnathan Ackerman
South Dakota School of Mines & Technology
Department of Mathematics and Computer Science

Contents

Chapter 1

Introduction

This document provides information about the B17 CPU emulated in software. This document explains how the system works and goes into depth on each function. Each function is documented and attributed to the author.

Chapter 2

System Overview

The software system is made up of 5 important pieces.

- Read Instruction Into Memory
- Read-Eval-Print Loop
- Instruction Switch
- Instruction Trace
- Execute Instruction

These subsystems will be discussed in this chapter and each function in each subsystem will be documented.

2.1 Read Object File Into Memory

Inside of *main*() a while loop is responsible for reading the input file into our virtual memory array.

```
while(obj >> hex >> nextAddress)
{
    if(obj >> hex >> toRead)
    {
        for(int i = 0; i < toRead; i++)
        {
            obj >> hex >> nextVal;
            memory[nextAddress + i] = nextVal;
        }
    }
}
```

2.2 Read-Eval-Print Loop

Also inside of *main*() is the infinite while loop that is responsible for reading, evaluating, and printing each instruction as it happens. The loop will run infinitely until a halting condition is met.

2.2.1 Read

```
while(true)
{ //Will end when program halts.
    ABUS = addmodemask & memory[IC];
    ABUS = ABUS >> 2;
    //Extract opcodes and addresses
    //Get the instruction
    IR = opmask & memory[IC];
    IR = IR >> 6; //Shift right after the mask is applied
    //And the proper memory address
    MAR = addrmask & memory[IC];
    MAR = MAR >> 12;
```

```
    MDR = memory[MAR];
    ...
}
```

This section of code will loop forever until and illegal or unimplemented addressing mode is encountered or a valid program halt is found.This part show above is the read section of of the Read-Eval-Print loop. It extracts all of the information about an instruction from memory. The next section will describe the *Evaluation* part of the loop.

2.2.2 Evaluation

```
switch( IR )
{
    case HALT:
    case NOP:
    ...
}
```

Once the information about an instruction has been extracted from the memory in the first part of the loop, the IR (Instruction Register) is used in a large switch statement to determine how the instruction is executed. Each instruction is a case within the switch statement and has a unique set of operations.

2.2.3 Print

The *trace(string mnemonic)* function handles the console output of the state of the machine to the user. It prints the memory address it is executing at, the mnemonic for the instruction, the addressing mode, and the values saved in the working registers X0, X1, X2, X3 as well as

the Accumulator Register AC.

Chapter 3

Function Overview

This chapter will cover each function in the program and what the responsibilities are of that function. This information can be found in the source file inside of each functions comment header.

3.1 main

Main has been made non-modular on purpose to more closely mimic the hardware of our CPU. This main function handles all of the actual operation of the CPU, it reads the .obj files into memory, and then begins execution until a halt is found. This utilizes our global variable registers in the same way the CPU would in hardware to perform each instruction so the state of our machine is always easily available to the other functions which will print the state to the user.

Authors

- Grant Hill - 40%
- Dylan Geyer - 30%
- Johnathan Ackerman - 30%

3.2 trace

Trace prints human readable data to the console output. Trace prints out the address the current instruction is at in memory, then the instruction code in HEX. It then prints out

the mnemonic for the instruction so humans can easily read it, then a mnemonic for the addressing mode. If there are no errors in these fields then the 24-bit AC, X0, X1, X2, X3 registers are displayed (in HEX).

example.

```
0c4: 050404 LD IMM AC[000050] X0[000] X1[000] X2[000] X3[000]
```

Authors

- Grant Hill - 50%
- Dylan Geyer - 20%
- Johnathan Ackerman - 30%

3.3 UnimplementedAddressing_Mode

This function is called whenever an instruction is attempting to use an unimplemented addressing mode. It makes sure the proper message is printed to the user, and the program is exited permanently with exit code: 3.

Authors

- Grant Hill - 34%
- Dylan Geyer - 33%
- Johnathan Ackerman - 33%

3.4 IllegalAddressing_Mode

This function is called whenever an instruction is attempting to use an Illegal addressing mode. It makes sure the proper message is printed to the user, and the program is exited permanently with exit code: 4.

Authors

- Grant Hill - 34%
- Dylan Geyer - 33%

- Johnathan Ackerman - 33%

Chapter 4

Program Testing

Testing was done on the program only to the extent that we verified each of the sample .obj files produced the desired outputs. These input/output combinations are shown below.

4.1 Object File 1

4.1.1 Input

```
50 1 000000
c4 5 050404 200800 300800 102840 050c00
101 2 300 9
200 1 30
300 1 10
c4
```

4.1.2 Output

```
0c4:  050404 LD   IMM  AC[000050]  X0[000]  X1[000]  X2[000]  X3[000]
0c5:  200800 ADD  200  AC[000080]  X0[000]  X1[000]  X2[000]  X3[000]
0c6:  300800 ADD  300  AC[000090]  X0[000]  X1[000]  X2[000]  X3[000]
0c7:  102840 SUB  102  AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
0c8:  050c00 J    050  AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
050:  000000 HALT      AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
Machine Halted - HALT instruction executed
```

4.2 Object File 2

4.2.1 Input

```
50 1 000000
100 5 200400 201840 202800 005844 050c84
200 3 30 31 10
100
```

4.2.2 Output

```
100:  200400 LD   200  AC[000030]  X0[000]  X1[000]  X2[000]  X3[000]
101:  201840 SUB  201  AC[ffffff]  X0[000]  X1[000]  X2[000]  X3[000]
102:  202800 ADD  202  AC[00000f]  X0[000]  X1[000]  X2[000]  X3[000]
103:  005844 SUB  IMM  AC[00000a]  X0[000]  X1[000]  X2[000]  X3[000]
104:  050c84 JN   ???  AC[00000a]  X0[000]  X1[000]  X2[000]  X3[000]
Machine Halted - illegal addressing mode
```

4.3 Object File 3

4.3.1 Input

```
50 1 000000
ff 7 075400 040804 077440 005884 076400 077800 030a04
75 3 30 20 10
ff
```

4.3.2 Output

```
0ff:  075400 LD   075  AC[000030]  X0[000]  X1[000]  X2[000]  X3[000]
100:  040804 ADD  IMM  AC[000070]  X0[000]  X1[000]  X2[000]  X3[000]
101:  077440 ST   077  AC[000070]  X0[000]  X1[000]  X2[000]  X3[000]
102:  005884 CLR  IMM  AC[000000]  X0[000]  X1[000]  X2[000]  X3[000]
103:  076400 LD   076  AC[000020]  X0[000]  X1[000]  X2[000]  X3[000]
104:  077800 ADD  077  AC[000030]  X0[000]  X1[000]  X2[000]  X3[000]
105:  030a04 ADDX IMM  AC[000030]  X0[000]  X1[000]  X2[000]  X3[000]
Machine Halted - unimplemented opcode
```

Chapter 5

Program Usage

This chapter tells the user very specifically how to build and run this project.

5.1 Build

Two files are required to build the project (B17.cpp & Makefile). Building the project is as simple as typing...

```
>> make
```

5.2 Run

After you have built the program using the provided makefile, g++ has created an executable file named B17 with executable permissions. Running this project is as simple as typing the program name and the name of a .obj input file.

```
>> ./B17 yourInputFile.obj
```