**Handbook**

**Team:**

Omoniyi A. Akande, Kaiwen Zhu, Yunzhi Ling, Hong Xuan

**Configuration:**

Language: JAVA

IDE: Eclipse

Version Control: Github

**Structure:**

Program is made up of three parts, Panel.java, Register.java, Register\_Set.java. Panel.java generates the UI which will be interacted with users. Register.java is the class of every Registers. It declares all the methods of register. Register\_Set.java is the control unit in this program. It will instantiate every register base on Register.java, and offers the method to decode the instruction and controls the Registers to implement the instruction. This design separates every part of program clearly, so that the program is easy to maintain and add new function.

|  |  |  |
| --- | --- | --- |
| **Class** | **Attributes** | **Method** |
| Panel.java  //The class creates UI | All element in UI. | UpdateFields() //initiate information of all registers  UpdateSwitchRegisterVal() //initiate the switch register.  BitToint(int[]) //convert the int[] to int  DoIPL() //implement IPL button.  DoSingleStep() //implement SingleStep button.  DoClear() //implement Clear button.  DoAL() //implement Address load button.  DoDP() //implement Deposit button.  Get\_Instruction() //fetch instruction from Memory |
| Register.java  //The base of all register | int Length; //the length of axis 1  int Height;  //the length of axis 2  int Memory[][];  //storage  int Output[];  int Pointer;  byte Flag; | Insert(int[],int) //insert data in register  Insert(int,int)  Binary\_to\_dec(int[])  Dec\_to\_dec(int)  Output(int[]) //output word  Output(int)  OutputAsString()  OutputAsInt()  OutputAsInt()  Pop()  UpdatePoint() //Pointer++ |
| Register\_Set.java  //The control unit | //GPR  Register R0;  Register R1;  Register R2;  Register R3;  //IR  Register X1;  Register X2;  Register X3;  //Main Memory  MainMemory Memory;  //PC  Register PC;  Register CC;  Register IR;  Register MAR;  Register MBR;  Register MSR;  Register MFR;  //GPR | Register\_Set() //initial functionality  Binary\_to\_dec(int[]) //convert binary to dec  Decoder(int[]) //decode the instruction  Get\_EA(int,int,int) //calculate Effective Address  LDR(int,int,int) //implement instruction LDR  STR(int,int,int) //implement instruction STR  LDA(int,int,int) //implement instruction LDA  LDX(int,int,int) //implement instruction LDX  STX(int,int,int) //implement instruction STX  JZ(int,int,int)  JNE(int,int,int,int)  JCC(int,int,int,int)  JMA(int,int,int,int)  JSR(int,int,int,int)  RFS(int,int,int,int)  SOB(int,int,int,int)  JGE(int,int,int,int)  AMR(int,int,int,int)  SMR(int,int,int,int)  AIR(int,int,int,int)  SIR(int,int,int,int)  MLT(int,int)  DVD(int,int)  TRR(int,int)  AND(int,int)  ORR(int,int)  NOT(int)  SRC(int,int,int,int)  RRC(int,int,int,int)  IN(int,int)  OUT(int,int)  CHK(int,int) |
| MainMemory.java  //MainMemory extends from Register | int nline;  int nrow;  int nword;  int[][] Address\_Table;  Cache cache; | UpdatePoint()  Initiate\_table()  Output(int[] )  Output(int)  Separate(int) |
| Cashe.java | int[][][] AM;//Associative Memory 0 is whether in Cash, 1 is which line in the group  int[][][][] HSM;//High Speed Memory  int HSMHeight;  int[] HSMpointer;//[line] check the HSM whether full of instruction.  int[][] HSMcheck;//check the HSM whether is empty, 1 is no, 0 is empty.  int nline;  int nrow;  int nword;//how many words in a units.  int[][] Table;  int linen;  int rown;  int wordn;  int Length; | Search(int,int,int,int[][])  InsertAddress(int,int,int,int[][]) |
| InputDevice.Java | Int[] Data  Int point; | Filter(string)  Output()  isInteger(string) |
| PrinterDevic.Java | output | Filter(string)  Output(string) |

Test Code:

|  |  |  |
| --- | --- | --- |
| Advance Code | Instruction | Machine Code |
| n=0;  for(int i=0;i<10;i++){  n=n+1;  }  Print n; | (6) R1<-10  (7) AMR R0<-R0+c(11)  (8) SOB R1<-R1-1  (9) X1<-c(12)  (10)JCC PC<-12  (11) 1  (12)OUT | 0000110100001010  0001000000001011  0100000100000111  1010010001001100  0011000100001100  0000000000000001  1111100000000001 |

**Cashe:**

**There are three parts in the Cashe. They are HSM(High Speed Memory), HSMCheck and AM(Associative Memory).**

The Structure of Memory is 16 lines, 16 rows, 16 groups and each group is 16 bits.

The Structure of AM is 16 lines, 16 rows, 1 groups and each group is 2 bits. The first bit stores 0 or 1, 1 means Memory in the Cache. The second bit stores the row in the HSM.

The Structure of HSM is 16 lines, 4 rows, 16 groups and each group is 16 bits. The rows of HSM is less than Memory. If there isn’t what we need in Cashe, the Cashe will insert the Whole goup from memory which has what we need.

The Structure of HSMCheck is 16 lines, 4 rows, 1 goups and each group is 1 bits. The HSMCheck record the order of group in the HSM. The HSMCheck decides which group will be replaced first.

Whenever obtain EA from decoder program will check Cashe first. If we can find what we need, program will fetch instruction from Cashe. If not, Cashe will pop old group according FIFO and insert new group from Memory and then pop the instruction from Cashe.

**User Interface:**

****

1. Display windows for registers. Allows user to see what value, in binary, is currently stored in each register.
2. Switch Register. Allows user to enter a 16 bit value that can be deposited (using the “Deposit”
3. Input field. Field where the user can enter in text that can be submitted to the program to be run using the “ENTER” button
4. Printer. Field display any data that the program sends to printer
5. Dev Console. Displays useful information about the Instructions being executed by the computer
6. IPL button. Used to set up the computer for execution. Needs to be pressed first before the any program instructions can be executed properly.
7. Single Step button. Executes a single instruction (loads Instruction from memory location pointed to by the PC, executes the instruction, increments PC unless otherwise moved by the instruction).
8. Load Text button. Used to load the text file that will be loaded into memory for use by Program 2.
9. Load Program From File button. Used to load the program to be run by the computer.
10. Address Load button. Loads the MBR with the data at the location in memory designated by the switch register.
11. Deposit button. Stores the value designated by the switch register into memory at the address currently in the MAR.
12. Run button. Continuously executes a “Single Step” until an error or a Halt instruction is reached.
13. Halt button. Stops the “Run” process.
14. Clear button. Clears the switch registers.

**Manual Operation:**

Step 1: Click the IPL button to initiate the Program variables.

Step 2: Input Binary Instruction from Switch Register.

Step 3:

Click the function button to choose what you want to do.

Click Deposit to store the data in Switch Register to Memory.

Click Address Load to show the word which store in address in Switch Register in Memory. Click Clear to clear the panel. Click single Step to execute one instruction in Memory. Click Run to execute all instructions in Memory.

Click Halt to stop the program.

**Test Program 1:**

Step 1: Open Program.

Step 2: Click IPL

Step 3: Click “Load Program From File”, Choose Program1.txt

Step 4: Click Run button to let Program 1 run

Step 5: Enter 20 numbers, separated by spaces, into the INPUT field then click Enter

Step 6: Enter 1 number into the INPUT field then click the “ENTER” button

**Test Program 2:**

Step 1: Open Program.

Step 2: Click IPL

Step 3: Click “Load Program From File” button, Choose Program2.txt

Step 4: Click “Load Text” button, Choose testFile.txt

Step 5: Click Run button to let Program 2 run

Step 6: Enter the text that you wish to search for into the INPUT field then click the “ENTER” button

Step 6a: Because of the way input was implemented, must hit “ENTER” an additional time for each character in the string EX: input “am” click enter 3 times