x86 Process Simulator for Mobile Phones

Robert Green

Advisor: Michael Black
Spring 2010
University Honors

Robert Green

Honors Capstone
Advisor: Michael Black

Why?

Over the past several years, the demands placed on cellular phones have increased greatly. Products such as Google's Android and Apple's iPhone feature a variety of applications ranging from useful references to games to computer applications which have been adapted for use on the phone. With the availability of so many capabilities, people's expectations have shifted from phones being used as communications devices to miniature computers. This is not an entirely unreasonable expectation, since smartphones currently have the same storage and processing capacity as a top of the line computer from the late '90s.

With that being said, what makes my project any different from the thousands of applications currently available? The answer requires a rudimentary knowledge of programming for mobile phones. Companies usually base programs for their products on a single language. Google uses Java to develop Android applications. In addition to the standard Java package, applications for Android utilize special packages which handle text and graphics displays, screen and keyboard input and other Android-specific functions. The purpose of my project is to create a process simulator which reads, interprets, and runs an executable file on the Google Android. My program translates an executable file into the appropriate Java code which will run on an Android. This will allow Android users without background knowledge of Android packages to write their own applications in a variety of languages and run those applications on their Android. This will bring mobile phones one step closer to functioning as personal computers.

What?

My first task was to gather the knowledge necessary to complete all parts of this project. I learned how to write simple programs in assembly language using the x86 instruction set. I also obtained an instruction set reference which allowed me to translate machine code into x86 instructions. My next task was to familiarize myself with the various headers associated with ELF files so that my program could interpret their information. Finally, I learned which android packages handle simple input and output functions.

Before customizing my project for the Android, I wrote a process simulator in Linux to run on my laptop. I started by writing a function to interpret the header information at the top of the executable file, and another to determine the size of the file. My program then reads the entire file into a character array based on the file size. I created a virtual memory for my simulator using the Hashtable class in Java. The executable file is broken down by sections and stored into virtual memory using the information gathered from the header. My program reads instructions from virtual memory byte by byte and interprets them using a large "if" statement. This is where the bulk of the programming took place, as it required interpreting each bit of every instruction. The instructions are executed by making the appropriate changes to virtual memory and the simulated registers. When the 'exit' instruction is reached, the final contents of the registers are printed.

My final task was to customize my simulator for the Android. This required rewriting my input and output operations using Android-specific commands. I loaded my program into the Android SDK using Eclipse, and ran my final program on an Android emulator.

Results

My process simulator currently has the ability to run an ELF executable file with a limited subset of x86 instructions. So far, I have programmed my simulator to recognize approximately 20 instructions. My program runs on a computer and an Android emulator. However, there is much work still to be done for this project to be complete. There are hundreds of instructions in the x86 instruction set left to program. The only other remaining task is to transfer my program to an actual Android device and ensure it runs with no errors.

Below I have included the Linux version of my code, as well as the version which has been modified to run on the Android.

Linux version

```
import java.util.*;
import java.io.*;

public class elfreader2
{
	public static void main(String[] args)
	{
		//Read the name of file from standard input into String s
		BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
		String s = "";
		System.out.print("Enter the name of an executable file: ");
		try
		{
			s = in.readLine();
		}
		catch(IOException e)
		{
			System.out.println("Error reading filename");
		}
		char somechar;
		// Find the size of the file, getsize returns 0 if an invalid name was given
		int filesize=getsize(s);
		//'file' is a character array which will store the entire elf file
		//'file' will remain unchanged throughout the program
		char[] file=new char[filesize];
		//'ram' is our virtual memory where we store certain sections of 'file'
		//'ram' will be changed by reads and writes throughout the program
		Hashtable ram=new Hashtable();
		//'elf' is an integer array which stores the values of the elf header
		int[] elf = new int[13];

		System.out.println("filesize = "+filesize);
		//If we have a valid file
		if (filesize>0)
		{
			try
			{
				//copy all data from our source file into 'file'
				FileInputStream fis = new FileInputStream(s);
				DataInputStream dis = new DataInputStream(fis);
				for (int i=0; i<filesize; i++)
				{
					somechar=(char)dis.readUnsignedByte();
					file[i]=somechar;
```

```
                        }
                    }
                    catch(IOException e)
                    {
                        System.out.println("Could not write 'file'");
                    }

//          for(int i=0; i<filesize; i++)
//          {
//          System.out.printf("%02x ", (int)file[i]);
//          }

                    //Populate 'elf' with values of elf header information
                    elf[0]=decconvert(file,16,2);
                    elf[1]=decconvert(file,18,2);
                    elf[2]=decconvert(file,20,4);
                    elf[3]=decconvert(file,24,4);
                    elf[4]=decconvert(file,28,4);
                    elf[5]=decconvert(file,32,4);
                    elf[6]=decconvert(file,36,4);
                    elf[7]=decconvert(file,40,2);
                    elf[8]=decconvert(file,42,2);
                    elf[9]=decconvert(file,44,2);
                    elf[10]=decconvert(file,46,2);
                    elf[11]=decconvert(file,48,2);
                    elf[12]=decconvert(file,50,2);

                    //For each section header, copy contents of its section from 'file' to 'ram'
                    for (int i=0; i<elf[11]; i++)
                    {
                        int base= elf[5]+elf[10]*i;
                        int addr= decconvert(file,base+12,4);
                        int off= decconvert(file,base+16,4);
                        int size= decconvert(file,base+20,4);

                        //don't look at sections starting at zero
                        //there can be one or more of these, and we don't care about them
                        if(addr!=0)
                        {
                            for (int j=0; j<size; j++)
                                writemem(ram,addr+j,file[off+j],1);
                        }
                    }

                    //prints ELF header, program headers and section headers
                    printfileinfo(elf,file);
```

```
                System.out.println(runprog(ram,elf[3]));
            }
            //If we have an invalid file name, the program automatically ends
            else
                System.out.println("Invalid file name... sorry");
        }

        public static int runprog(Hashtable ram, int entry)
        {
            //array for our 8 registers, and another for their names
            int[] reg = new int[8];
            String[] regname = { "eax", "ecx", "edx", "ebx", "esp", "ebp", "epi", "edi" };
            //flag to store results of a CMP command
            int cmpflag=0;
            //flag which stores sizes of info pushed to the stack (used when things are popped)
            int bflag=4;
            //default value for ebx, which was initially used as our return value
            reg[3]=0;
            //size of the stack, set to whatever number you need
            reg[4]=500;
            //'opcode' stores opcode of instruction we are currently handling
            //initially stores value determined from ELF header as the entry point
            int opcode=decconvert2(ram,entry,1);
            //pointer which determines where the next instruction lies (changed at the end of
each instruction)
            int point=entry;

            //each loop handles one instruction
            while(true)
            {
                if(opcode==0x89) //MOV r/m32,r32
                {
                    int modrm=decconvert2(ram,point+1,1);
                    int source=(modrm&63)>>3;
                        int mod=modrm>>6;
                    if(mod==3)
                    {
                        int dest=modrm&7;
                        System.out.println("mov "+regname[source]+" to
"+regname[dest]);
                        point=point+2;
                        reg[dest]=reg[source];
                    }
                    else
                    {
                        int[] dest = readmodrm(ram,point+1,reg);
```

```
                                        System.out.println("mov "+regname[source]+" to
mem["+dest[0]+"]");
                                        point=point+dest[1];
                                        writemem(ram,dest[0],reg[source],4);
                                }
                        }
                        else if(opcode==0x8b) // MOV r32,r/m32
                        {
                                int modrm=decconvert2(ram,point+1,1);
                                int dest=(modrm&63)>>3;
                                int mod=modrm>>6;
                                if(mod==3)
                                {
                                        int source=modrm&7;
                                        System.out.println("mov "+regname[source]+" to
"+regname[dest]);
                                        point=point+2;
                                        reg[dest]=reg[source];
                                }
                                else
                                {
                                        int[] source = readmodrm(ram,point+1,reg);
                                        System.out.println("mov mem["+source[0]+"] to
"+regname[dest]);
                                        point=point+source[1];
                                        reg[dest]=decconvert2(ram,source[0],4);
                                }
                        }
                        else if(opcode<0xc0&&opcode>0xb7) //MOV r32,imm32
                        {
                                int dest = opcode&7;
                                int source = decconvert2(ram,point+1,4);
                                System.out.println("mov $"+source+" to "+regname[dest]);
                                reg[dest]=source;
                                point=point+5;
                        }
                        else if(opcode==0x39) //CMP r/m32,r32
                        {
                                int modrm=decconvert2(ram,point+1,1);
                                int op2=(modrm&63)>>3;
                                int mod=modrm>>6;
                                if(mod==3)
                                {
                                        int op1=modrm&7;
                                        point=point+2;
                                        if (reg[op1]-reg[op2]<0){
```

```
                                        cmpflag=2;
                                        System.out.println("cmp "+regname[op1]+" <
"+regname[op2]);}
                                else if (reg[op1]-reg[op2]>0){
                                        cmpflag=3;
                                        System.out.println("cmp "+regname[op1]+" >
"+regname[op2]);}
                                else{
                                        cmpflag=1;
                                        System.out.println("cmp "+regname[op1]+" =
"+regname[op2]);}
                        }
                        else
                        {
                                int[] op1 = readmodrm(ram,point+1,reg);
                                point=point+op1[1];
                                int cr=decconvert2(ram,op1[0],4)-reg[op2];
                                if (cr<0){
                                        cmpflag=2;
                                        System.out.println("cmp mem["+op1[0]+"] <
"+regname[op2]);}
                                else if (cr>0){
                                        cmpflag=3;
                                        System.out.println("cmp mem["+op1[0]+"] >
"+regname[op2]);}
                                else{
                                        cmpflag=1;
                                        System.out.println("cmp mem["+op1[0]+"] =
"+regname[op2]);}
                        }
                }
                else if(opcode==0x3B) //CMP r32,r/m32
                {
                        int modrm=decconvert2(ram,point+1,1);
                        int op1=(modrm&63)>>3;
                        int mod=modrm>>6;
                        if(mod==3)
                        {
                                int op2=modrm&7;
                                point=point+2;
                                if (reg[op1]-reg[op2]<0){
                                        cmpflag=2;
                                        System.out.println("cmp "+regname[op1]+" <
"+regname[op2]);}
                                else if (reg[op1]-reg[op2]>0){
                                        cmpflag=3;
```

```
                                        System.out.println("cmp "+regname[op1]+" >
"+regname[op2]);}
                                else{
                                        cmpflag=1;
                                        System.out.println("cmp "+regname[op1]+" =
"+regname[op2]);}
                        }
                        else
                        {
                                int[] op2 = readmodrm(ram,point+1,reg);
                                point=point+op2[1];
                                int cr=reg[op1]-decconvert2(ram,op2[0],4);
                                if (cr<0){
                                        cmpflag=2;
                                        System.out.println("cmp "+regname[op1]+" <
mem["+op2[0]+"]");}
                                else if (cr>0){
                                        cmpflag=3;
                                        System.out.println("cmp "+regname[op1]+" >
mem["+op2[0]+"]");}
                                else{
                                        cmpflag=1;
                                        System.out.println("cmp "+regname[op1]+" =
mem["+op2[0]+"]");}
                        }
                }
                else if(opcode==0x83)
                {
                        int modrm=decconvert2(ram,point+1,1);
                        int mod=modrm>>6;
                        int ro=(modrm>>3)&7;
                        int rm=modrm&7;
                        if (ro==7) //CMP r/m,imm8
                        {
                                if (mod==3)
                                {
                                        int op1=rm;
                                        int op2=decconvert2(ram,point+2,1);
                                        if (reg[op1]-op2<0){
                                                cmpflag=2;
                                                System.out.println("cmp "+regname[op1]+"
< "+op2);}
                                        else if (reg[op1]-op2>0){
                                                cmpflag=3;
                                                System.out.println("cmp
"+regname[op1]+" > "+op2);}
```

```
                                        else{
                                                cmpflag=1;
                                                System.out.println("cmp "+regname[op1]+"
= "+op2);}
                                        point=point+3;
                                }
                                else
                                {
                                        int[] op1=readmodrm(ram,point+1,reg);
                                        int op2=decconvert2(ram,point+op1[1],1);
                                        point=point+op1[1]+1;
                                        if (decconvert2(ram,op1[0],4)-op2<0){
                                                cmpflag=2;
                                                System.out.println("cmp mem["+op1[0]+"]
< "+op2);}
                                        else if (decconvert2(ram,op1[0],4)-op2>0){
                                                cmpflag=3;
                                                System.out.println("cmp
mem["+op1[0]+"] > "+op2);}
                                        else{
                                                cmpflag=1;
                                                System.out.println("cmp mem["+op1[0]+"]
= "+op2);}
                                }
                        }
                        else if (ro==0) // ADD r/m32,imm8
                        {
                                if(mod==3)
                                {
                                        int op1=rm;
                                        int op2=decconvert2(ram,point+2,1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("add "+op2+" to
"+regname[op1]);
                                        point=point+3;
                                        reg[op1]=reg[op1]+op2;
                                }
                                else
                                {
                                        int[] op1 = readmodrm(ram,point+1,reg);
                                        int op2=decconvert2(ram,point+op1[1],1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("add "+op2+" to
mem["+op1[0]+"]");
```

```
                                        point=point+op1[1]+1;
                                        int result=decconvert2(ram,op1[0],4)+op2;
                                        writemem(ram,op1[0],result,4);
                                }
                        }
                        else if (ro==5) // SUB r/m32,imm8
                        {
                                if(mod==3)
                                {
                                        int op1=rm;
                                        int op2=decconvert2(ram,point+2,1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("sub "+op2+" from
"+regname[op1]);
                                        point=point+3;
                                        reg[op1]=reg[op1]-op2;
                                }
                                else
                                {
                                        int[] op1 = readmodrm(ram,point+1,reg);
                                        int op2=decconvert2(ram,point+op1[1],1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("sub "+op2+" from
mem["+op1[0]+"]");
                                        point=point+op1[1]+1;
                                        int result=decconvert2(ram,op1[0],4)-op2;
                                        writemem(ram,op1[0],result,4);
                                }
                        }
                        else
                                System.out.println("Something went wrong with op 83");
                }
                else if(opcode==0xeb)//JMP
                {
                        int jmp=decconvert2(ram,point+1,1);
                        if ((jmp>>7)==1)
                                jmp=(jmp&127)-128;
                        point=point+jmp+2;
                        System.out.println("jmp");
                }
                else if(opcode<0x80&&opcode>0x6f)//JCC
                {
                        int jmp=decconvert2(ram,point+1,1);
                        if ((jmp>>7)==1)
```

```
                jmp=(jmp&127)-128;
        point=point+2;
        if (opcode==0x74)
        {
                System.out.println("jmp if e");
                if (cmpflag==1)
                        point=point+jmp;
        }
        else if (opcode==0x75)
        {
                System.out.println("jmp if ne");
                if (cmpflag==2||cmpflag==3)
                        point=point+jmp;
        }
        else if (opcode==0x7c)
        {
                System.out.println("jmp if l");
                if (cmpflag==2)
                        point=point+jmp;
        }
        else if (opcode==0x7d)
        {
                System.out.println("jmp if ge");
                if (cmpflag==1||cmpflag==3)
                        point=point+jmp;
        }
        else if (opcode==0x7e)
        {
                System.out.println("jmp if le");
                if (cmpflag==1||cmpflag==2)
                        point=point+jmp;
        }
        else if (opcode==0x7f)
        {
                System.out.println("jmp if g");
                if (cmpflag==3)
                        point=point+jmp;
        }
        else
        {
                System.out.println("jcc error!");
        }
        cmpflag=0;
}
else if(opcode<0x48&&opcode>0x3f)//INC
{
```

```
                int r=opcode&7;
                System.out.println("Increment "+regname[r]);
                reg[r]=reg[r]+1;
                point=point+1;
            }
            else if(opcode<0x50&&opcode>47)//DEC
            {
                int r=opcode&7;
                System.out.println("Decrement "+regname[r]);
                reg[r]=reg[r]-1;
                point=point+1;
            }
            else if(opcode==0x01)//ADD r/m32,r32
            {
                int modrm=decconvert2(ram,point+1,1);
                int mod=modrm>>6;
                int op2=(modrm>>3)&7;
                int rm=modrm&7;
                if(mod==3)
                {
                    int op1=rm;
                    System.out.println("add "+regname[op2]+" to
"+regname[op1]);
                    point=point+2;
                    reg[op1]=reg[op1]+reg[op2];
                }
                else
                {
                    int[] op1 = readmodrm(ram,point+1,reg);
                    System.out.println("add "+regname[op2]+" to
mem["+op1[0]+"]");
                    point=point+op1[1];
                    int result=decconvert2(ram,op1[0],4)+reg[op2];
                    writemem(ram,op1[0],result,4);
                }
            }
            else if(opcode==0x03)//ADD r32,r/m32
            {
                int modrm=decconvert2(ram,point+1,1);
                int mod=modrm>>6;
                int op1=(modrm>>3)&7;
                int rm=modrm&7;
                if(mod==3)
                {
                    int op2=rm;
```

```
                                        System.out.println("add "+regname[op2]+" to
"+regname[op1]);
                                        point=point+2;
                                        reg[op1]=reg[op1]+reg[op2];
                                }
                                else
                                {
                                        int[] op2 = readmodrm(ram,point+1,reg);
                                        System.out.println("add mem["+op2[0]+"] to
"+regname[op1]);
                                        point=point+op2[1];
                                        reg[op1]=decconvert2(ram,op2[0],4)+reg[op1];
                                }
                        }
                        else if(opcode==0x0F)//IMUL r32,r/m32
                        {
                                if (decconvert2(ram,point+1,1)==0xAF)
                                {
                                        int modrm=decconvert2(ram,point+2,1);
                                        int mod=modrm>>6;
                                        int op1=(modrm>>3)&7;
                                        int rm=modrm&7;
                                        if(mod==3)
                                        {
                                                int op2=rm;
                                                System.out.println("mul "+regname[op1]+" by
"+regname[op2]);
                                                point=point+3;
                                                reg[op1]=reg[op1]*reg[op2];
                                        }
                                        else
                                        {
                                                int[] op2 = readmodrm(ram,point+2,reg);
                                                System.out.println("mul "+regname[op1]+" by
mem["+op2[0]+"]");
                                                point=point+op2[1]+1;
                                                reg[op1]=decconvert2(ram,op2[0],4)*reg[op1];
                                        }
                                }
                                else
                                        System.out.println("IMUL Error");
                        }
                        else if(opcode<0x58&&opcode>0x4F)//PUSH r32
                        {
                                int source=opcode&7;
                                reg[4]=reg[4]-4;
```

```
            writemem(ram,reg[4],reg[source],4);
            point++;
            System.out.println("Pushed "+regname[source]+" to the stack");
            bflag=(bflag<<3)+4;
    }
    else if(opcode==0x6A)//PUSH imm8
    {
            int source = decconvert2(ram,point+1,1);
            reg[4]=reg[4]-4;
            writemem(ram,reg[4],source,4);
            point=point+2;
            System.out.println("Pushed "+source+" to the stack");
            bflag=(bflag<<3)+4;
    }
    else if(opcode==0x68)//PUSH imm32
    {
            int source = decconvert2(ram,point+1,4);
            reg[4]=reg[4]-4;
            writemem(ram,reg[4],source,4);
            point=point+5;
            System.out.println("Pushed "+source+" to the stack");
            bflag=(bflag<<3)+4;
    }
    else if(opcode<0x60&&opcode>0x57)//POP r32
    {
            int size=bflag&7;
            int dest=opcode&7;
            if (size==2)
                    reg[dest]=decconvert2(ram,reg[4],2);
            else if (size==4)
                    reg[dest]=decconvert2(ram,reg[4],4);
            else
                    System.out.println("Size error in Pop");
            reg[4]=reg[4]+size;
            bflag=bflag>>3;
            point++;
            System.out.println("Popped from the stack to "+regname[dest]);
    }
    else if(opcode==0xE8)//CALL
    {
            point=point+5;
            System.out.print("Return address = "+point+" ");
            reg[4]=reg[4]-4;
            writemem(ram,reg[4],point,4);
            System.out.println("Call function");
            bflag=(bflag<<3)+4;
```

```
                int jmp=decconvert2(ram,point-4,4);
                if ((jmp>>31)==1)
                        jmp=(jmp&2147483647)-2147483647-1;
                point=point+jmp;
        }
        else if(opcode==0xC3)//RET
        {
                bflag=bflag>>3;
                point=decconvert2(ram,reg[4],4);
                reg[4]=reg[4]+4;
                System.out.println("Returned from function at "+point);
        }
        else if(opcode==0xcd)//INT
        {
                if (decconvert2(ram,point+1,1)==0x80){
                        System.out.println("Interrupt 80");
                        if (reg[0]==1){
                                System.out.println("exit");
                                break;}}
                else{
                        System.out.println("some other syscall");
                        point=point+2;}
        }
        else
        {
                System.out.println("Something went wrong!");
                break;
        }
        opcode=decconvert2(ram,point,1);
    }

    return reg[3];
}

//interprets ModR/M byte based on info. in table on page 36 of Intel manual
public static int[] readmodrm(Hashtable ram, int loc, int[] reg)
{
    int[] result = new int[2];
    int modrm=decconvert2(ram,loc,1);
    int mod=modrm>>6;
    int rm=modrm&7;
    int ro=(modrm>>3)&7;
    if (mod==0)
    {
        if (rm==4)
        {
```

```
                int sib=decconvert2(ram,loc+1,1);
                int base=sib&7;
                if (base==5)
                {
                        result[0]=readsib(sib,mod,reg)+decconvert2(ram,loc+2,4);
                        result[1]=7;
                }
                else
                {
                        result[0]=readsib(sib,mod,reg);
                        result[1]=3;
                }
        }
        else if (rm==5)
        {
                result[0]=decconvert2(ram,loc+1,4);
                result[1]=6;
        }
        else
        {
                result[0]=reg[rm];
                result[1]=2;
        }
}
else if (mod==1)
{
        if (rm==4)
        {
                int sib=decconvert2(ram,loc+1,1);
                int disp=decconvert2(ram,loc+2,1);
                if ((disp>>7)==1)
                        disp=(disp&127)-128;
                else
                        disp=disp&127;
                result[0]=readsib(sib,mod,reg)+disp;
                result[1]=4;
        }
        else
        {
                int disp=decconvert2(ram,loc+1,1);
                if ((disp>>7)==1)
                        disp=(disp&127)-128;
                else
                        disp=disp&127;
                result[0]=reg[rm]+disp;
                result[1]=3;
```

```
                }
        }
        else if (mod==2)
        {
                if (rm==4)
                {
                        int sib=decconvert2(ram,loc+1,1);
                        result[0]=readsib(sib, mod, reg)+decconvert2(ram,loc+2,4);
                        result[1]=7;
                }
                else
                {
                        result[0]=reg[rm]+decconvert2(ram,loc+1,4);
                        result[1]=6;
                }
        }
        else
        {
                System.out.println("readmodrm error!");
        }
        return result;
}

//interprets SIB byte based on info. in table on page 37 of Intel manual
public static int readsib(int sib, int mod, int[] reg)
{
        int s=sib>>6;
        int scale=0;
        int index=(sib>>3)&7;
        int base=sib&7;
        if (s==0)
                scale=1;
        else if (s==1)
                scale=2;
        else if (s==2)
                scale=4;
        else if (s==3)
                scale=8;
        else
                System.out.println("readsib error!");
        int si = 0;

        if (index!=4)
                si=reg[index]*scale;

        if (base!=5)
```

```
                return reg[base]+si;
            else if (mod==0)
                return si;
            else
                return reg[5]+si;
        }

        //prints out all information in ELF header, program headers and section headers
        public static void printfileinfo(int[] elf, char[] file)
        {
            System.out.printf("\nELF HEADER\nType Code: %x\n", elf[0]);
            System.out.printf("Machine Code: %x\n", elf[1]);
            System.out.printf("Version: 0x%x\n", elf[2]);
            System.out.printf("Entry point address: 0x%x\n", elf[3]);
            System.out.println("Start of program headers: "+elf[4]+" (bytes into file)");
            System.out.println("Start of section headers: "+elf[5]+" (bytes into file)");
            System.out.printf("Flags: 0x%x\n", elf[6]);
            System.out.println("Size of ELF Header: "+elf[7]+" (bytes)");
            System.out.println("Size of program headers: "+elf[8]+" (bytes)");
            System.out.println("Number of program headers: "+elf[9]);
            System.out.println("Size of section headers: "+elf[10]+" (bytes)");
            System.out.println("Number of section headers: "+elf[11]);
            System.out.println("Section Header String Table Index: "+elf[12]);

            for (int i=0; i<elf[9]; i++)
            {
                int base= elf[4]+elf[8]*i;
                System.out.println("\n"+"Program Header "+(i+1)+" of "+elf[9]);
                System.out.printf("Type Code: %x\n", decconvert(file,base,4));
                System.out.printf("Offset: 0x%x\n", decconvert(file,base+4,4));
                System.out.printf("Virtual Memory Address: 0x%x\n",
decconvert(file,base+8,4));
                System.out.printf("Physical Address: 0x%x\n",
decconvert(file,base+12,4));
                System.out.printf("Bytes in file image: 0x%x\n",
decconvert(file,base+16,4));
                System.out.printf("Bytes in memory image: 0x%x\n",
decconvert(file,base+20,4));
                System.out.printf("Flags: 0x%x\n", decconvert(file,base+24,4));
                System.out.printf("Segment Alignment: 0x%x\n",
decconvert(file,base+28,4));
            }

            for (int i=0; i<elf[11]; i++)
            {
                int base= elf[5]+elf[10]*i;
```

```
                        System.out.println("\n"+"Section Header "+(i+1)+" of "+elf[11]);
                        System.out.println("Name Index: "+decconvert(file,base,4));
                        System.out.printf("Type Code: %x\n", decconvert(file,base+4,4));
                        System.out.printf("Flags: 0x%x\n", decconvert(file,base+8,4));
                        System.out.printf("Address: 0x%x\n", decconvert(file,base+12,4));
                        System.out.printf("Offset: 0x%x\n", decconvert(file,base+16,4));
                        System.out.printf("Size: 0x%x\n", decconvert(file,base+20,4));
                        System.out.printf("Link: 0x%x\n", decconvert(file,base+24,4));
                        System.out.printf("Info: 0x%x\n", decconvert(file,base+28,4));
                        System.out.printf("Section Alignment: 0x%x\n",
decconvert(file,base+32,4));
                        System.out.printf("Entry Size: 0x%x\n", decconvert(file,base+36,4));
                }
                System.out.println();
        }

        //returns integer value of 1,2 or 4 byte piece of information in 'file'
        public static int decconvert(char[] source, int loc, int size)
        {
                int result=0;
                for (int i=0; i<size; i++)
                {
                        result=result+(((int)source[loc+i])<<(8*i));
                }
                return result;
        }

        //returns integer value of 1,2 or 4 byte piece of information in 'ram'
        public static int decconvert2(Hashtable source, int loc, int size)
        {
                int result=0;
                for (int i=0; i<size; i++)
                {
                        Character info = (Character)source.get(""+(loc+i));
                        result=result+(info.hashCode()<<(8*i));
                }
                return result;
        }

        //method which controls writing to memory, can write 1,2 or 4 bytes at a time
        public static void writemem(Hashtable source, int loc, int info, int size)
        {
                if (size==1)
                {
                        String newstring = ""+loc;
                        source.put(newstring, new Character((char)info));
```

```
		}
		else if (size==2)
		{
			String newstring1 = ""+(loc+1);
			String newstring2 = ""+loc;
			int info1=(info>>8)&255;
			int info2=info&255;
			source.put(newstring1, new Character((char)info1));
			source.put(newstring2, new Character((char)info2));
		}
		else if (size==4)
		{
			String newstring1 = ""+(loc+3);
			String newstring2 = ""+(loc+2);
			String newstring3 = ""+(loc+1);
			String newstring4 = ""+loc;
			int info1=(info>>24)&255;
			int info2=(info>>16)&255;
			int info3=(info>>8)&255;
			int info4=info&255;
			source.put(newstring1, new Character((char)info1));
			source.put(newstring2, new Character((char)info2));
			source.put(newstring3, new Character((char)info3));
			source.put(newstring4, new Character((char)info4));
		}
		else
			System.out.println("Size error for writemem");

	}

	//if 'filename' is invalid, getsize returns 0, otherwise it returns the size of the file (in bytes)
	public static int getsize(String filename)
	{
		int sizecounter=0;
		try{
			FileInputStream fis = new FileInputStream(filename);
			DataInputStream dis = new DataInputStream(fis);
			while(true)
			{
				//readUnsignedByte will fail when we reach end of file, thus
diverting getsize to the catch
				dis.readUnsignedByte();
				sizecounter++;
			}
		}
		catch(IOException e)
```

```
                {
                        return sizecounter;
                }
        }
}
```

Android version

```
package com.example.capstone;

import java.io.*;
import java.util.*;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class capstone extends Activity {
   /** Called when the activity is first created. */
   @Override
         //Essentially, this method takes the place of your main class
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
         //I have only figured out how to print a single line to the screen, so I put everything in this
variable 's'
      String s=" ";
         //I have not yet figured out how to read input from android, so I just point to a file
      String filename="sdcard/factorial";
      int filesize=0;
      filesize=getsize(filename);
      char[] file =new char[filesize];
      Hashtable ram=new Hashtable();
      int[] elf = new int[13];
      char somechar;

      if(filesize>0){
            try{
            FileInputStream fis = new FileInputStream(filename);
            DataInputStream dis = new DataInputStream(fis);
            for (int i=0; i<filesize; i++){
                  somechar=(char)dis.readUnsignedByte();
                  file[i]=somechar;
            }
         }
         catch(IOException e)
         {
         s="Bad News: Failed to read the file";
         }

         elf[0]=decconvert(file,16,2);
         elf[1]=decconvert(file,18,2);
         elf[2]=decconvert(file,20,4);
         elf[3]=decconvert(file,24,4);
         elf[4]=decconvert(file,28,4);
```

```
        elf[5]=decconvert(file,32,4);
        elf[6]=decconvert(file,36,4);
        elf[7]=decconvert(file,40,2);
        elf[8]=decconvert(file,42,2);
        elf[9]=decconvert(file,44,2);
        elf[10]=decconvert(file,46,2);
        elf[11]=decconvert(file,48,2);
        elf[12]=decconvert(file,50,2);

        for (int i=0; i<elf[11]; i++)
        {
                int base= elf[5]+elf[10]*i;
                int addr= decconvert(file,base+12,4);
                int off= decconvert(file,base+16,4);
                int size= decconvert(file,base+20,4);

                        if(addr!=0)
                        {
                                for (int j=0; j<size; j++)
                                {
                                String newstring = ""+(addr+j);
                                ram.put(newstring,new Character(file[off+j]));
                                }
                        }
        }

        int[] registers=runprog(ram,elf[3]);
                //This is where I say what I am printing to the screen, in this case it is the values
of my registers
        s="EAX = "+registers[0]+", ECX = "+registers[1]+", EDX = "+registers[2]+", EBX =
"+registers[3]+", ESP = "+registers[4]+", EBP = "+registers[5]+", EPI = "+registers[6]+", EDI =
"+registers[7];
      }
      else{
         s="Invalid File name, filesize=0";
      }
                //This is where I am actually putting my String on the screen
           TextView tv=new TextView(this);
              tv.setText(s);
              setContentView(tv);
           }
//From here on, the program is exactly the same as the Linux version
        public static int[] runprog(Hashtable ram, int entry)
        {
                //array for our 8 registers, and another for their names
                int[] reg = new int[8];
```

```
		String[] regname = { "eax", "ecx", "edx", "ebx", "esp", "ebp", "epi", "edi" };
		//flag to store results of a CMP command
		int cmpflag=0;
		//flag which stores sizes of info pushed to the stack (used when things are popped)
		int bflag=4;
		//default value for ebx, which was initially used as our return value
		reg[3]=0;
		//size of the stack, set to whatever number you need
		reg[4]=500;
		//'opcode' stores opcode of instruction we are currently handling
		//initially stores value determined from ELF header as the entry point
		int opcode=decconvert2(ram,entry,1);
		//pointer which determines where the next instruction lies (changed at the end of
each instruction)
		int point=entry;

		//each loop handles one instruction
		while(true)
		{
			if(opcode==0x89) //MOV r/m32,r32
			{
				int modrm=decconvert2(ram,point+1,1);
				int source=(modrm&63)>>3;
					int mod=modrm>>6;
				if(mod==3)
				{
					int dest=modrm&7;
					System.out.println("mov "+regname[source]+" to
"+regname[dest]);
					point=point+2;
					reg[dest]=reg[source];
				}
				else
				{
					int[] dest = readmodrm(ram,point+1,reg);
					System.out.println("mov "+regname[source]+" to
mem["+dest[0]+"]");
					point=point+dest[1];
					writemem(ram,dest[0],reg[source],4);
				}
			}
			else if(opcode==0x8b) // MOV r32,r/m32
			{
				int modrm=decconvert2(ram,point+1,1);
				int dest=(modrm&63)>>3;
				int mod=modrm>>6;
```

```
                        if(mod==3)
                        {
                                int source=modrm&7;
                                System.out.println("mov "+regname[source]+" to
"+regname[dest]);
                                point=point+2;
                                reg[dest]=reg[source];
                        }
                        else
                        {
                                int[] source = readmodrm(ram,point+1,reg);
                                System.out.println("mov mem["+source[0]+"] to
"+regname[dest]);
                                point=point+source[1];
                                reg[dest]=decconvert2(ram,source[0],4);
                        }
                }
                else if(opcode<0xc0&&opcode>0xb7) //MOV r32,imm32
                {
                        int dest = opcode&7;
                        int source = decconvert2(ram,point+1,4);
                        System.out.println("mov $"+source+" to "+regname[dest]);
                        reg[dest]=source;
                        point=point+5;
                }
                else if(opcode==0x39) //CMP r/m32,r32
                {
                        int modrm=decconvert2(ram,point+1,1);
                        int op2=(modrm&63)>>3;
                        int mod=modrm>>6;
                        if(mod==3)
                        {
                                int op1=modrm&7;
                                point=point+2;
                                if (reg[op1]-reg[op2]<0){
                                        cmpflag=2;
                                        System.out.println("cmp "+regname[op1]+" <
"+regname[op2]);}
                                else if (reg[op1]-reg[op2]>0){
                                        cmpflag=3;
                                        System.out.println("cmp "+regname[op1]+" >
"+regname[op2]);}
                                else{
                                        cmpflag=1;
                                        System.out.println("cmp "+regname[op1]+" =
"+regname[op2]);}
```

```
                }
                else
                {
                        int[] op1 = readmodrm(ram,point+1,reg);
                        point=point+op1[1];
                        int cr=decconvert2(ram,op1[0],4)-reg[op2];
                        if (cr<0){
                                cmpflag=2;
                                System.out.println("cmp mem["+op1[0]+"] <
"+regname[op2]);}
                        else if (cr>0){
                                cmpflag=3;
                                System.out.println("cmp mem["+op1[0]+"] >
"+regname[op2]);}
                        else{
                                cmpflag=1;
                                System.out.println("cmp mem["+op1[0]+"] =
"+regname[op2]);}
                }
        }
        else if(opcode==0x3B) //CMP r32,r/m32
        {
                int modrm=decconvert2(ram,point+1,1);
                int op1=(modrm&63)>>3;
                int mod=modrm>>6;
                if(mod==3)
                {
                        int op2=modrm&7;
                        point=point+2;
                        if (reg[op1]-reg[op2]<0){
                                cmpflag=2;
                                System.out.println("cmp "+regname[op1]+" <
"+regname[op2]);}
                        else if (reg[op1]-reg[op2]>0){
                                cmpflag=3;
                                System.out.println("cmp "+regname[op1]+" >
"+regname[op2]);}
                        else{
                                cmpflag=1;
                                System.out.println("cmp "+regname[op1]+" =
"+regname[op2]);}
                }
                else
                {
                        int[] op2 = readmodrm(ram,point+1,reg);
                        point=point+op2[1];
```

```
                                        int cr=reg[op1]-decconvert2(ram,op2[0],4);
                                        if (cr<0){
                                                cmpflag=2;
                                                System.out.println("cmp "+regname[op1]+" <
mem["+op2[0]+"]");}
                                        else if (cr>0){
                                                cmpflag=3;
                                                System.out.println("cmp "+regname[op1]+" >
mem["+op2[0]+"]");}
                                        else{
                                                cmpflag=1;
                                                System.out.println("cmp "+regname[op1]+" =
mem["+op2[0]+"]");}
                                }
                        }
                        else if(opcode==0x83)
                        {
                                int modrm=decconvert2(ram,point+1,1);
                                int mod=modrm>>6;
                                int ro=(modrm>>3)&7;
                                int rm=modrm&7;
                                if (ro==7) //CMP r/m,imm8
                                {
                                        if (mod==3)
                                        {
                                                int op1=rm;
                                                int op2=decconvert2(ram,point+2,1);
                                                if (reg[op1]-op2<0){
                                                        cmpflag=2;
                                                        System.out.println("cmp "+regname[op1]+"
< "+op2);}
                                                else if (reg[op1]-op2>0){
                                                        cmpflag=3;
                                                        System.out.println("cmp
"+regname[op1]+" > "+op2);}
                                                else{
                                                        cmpflag=1;
                                                        System.out.println("cmp "+regname[op1]+"
= "+op2);}
                                                point=point+3;
                                        }
                                        else
                                        {
                                                int[] op1=readmodrm(ram,point+1,reg);
                                                int op2=decconvert2(ram,point+op1[1],1);
                                                point=point+op1[1]+1;
```

```
                                        if (decconvert2(ram,op1[0],4)-op2<0){
                                                cmpflag=2;
                                                System.out.println("cmp mem["+op1[0]+"]
< "+op2);}
                                        else if (decconvert2(ram,op1[0],4)-op2>0){
                                                cmpflag=3;
                                                System.out.println("cmp
mem["+op1[0]+"] > "+op2);}
                                        else{
                                                cmpflag=1;
                                                System.out.println("cmp mem["+op1[0]+"]
= "+op2);}
                                }
                        }
                        else if (ro==0) // ADD r/m32,imm8
                        {
                                if(mod==3)
                                {
                                        int op1=rm;
                                        int op2=decconvert2(ram,point+2,1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("add "+op2+" to
"+regname[op1]);
                                        point=point+3;
                                        reg[op1]=reg[op1]+op2;
                                }
                                else
                                {
                                        int[] op1 = readmodrm(ram,point+1,reg);
                                        int op2=decconvert2(ram,point+op1[1],1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("add "+op2+" to
mem["+op1[0]+"]");
                                        point=point+op1[1]+1;
                                        int result=decconvert2(ram,op1[0],4)+op2;
                                        writemem(ram,op1[0],result,4);
                                }
                        }
                        else if (ro==5) // SUB r/m32,imm8
                        {
                                if(mod==3)
                                {
                                        int op1=rm;
                                        int op2=decconvert2(ram,point+2,1);
```

```
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("sub "+op2+" from
"+regname[op1]);
                                        point=point+3;
                                        reg[op1]=reg[op1]-op2;
                                }
                                else
                                {
                                        int[] op1 = readmodrm(ram,point+1,reg);
                                        int op2=decconvert2(ram,point+op1[1],1);
                                        if ((op2>>7)==1)
                                                op2=(op2&127)-128;
                                        System.out.println("sub "+op2+" from
mem["+op1[0]+"]");
                                        point=point+op1[1]+1;
                                        int result=decconvert2(ram,op1[0],4)-op2;
                                        writemem(ram,op1[0],result,4);
                                }
                        }
                        else
                                System.out.println("Something went wrong with op 83");
                }
                else if(opcode==0xeb)//JMP
                {
                        int jmp=decconvert2(ram,point+1,1);
                        if ((jmp>>7)==1)
                                jmp=(jmp&127)-128;
                        point=point+jmp+2;
                        System.out.println("jmp");
                }
                else if(opcode<0x80&&opcode>0x6f)//JCC
                {
                        int jmp=decconvert2(ram,point+1,1);
                        if ((jmp>>7)==1)
                                jmp=(jmp&127)-128;
                        point=point+2;
                        if (opcode==0x74)
                        {
                                System.out.println("jmp if e");
                                if (cmpflag==1)
                                        point=point+jmp;
                        }
                        else if (opcode==0x75)
                        {
                                System.out.println("jmp if ne");
```

```
                if (cmpflag==2||cmpflag==3)
                        point=point+jmp;
        }
        else if (opcode==0x7c)
        {
                System.out.println("jmp if l");
                if (cmpflag==2)
                        point=point+jmp;
        }
        else if (opcode==0x7d)
        {
                System.out.println("jmp if ge");
                if (cmpflag==1||cmpflag==3)
                        point=point+jmp;
        }
        else if (opcode==0x7e)
        {
                System.out.println("jmp if le");
                if (cmpflag==1||cmpflag==2)
                        point=point+jmp;
        }
        else if (opcode==0x7f)
        {
                System.out.println("jmp if g");
                if (cmpflag==3)
                        point=point+jmp;
        }
        else
        {
                System.out.println("jcc error!");
        }
        cmpflag=0;
}
else if(opcode<0x48&&opcode>0x3f)//INC
{
        int r=opcode&7;
        System.out.println("Increment "+regname[r]);
        reg[r]=reg[r]+1;
        point=point+1;
}
else if(opcode<0x50&&opcode>47)//DEC
{
        int r=opcode&7;
        System.out.println("Decrement "+regname[r]);
        reg[r]=reg[r]-1;
        point=point+1;
```

```
                }
                else if(opcode==0x01)//ADD r/m32,r32
                {
                        int modrm=decconvert2(ram,point+1,1);
                        int mod=modrm>>6;
                        int op2=(modrm>>3)&7;
                        int rm=modrm&7;
                        if(mod==3)
                        {
                                int op1=rm;
                                System.out.println("add "+regname[op2]+" to
"+regname[op1]);
                                point=point+2;
                                reg[op1]=reg[op1]+reg[op2];
                        }
                        else
                        {
                                int[] op1 = readmodrm(ram,point+1,reg);
                                System.out.println("add "+regname[op2]+" to
mem["+op1[0]+"]");
                                point=point+op1[1];
                                int result=decconvert2(ram,op1[0],4)+reg[op2];
                                writemem(ram,op1[0],result,4);
                        }
                }
                else if(opcode==0x03)//ADD r32,r/m32
                {
                        int modrm=decconvert2(ram,point+1,1);
                        int mod=modrm>>6;
                        int op1=(modrm>>3)&7;
                        int rm=modrm&7;
                        if(mod==3)
                        {
                                int op2=rm;
                                System.out.println("add "+regname[op2]+" to
"+regname[op1]);
                                point=point+2;
                                reg[op1]=reg[op1]+reg[op2];
                        }
                        else
                        {
                                int[] op2 = readmodrm(ram,point+1,reg);
                                System.out.println("add mem["+op2[0]+"] to
"+regname[op1]);
                                point=point+op2[1];
                                reg[op1]=decconvert2(ram,op2[0],4)+reg[op1];
```

```
                }
            }
            else if(opcode==0x0F)//IMUL r32,r/m32
            {
                if (decconvert2(ram,point+1,1)==0xAF)
                {
                    int modrm=decconvert2(ram,point+2,1);
                    int mod=modrm>>6;
                    int op1=(modrm>>3)&7;
                    int rm=modrm&7;
                    if(mod==3)
                    {
                        int op2=rm;
                        System.out.println("mul "+regname[op1]+" by
"+regname[op2]);
                        point=point+3;
                        reg[op1]=reg[op1]*reg[op2];
                    }
                    else
                    {
                        int[] op2 = readmodrm(ram,point+2,reg);
                        System.out.println("mul "+regname[op1]+" by
mem["+op2[0]+"]");
                        point=point+op2[1]+1;
                        reg[op1]=decconvert2(ram,op2[0],4)*reg[op1];
                    }
                }
                else
                    System.out.println("IMUL Error");
            }
            else if(opcode<0x58&&opcode>0x4F)//PUSH r32
            {
                int source=opcode&7;
                reg[4]=reg[4]-4;
                writemem(ram,reg[4],reg[source],4);
                point++;
                System.out.println("Pushed "+regname[source]+" to the stack");
                bflag=(bflag<<3)+4;
            }
            else if(opcode==0x6A)//PUSH imm8
            {
                int source = decconvert2(ram,point+1,1);
                reg[4]=reg[4]-4;
                writemem(ram,reg[4],source,4);
                point=point+2;
                System.out.println("Pushed "+source+" to the stack");
```

```
        bflag=(bflag<<3)+4;
}
else if(opcode==0x68)//PUSH imm32
{
        int source = decconvert2(ram,point+1,4);
        reg[4]=reg[4]-4;
        writemem(ram,reg[4],source,4);
        point=point+5;
        System.out.println("Pushed "+source+" to the stack");
        bflag=(bflag<<3)+4;
}
else if(opcode<0x60&&opcode>0x57)//POP r32
{
        int size=bflag&7;
        int dest=opcode&7;
        if (size==2)
                reg[dest]=decconvert2(ram,reg[4],2);
        else if (size==4)
                reg[dest]=decconvert2(ram,reg[4],4);
        else
                System.out.println("Size error in Pop");
        reg[4]=reg[4]+size;
        bflag=bflag>>3;
        point++;
        System.out.println("Popped from the stack to "+regname[dest]);
}
else if(opcode==0xE8)//CALL
{
        point=point+5;
        System.out.print("Return address = "+point+" ");
        reg[4]=reg[4]-4;
        writemem(ram,reg[4],point,4);
        System.out.println("Call function");
        bflag=(bflag<<3)+4;
        int jmp=decconvert2(ram,point-4,4);
        if ((jmp>>31)==1)
                jmp=(jmp&2147483647)-2147483647-1;
        point=point+jmp;
}
else if(opcode==0xC3)//RET
{
        bflag=bflag>>3;
        point=decconvert2(ram,reg[4],4);
        reg[4]=reg[4]+4;
        System.out.println("Returned from function at "+point);
}
```

```
                else if(opcode==0xcd)//INT
                {
                        if (decconvert2(ram,point+1,1)==0x80){
                                System.out.println("Interrupt 80");
                                if (reg[0]==1){
                                        System.out.println("exit");
                                        break;}}
                        else{
                                System.out.println("some other syscall");
                                point=point+2;}
                }
                else
                {
                        System.out.println("Something went wrong!");
                        break;
                }
                opcode=decconvert2(ram,point,1);
        }

        return reg;
}

//interprets ModR/M byte based on info. in table on page 36 of Intel manual
public static int[] readmodrm(Hashtable ram, int loc, int[] reg)
{
        int[] result = new int[2];
        int modrm=decconvert2(ram,loc,1);
        int mod=modrm>>6;
        int rm=modrm&7;
        int ro=(modrm>>3)&7;
        if (mod==0)
        {
                if (rm==4)
                {
                        int sib=decconvert2(ram,loc+1,1);
                        int base=sib&7;
                        if (base==5)
                        {
                                result[0]=readsib(sib,mod,reg)+decconvert2(ram,loc+2,4);
                                result[1]=7;
                        }
                        else
                        {
                                result[0]=readsib(sib,mod,reg);
                                result[1]=3;
                        }
```

```
		}
		else if (rm==5)
		{
			result[0]=decconvert2(ram,loc+1,4);
			result[1]=6;
		}
		else
		{
			result[0]=reg[rm];
			result[1]=2;
		}
	}
	else if (mod==1)
	{
		if (rm==4)
		{
			int sib=decconvert2(ram,loc+1,1);
			int disp=decconvert2(ram,loc+2,1);
			if ((disp>>7)==1)
				disp=(disp&127)-128;
			else
				disp=disp&127;
			result[0]=readsib(sib,mod,reg)+disp;
			result[1]=4;
		}
		else
		{
			int disp=decconvert2(ram,loc+1,1);
			if ((disp>>7)==1)
				disp=(disp&127)-128;
			else
				disp=disp&127;
			result[0]=reg[rm]+disp;
			result[1]=3;
		}
	}
	else if (mod==2)
	{
		if (rm==4)
		{
			int sib=decconvert2(ram,loc+1,1);
			result[0]=readsib(sib, mod, reg)+decconvert2(ram,loc+2,4);
			result[1]=7;
		}
		else
		{
```

```
                        result[0]=reg[rm]+decconvert2(ram,loc+1,4);
                        result[1]=6;
                }
        }
        else
        {
                System.out.println("readmodrm error!");
        }
        return result;
}

//interprets SIB byte based on info. in table on page 37 of Intel manual
public static int readsib(int sib, int mod, int[] reg)
{
        int s=sib>>6;
        int scale=0;
        int index=(sib>>3)&7;
        int base=sib&7;
        if (s==0)
                scale=1;
        else if (s==1)
                scale=2;
        else if (s==2)
                scale=4;
        else if (s==3)
                scale=8;
        else
                System.out.println("readsib error!");
        int si = 0;

        if (index!=4)
                si=reg[index]*scale;

        if (base!=5)
                return reg[base]+si;
        else if (mod==0)
                return si;
        else
                return reg[5]+si;
}

//prints out all information in ELF header, program headers and section headers
public static void printfileinfo(int[] elf, char[] file)
{
        System.out.printf("\nELF HEADER\nType Code: %x\n", elf[0]);
        System.out.printf("Machine Code: %x\n", elf[1]);
```

```
                System.out.printf("Version: 0x%x\n", elf[2]);
                System.out.printf("Entry point address: 0x%x\n", elf[3]);
                System.out.println("Start of program headers: "+elf[4]+" (bytes into file)");
                System.out.println("Start of section headers: "+elf[5]+" (bytes into file)");
                System.out.printf("Flags: 0x%x\n", elf[6]);
                System.out.println("Size of ELF Header: "+elf[7]+" (bytes)");
                System.out.println("Size of program headers: "+elf[8]+" (bytes)");
                System.out.println("Number of program headers: "+elf[9]);
                System.out.println("Size of section headers: "+elf[10]+" (bytes)");
                System.out.println("Number of section headers: "+elf[11]);
                System.out.println("Section Header String Table Index: "+elf[12]);

                for (int i=0; i<elf[9]; i++)
                {
                        int base= elf[4]+elf[8]*i;
                        System.out.println("\n"+"Program Header "+(i+1)+" of "+elf[9]);
                        System.out.printf("Type Code: %x\n", decconvert(file,base,4));
                        System.out.printf("Offset: 0x%x\n", decconvert(file,base+4,4));
                        System.out.printf("Virtual Memory Address: 0x%x\n",
decconvert(file,base+8,4));
                        System.out.printf("Physical Address: 0x%x\n",
decconvert(file,base+12,4));
                        System.out.printf("Bytes in file image: 0x%x\n",
decconvert(file,base+16,4));
                        System.out.printf("Bytes in memory image: 0x%x\n",
decconvert(file,base+20,4));
                        System.out.printf("Flags: 0x%x\n", decconvert(file,base+24,4));
                        System.out.printf("Segment Alignment: 0x%x\n",
decconvert(file,base+28,4));
                }

                for (int i=0; i<elf[11]; i++)
                {
                        int base= elf[5]+elf[10]*i;
                        System.out.println("\n"+"Section Header "+(i+1)+" of "+elf[11]);
                        System.out.println("Name Index: "+decconvert(file,base,4));
                        System.out.printf("Type Code: %x\n", decconvert(file,base+4,4));
                        System.out.printf("Flags: 0x%x\n", decconvert(file,base+8,4));
                        System.out.printf("Address: 0x%x\n", decconvert(file,base+12,4));
                        System.out.printf("Offset: 0x%x\n", decconvert(file,base+16,4));
                        System.out.printf("Size: 0x%x\n", decconvert(file,base+20,4));
                        System.out.printf("Link: 0x%x\n", decconvert(file,base+24,4));
                        System.out.printf("Info: 0x%x\n", decconvert(file,base+28,4));
                        System.out.printf("Section Alignment: 0x%x\n",
decconvert(file,base+32,4));
                        System.out.printf("Entry Size: 0x%x\n", decconvert(file,base+36,4));
```

```
        }
        System.out.println();
}

//returns integer value of 1,2 or 4 byte piece of information in 'file'
public static int decconvert(char[] source, int loc, int size)
{
        int result=0;
        for (int i=0; i<size; i++)
        {
                result=result+(((int)source[loc+i])<<(8*i));
        }
        return result;
}

//returns integer value of 1,2 or 4 byte piece of information in 'ram'
public static int decconvert2(Hashtable source, int loc, int size)
{
        int result=0;
        for (int i=0; i<size; i++)
        {
                Character info = (Character)source.get(""+(loc+i));
                result=result+(info.hashCode()<<(8*i));
        }
        return result;
}

//method which controls writing to memory, can write 1,2 or 4 bytes at a time
public static void writemem(Hashtable source, int loc, int info, int size)
{
        if (size==1)
        {
                String newstring = ""+loc;
                source.put(newstring, new Character((char)info));
        }
        else if (size==2)
        {
                String newstring1 = ""+(loc+1);
                String newstring2 = ""+loc;
                int info1=(info>>8)&255;
                int info2=info&255;
                source.put(newstring1, new Character((char)info1));
                source.put(newstring2, new Character((char)info2));
        }
        else if (size==4)
        {
```

```
                String newstring1 = ""+(loc+3);
                String newstring2 = ""+(loc+2);
                String newstring3 = ""+(loc+1);
                String newstring4 = ""+loc;
                int info1=(info>>24)&255;
                int info2=(info>>16)&255;
                int info3=(info>>8)&255;
                int info4=info&255;
                source.put(newstring1, new Character((char)info1));
                source.put(newstring2, new Character((char)info2));
                source.put(newstring3, new Character((char)info3));
                source.put(newstring4, new Character((char)info4));
        }
        else
                System.out.println("Size error for writemem");

    }

    //if 'filename' is invalid, getsize returns 0, otherwise it returns the size of the file (in bytes)
    public static int getsize(String filename)
    {
        int sizecounter=0;
        try{
                FileInputStream fis = new FileInputStream(filename);
                DataInputStream dis = new DataInputStream(fis);
                while(true)
                {
                        //readUnsignedByte will fail when we reach end of file, thus
diverting getsize to the catch
                        dis.readUnsignedByte();
                        sizecounter++;
                }
        }
        catch(IOException e)
        {
                return sizecounter;
        }
    }
}
```