IAS processor design

Course Name: EG 212 Computer Architecture – Processor design
Student Name: Kanav Bhardwaj, Lesin, Ivan Bhargava
Student ID: IMT2023024, IMT2023565, IMT2023022

Jan 2024

1 PROJECT DESCRIPTION:

1. Writing a C program and writing the assembly code for the C program. 2. Coding an assembler to convert assembly code to machine code. 3. Coding a processor to decode and execute the machine code output given by the assembler.

2 C Program Implemented:

To print the standard deviation of four integers

```c
#include<stdio.h>
#include<math.h>

int main(){
    int a[4]={8,16,24,32}; //initialize array with 4 integers
    double sum=0,avg,sum2=0,std;
    for(int i=0;i<4;i++){
        sum=sum+a[i]; //summing int
    }
    avg=sum/4; //taking avg
    for(int i=0;i<4;i++){
        double k=(a[i]-avg);
        double t=pow(k,2);
        sum2=sum2+t; //summing (x-m)**2
    }
    std=sqrt(sum2/4); // taking avg and square rooting it
    printf("%f", (std));
}
```

3 Memory and Assembly Code:

We have implemented the memory in a python dictionary. The data is from the memory 1 to 8. The instructions are from memory 9 to 27.

4 NEW INSTRUCTIONS INTRODUCED:

1.NOP OPCODE: 00100010 FUNCTION: It just passes onto next line
2.SQUARE: OPCODE: 00100011 FUNCTION: Squares the value in AC.
2.SQUAREROOT: OPCODE: 00100100 FUNCTION: Takes square root of value in AC.

5 ASSEMBLER:

1. The method opens the assembly file (Assembly code.txt) in read mode and the machine code file (Machine code.txt) in write mode. 2. It iterates through each line in the assembly file (fileI). 3. Each line is split into a list of words (instruc), and the leading word is extracted (word). 4. If the word is present in the opcode dictionary (instruc), the corresponding binary code is retrieved and appended to binarycode. 5. If the word is not in the dictionary, it is assumed to be a numerical value. In this case, the code converts the numerical value to a 40-bit binary string and appends it to binarycode. 6. The resulting binarycode is then padded with zeros to ensure a total length of 40 bits. 7. The final binary code is written to the machine code file. The assembler begins by retrieving assembly code from an external text file. Subsequently, it transforms the assembly code into machine code, and the resulting output is written to a separate external file.

6 INPUT FILE: ASSEMBLY CODE

```
 1    8
 2    16
 3    24
 4    32
 5    0
 6    1
 7    -4
 8    1
 9    LOAD_M 1 ADD_M 2
10    ADD_M 3 ADD_M 4
11    RSH RSH
12    STORE_M 5 NOP
13    LOAD_M 6 STORE_M_28:39 18
14    STORE_M_8:19 17 LOAD_M 7
15    JMP+_M_0:19 21 ADD_M 8
16    STORE_M 7
17    LOAD_M 2 SUB_M 5
18    SQUARE STORE_M 2
19    LOAD_M 6 ADD_M 8
20    STORE_M 6 JMP_M_0:19 13
21    LOAD_M 1 ADD_M 2
22    ADD_M 3 ADD_M 4
23    LOAD_M 2 SUB_M 5
24    SQUARE STORE_M 2
25    RSH RSH
26    SQUAREROOT STORE_M 5
27    HALT
```

7 OUTPUT FILE: MACHINE CODE

```
 1    00000000000000000000000000000001000
 2    00000000000000000000000000000010000
 3    00000000000000000000000000000011000
 4    00000000000000000000000000000100000
 5    00000000000000000000000000000000000
 6    00000000000000000000000000000000001
 7    00000000000000000000000011111111100
 8    00000000000000000000000000000000001
 9    00000001000000000001000001010000000010
10    00000101000000000011000001010000000100
11    00010101000000000000000101010000000000
12    00100001000000000101001000100000000000
13    00000001000000000110000100110000000010
14    00010010000000010001000000100000000111
15    00001111000000001010100000101000000001000
16    00100001000000000111000000000000000000
17    00000001000000000010000001100000000101
18    00100011000000000000010000100000000010
19    00000001000000000110000001010000000001000
20    00100001000000000110000011010000000001101
21    00000001000000000010000001010000000000010
22    00000101000000000011000001010000000000100
23    00000001000000000010000001100000000000101
24    00100011000000000000010000100000000000010
25    00010101000000000000000101010000000000000
26    00100100000000000000010000100000000000101
27    00000000000000000000000000000000000000000
```

8 PROCESSOR:

The processor code has been implemented using python programming language. The machine code input has been taken from the output file produced by the assembler and then stored in a instruction list. The data in the memory has been stored as a data list. Together the data list and the instruction list form the memory list. PC, address [MAR], MBR, IBR, opcode [IR], AC, MQ registers have been implemented in the form of variables and PC has been set to the index of first instruction. The fetch, decode and execute cycles for each instruction has been implemented using loops and conditional statements.

1. FETCH phase: a. PC value goes to MAR. b. MAR value(address) is fetched from main memory and sent to the MBR. c. The right instruction in MBR is stored in IBR d. The left instruction is split into opcode and address. Opcode is stored in IR and address is stored in MAR. e. PC = PC + 1
2. DECODE phase: The specific instruction is decoded using its opcode.
3. EXECUTE phase : The decoded instruction is then executed accordingly.

9 PROCESSOR OUTPUT:

```
AC: 00000000000000000000000000000000010000 | PC: 9 | MAR: 8 | MBR: 00000001000000000001000001010000000010
AC: 00000000000000000000000000000000010010 | PC: 9 | MAR: 8 | MBR: 00000001000000000001000001010000000010
AC: 00000000000000000000000000000000010101 | PC: 10 | MAR: 9 | MBR: 00000101000000000011000001010000000100
AC: 00000000000000000000000000000000011001 | PC: 10 | MAR: 9 | MBR: 00000101000000000011000001010000000100
AC: 001100 | PC: 11 | MAR: 10 | MBR: 00010101000000000000001010100000000000
AC: 00110 | PC: 11 | MAR: 10 | MBR: 00010101000000000000001010100000000000
AC: 00110 | PC: 12 | MAR: 11 | MBR: 00100001000000000101001000100000000000
AC: 00110 | PC: 12 | MAR: 11 | MBR: 00100001000000000101001000100000000000
AC: 00000000000000000000000000011111111100 | PC: 13 | MAR: 12 | MBR: 00000001000000000110000100110000000010010
AC: 00000000000000000000000000011111111100 | PC: 13 | MAR: 12 | MBR: 00000001000000000110000100110000000010010
AC: 00000000000000000000000000011111111100 | PC: 14 | MAR: 13 | MBR: 00010010000000010001000000010000000000111
AC: 00000000000000000000000000000000000001 | PC: 14 | MAR: 13 | MBR: 00010010000000010001000000010000000000111
AC: 00000000000000000000000000000000000001 | PC: 21 | MAR: 14 | MBR: 00001111000000010101000001010000000001000
AC: 00000000000000000000000000000000001001 | PC: 21 | MAR: 14 | MBR: 00001111000000010101000001010000000001000
AC: 00000000000000000000000000000000001100 | PC: 22 | MAR: 21 | MBR: 00000101000000000011000001010000000000100
AC: 00000000000000000000000000000000010000 | PC: 22 | MAR: 21 | MBR: 00000101000000000011000001010000000000100
AC: 00000000000000000000000000000000011000 | PC: 23 | MAR: 22 | MBR: 00000001000000000010000001100000000000101
AC: 00000000000000000000000000000000010011 | PC: 23 | MAR: 22 | MBR: 00000001000000000010000001100000000000101
AC: 00000000000000000000000000000000101101001 | PC: 24 | MAR: 23 | MBR: 00100011000000000000001000010000000000010
AC: 00000000000000000000000000000000101101001 | PC: 24 | MAR: 23 | MBR: 00100011000000000000001000010000000000010
AC: 0010110100 | PC: 25 | MAR: 24 | MBR: 00010101000000000000001010100000000000
AC: 001011010 | PC: 25 | MAR: 24 | MBR: 00010101000000000000001010100000000000
AC: 00000000000000000000000000000000001001 | PC: 26 | MAR: 25 | MBR: 00100100000000000000001000010000000000101
AC: 00000000000000000000000000000000001001 | PC: 26 | MAR: 25 | MBR: 00100100000000000000001000010000000000101
AC: 00000000000000000000000000000000001001 | PC: 27 | MAR: 26 | MBR: 00000000000000000000000000000000000000000
AC: 00000000000000000000000000000000001001 | PC: 27 | MAR: 26 | MBR: 00000000000000000000000000000000000000000
standard deviation=00000000000000000000000000000000001001
```