Project Report-IAS Architecture

Course Name: EG 212, Computer Architecture

Student Names: Satyam Ambi,Hitanshu Seth,Mayank Tanwar
Student IDs: IMT2023623, IMT2023100, IMT2023012

26 January 2024

Introduction

INTRO

This report presents detailed analysis of the IAS architecture which we have implemented using the C++ programming language.

1 CALCULATING NUMBER OF PERMUTATIONS

In our computer architecture project we have showcased the data flow in a processor by using C++ programming language.we have also written the code for the same in assembly language.Further we have converted that assembly code into machine language code and then showed the flow of data in IAS Architecture by using various instructions. We have also implemented two of our own instructions CMP(compare) and FACT(factorial) and finally by using the above processes we have computed the no of permutations of two numbers.

1.1 First Instruction FACT(Opcode:10111011)

This instruction can be called by using FACT M(X) followed by memory address as shown below.This helps us calculate to calculate the factorial of the number present in the memory and store the result in accummulator.

```
case 0b10111011:
    printf("Detected FACT M(x)\n");
    MBR.data = MainMemory[MAR];

    // Initialize ACC to 1 for factorial computation
    ACC = 1;

    // Perform factorial computation in a loop
    while (MBR.data >= 1) {
        ACC *= MBR.data;
        MBR.data--;
    }

    printf("Fact Successful\n");
    PrintStatus_Instruction();
    break;
case 0b11111111:
```

Figure 1: Choices

1.2 Second Instruction CMP (Opcode:11111111)

This instruction can be called by using CMP M(X) followed by memory location this instruction compares the value in ACC and in memory.If the value in ACC is less than the memory then the process terminates as we cannot calculate permutations where n is less than r else it continues the process.

```
case 0b11111111:
    MBR.data = MainMemory[MAR];
    if(ACC < MBR.data)
    {
        printf("Invalid input(your n must be greater than r)");
        PC_index+=5;
    }
    else
    {
        printf("Valid Input");
    }
    break;
```

Figure 2: CMP

2 RESULTS

2.1 Input

Here we are asking the user if he wants to calculate the permutation or wants to exit the machine

```
Welcome to our IAS machine
Input 1 to compute no of permutations of a number.
Input 2 to switch off the machine.
This program displays values in registers and instructions running during each step.

Enter your choice
```

Figure 3: CHOICES

2.2 Data flow in case of n less than r

If our n is less than r then the process terminates in the CMP instruction

```
Welcome to our IAS machine
Input 1 to compute no of permutations of a number.
Input 2 to switch off the machine.
This program displays values in registers and instructions running during each step.

Enter your choice
1
Enter the values of n and r
2 5
Detected LOAD M(X)Load successful
PC = 0
IR = 1
MAR = 100
IBR = 1044581
MBR = 2
Left Instruction = 4196
Right Instruction = 1044581

Invalid input(your n must be greater than r)
Enter your choice
```

Figure 4: Data Flow in case of invalid input

2.3 Data flow if n is more than r

This gives us the output desired by us by using the given inputs

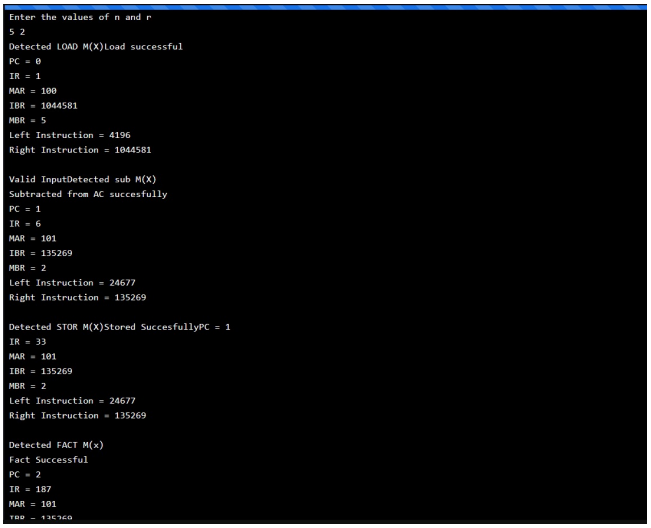

Figure 5: Data Flow

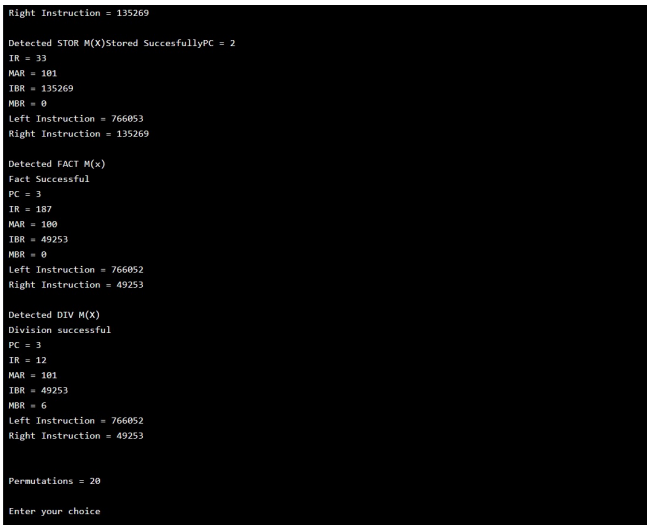

Figure 6: Data Flow

2.4 C++ PROGRAM ON WHICH OUR PROCESSOR IS DOING COMPUTATION

References

```cpp
#include <bits/stdc++.h>

using namespace std;
long long nPr(int n, int r) {
    if (n < r) {
        printf("Invalid input");
        return 0;
    }

    long long result = 1;


    for (int i = 0; i < r; ++i) {
        result *= (n - i);
    }

    return result;
}

int main() {
    int n, r;

    cout << "Enter the value of n and r: ";
    cin >> n >> r;



    // Calculate and print NPR
    cout << "Permutations(" << n << ", " << r << ") = " << nPr(n, r) << endl;

    return 0;
}
```

Figure 7: C++ Version of the program

```
MainMemory[0]: 00000001000001100100111111110000001100101
MainMemory[1]: 00000110000001100101001000010000001100101
MainMemory[2]: 10111011000001100101001000010000001100101
MainMemory[3]: 10111011000001100100000011000000001100101
```

Figure 8: Assembler Output