Computer Architecture

Assignment 1

Ankit Agrawal

IMT2019010

Contents

1 Submission Details

My submission for Assignment 1 consists of the following files:

1. **IMT2019010_NaturalNumbers.cpp**: This C++ file contains the program of IAS Machine (with memory programmed for executing program for finding sum of first 10 natural numbers). This file can be opened with the help of any text editor such as Notepad and Sublime Text Editor.

2. **IMT2019010_Fibonacci.cpp**: This C++ file contains the program of IAS Machine (with memory programmed for executing program for finding first 10 Fibonacci numbers). This file can be opened with the help of any text editor such as Notepad and Sublime Text Editor.

3. **IMT2019010_EmptyIASMachine.cpp**: This IAS machine does not contain any pre-programmed memory. This empty IAS machine can be used once it's memory has been programmed. I am submitting this file so that the work of copying the code for IAS machine when implementing a new program in it can be avoided.

4. **IMT2019010_SubmissionReport.pdf** : This file contains the instructions supported, steps for compiling and running the programs on Ubuntu, programs executed and References that were used by me for creating the program.

5. **IMT2019010_Compiling and Running.m4v**: This video file shows step-by-step compilation and running of the program. This video file can be opened with the help of Windows Media Player and VLC Media Player. Any other media player should also be able to open this file.

2 Features of the Program

There are many features of the program, some of which are as follows:

1. Support for **all** 21 instructions (and additional instruction HALT)

2. Ability to display state of IAS machine after execution of an instruction

3. Memory can be programmed easily to create and build programs on the IAS machine

3 Creating a new program in IAS computer

3.1 Programming the memory

In order to create a new program, we will need to program the memory of the IAS computer so that it holds the instructions which will be executed. The following function is used to program the memory of the IAS computer:

```
Program_Memory(int Memory_Location, string Instruction);
```

The above function has two arguments, which are described below:

- `int Memory_Location`: This is the memory location which is going to be programmed. In other words, the 40 bit instruction will be put at this memory location. In the IAS computer, the memory location ranges from 0 to 999. So, the instructions (or data) can be put at any memory location starting from 0 and ending at 999. Please note that this value is in decimal and *not* binary.
- `string Instruction`: This argument is the instruction (or data) to be programmed in the IAS computer. The instruction should have a total of 40 bits, without spaces, and should be enclosed within " ".

Example: Suppose we would like to program the memory location 399 so that it contains the number 40. As the number 40 in binary is 101000, we would use the following code to achieve this task:

```
Program_Memory(399,"0000000000000000000000000000000000101000");
```

Please note that in the above example, the most significant bit is 0, as the number 40 is positive. Also, the length of the instruction is 40 bits. **Also, the ';' (semicolon) at the end is compulsory**

Example: Suppose we would like to program the memory location 100 so that it contains the instructions "LOAD M(100) ADD M(102)". The first 20 bits (which are for LOAD M(100)) of the instruction are 00000001 000001100100 and the next 20 bits (which are for ADD M(102)) of the instruction are 00000101 000001100110. So, we write the following code to program the memory:

```
Program_Memory(100,"0000000100000110010000000101000001100110");
```

Please see the **Instruction Set** for the list of Opcodes of instructions.

3.2 Inserting the code at correct place

Now that we know how to program the memory correctly, we need to insert this code at the correct place in order to make the program work correctly. The code for the IAS machine looks as shown below:

```
Some Code for IAS machine...
int main()
{
    long long int Display_Memory_Start = 0;
    long long int Display_Memory_End = 999;
    ⟨ Program Memory here ⟩
    ⟨ Program Memory here ⟩
    ⟨ Program Memory here ⟩
                ⋮
    ⟨ Program Memory here ⟩
    Reset_IAS();
    while(1)
    {
         Code inside while loop...
     }
     Some more code for IAS machine...
}
```

We need to insert the lines which program the memory at the positions indicated by `⟨ Program Memory here ⟩`, as shown below:

```
Some Code for IAS machine...
int main()
{
    long long int Display_Memory_Start = 0;
    long long int Display_Memory_End = 999;
    Program_Memory(100,"0000000100000110010000000101000001100110");
    Program_Memory(399,"0000000000000000000000000000000000101000");
    Reset_IAS();
    while(1)
    {
         Code inside while loop...
     }
     Some more code for IAS machine...
}
```

3.3 Few things to take care of ...

Once the memory has been programmed correctly, the code containing the programmed memory and the IAS machine can be compiled in order to run the instructions. However, there are few things which must be taken care while programming the memory, and running the program:

- The value of PC (i.e. Program Counter) is 0 at the beginning of the program. This means that the IAS computer will begin executing instructions starting from address 0.

- In case the Opcode of an instruction does not match with any of the Opcodes mentioned in the **Instruction Set**, the IAS machine will skip that instruction, and proceed to the next instruction, without halting.

- The value of numbers, and result of arithmetic operations in the IAS computer is limited by the maximum and the minimum number that C++ allows to store and compute, without causing overflows. C++ allows users to store numbers which are in the range of $[-2^{63}, 2^{63} - 1]$. Any number out of this range causes overflow, which C++ cannot handle, and might result in errors while performing calculations.

- The memory address runs from 0 to 999 in the IAS computer.

- Please avoid modifying the parts of the IAS machine which have not been described in this document, as this may result in malfunctioning of the IAS machine.

- Because the left instruction in the IAS machine is optional, sometimes a 40-bit word might not contain the left instruction. In such cases, please keep the left instruction as "00000000000000000000". Although, any other left instruction whose first 8 bits (i.e. the opcode of the left instruction) does not match any of the Opcodes mentioned in the **Instruction Set** will be fine, however, it is better to avoid it for safety.

4 Compiling the program on Linux

The video file "IMT2019010_Compiling and Running.mp4" shows detailed steps on how to compile and run the program.

Following are the steps for compiling and running the file which is named "IMT2019010_NaturalNumbers.cpp"

1. Open the terminal window and navigate to the directory where the submission files have been saved.

2. In the terminal window, type the following command to compile the file "IMT2019010_NaturalNumbers.cpp":

```
g++ IMT2019010_NaturalNumbers.cpp
```

3. This will generate the file "a.out" file in the same directory containing the submission file. Now, run this by putting the following command in the terminal window:

```
./a.out
```

4. The output could be seen the on the terminal window. Please see the section **Result of Execution of Programs** for the result of program.

In order to view the intermediate state of the IAS machine (i.e. the values of AC, MBR...) after execution of each instruction, please follow the following steps:

1. Open the file "IMT2019010_NaturalNumbers.cpp" with the help of any text editor

2. Find the "main" function by scrolling down the code. Inside the "main" function, there is a "while" loop. At the end of the while loop, there are the following two lines:

```
        //display_IAS(); //This piece of code ...
//cout<<"--------------------------------------"<<"\n";
```

(Please see the next page for the remaining steps . . .)

3. We will uncomment these lines, i.e. delete "//" from the front of these lines, as shown below:

```
       display_IAS(); //This piece of code ...
cout<<"---------------------------------------"<<"\n";
```

4. Now, compile and run the file again to view the state of machine after execution of each instruction.

The next page of this document shows the Instruction Set of the IAS machine.

5 Instruction Set

Opcode	Symbol	Comments
00001010	LOAD MQ	Transfer MQ to AC
00001001	LOAD MQ,M(X)	Transfer M(X) to MQ
00100001	STOR M(X)	Transfer AC to M(X)
00000001	LOAD M(X)	Transfer M(X) to AC
00000010	LOAD −M(X)	Transfer −M(X) to AC
00000011	LOAD |M(X)|	Transfer |M(X)| to AC
00000100	LOAD −|M(X)|	Transfer −|M(X)| to AC
00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
00001111	JUMP+ M(X,0:19)	If AC $\geq$ 0, take next instruction from left half of M(X)
00010000	JUMP+ M(X,20:39)	If AC $\geq$ 0, take next instruction from right half of M(X)
00000101	ADD M(X)	AC = AC + M(X)
00000111	ADD |M(X)|	AC = AC + |M(X)|
00000110	SUB M(X)	AC = AC − M(X)
00001000	SUB |M(X)|	AC = AC − |M(X)|
00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
00010100	LSH	Shift left one bit position
00010101	RSH	Shift right bit one position
00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC
11111111	HALT	New instruction : halts the machine

The above table lists 22 instructions.The original IAS has 21 instructions. The HALT instruction is a new instruction, which was not present originally.

6 Implementation of Fetch Cycle

The fetch cycle is an important cycle in the IAS machine, which must be run in order to proceed to the execute cycle. The fetch cycle is implemented by 3 different functions in the program, each of which execute depending upon the value of the variable `"Flag"` in the program.

The three functions which are responsible for running the fetch cycle are described below:

1. `Fetch_Cycle_Left`: This function is responsible for running the fetch cycle when we have to execute the left instruction. This function copies RHS instruction to IBR, copies the first 8 bits of LHS instruction to the IR and the next 12 bits of the LHS instruction to the MAR.

2. `Fetch_Cycle_Right_No_Deviation`: This function is responsible for running the fetch cycle when we have to execute the right instruction, provided that the left instruction (which is in the same address as the address of the right instruction) has already been executed. This function copies the first 8 bits of IBR to IR, and the remaining 12 bits of the IBR to MAR. This function also increments the value of PC by 1, once it has completed the fetch cycle.

3. `Fetch_Cycle_Right_Deviation`: This function is responsible for running the fetch cycle when we have to execute the right instruction, provided that the left instruction (which is in the same address as the address of the right instruction) has not been executed. This function copies the value of PC to MAR, reads the instruction at the address contained by MAR from memory and copies it to MBR. Finally, it copies the first 8 bits of RHS instruction from MBR to IR, next 12 bits to MAR, and increments the value of PC by 1.

The next page of this document describes how different instructions are executed by the IAS machine in the execute cycle . . .

7 Steps for executing instructions in Execute Cycle

This section lists the steps that have been followed in order to perform a specific instruction in the IAS machine. May kindly note that we assume that the fetch cycle has already been executed before the starting of execute cycle.

S. No.	Instruction	Steps followed
1	LOAD MQ	MQ ← AC

2	LOAD MQ,M(X)	MBR ← M(X)
		MQ ← MBR

3	STOR M(X)	AC ← MBR M(X) ← MBR

4	LOAD M(X)	MBR ← M(X)
		AC ← MBR

5	LOAD −M(X)	MBR ← M(X)
		ALU ← MBR
		AC ← ALU

6	LOAD |M(X)|	MBR ← M(X)
		ALU ← MBR
		AC ← ALU

7	LOAD −|M(X)|	MBR ← M(X)
		ALU ← MBR
		AC ← ALU

8	JUMP M(X,0:19)	PC ← MAR Set's flag to start fetch cycle for new address
9	JUMP M(X,20:39)	PC ← MAR Set's flag to start fetch cycle for new address

S. No.	Instruction	Steps followed
10	JUMP+ M(X,0:19)	**If AC $\geq$ 0** PC $\leftarrow$ MAR Set's flag to start fetch cycle for new address **Otherwise** Start fetch cycle for next instruction
11	JUMP+ M(X,20:39)	**If AC $\geq$ 0** PC $\leftarrow$ MAR Set's flag to start fetch cycle for new address **Otherwise** Start fetch cycle for next instruction
12	ADD M(X)	MBR $\leftarrow$ M(X) ALU $\leftarrow$ MBR* ALU $\leftarrow$ AC* AC $\leftarrow$ ALU *ALU inputs both AC, MBR and puts result in AC
13	ADD|M(X)|	MBR $\leftarrow$ M(X) ALU $\leftarrow$ MBR* ALU $\leftarrow$ AC* AC $\leftarrow$ ALU *ALU inputs both AC, MBR and puts result in AC
14	SUB M(X)	MBR $\leftarrow$ M(X) ALU $\leftarrow$ MBR* ALU $\leftarrow$ AC* AC $\leftarrow$ ALU *ALU inputs both AC, MBR and puts result in AC

S. No.	Instruction	Steps followed

15	SUB |M(X)|	MBR ← M(X)
		ALU ← MBR*
		ALU ← AC*
		AC ← ALU
		*ALU inputs both AC, MBR and puts result in AC

16	MUL M(X)	MBR ← M(X)
		ALU ← MBR*
		ALU ← MQ*
		AC ← ALU[0:39]
		MQ ← ALU[40:79]
		*ALU inputs both MQ, MBR

17	DIV M(X)	MBR ← M(X)
		If MBR $\neq$ 0
		ALU ← MBR*
		ALU ← AC*
		MQ ← Quotient
		AC ← Remainder
		*ALU inputs both AC, MBR
		Otherwise
		Halt the machine

18	LSH	ALU ← AC
		AC ← ALU

19	RSH	ALU ← AC
		AC ← ALU

S. No.	Instruction	Steps followed

20	STOR M(X,8:19)	MBR ← M(X)
		ALU ← MBR*
		ALU ← AC*
		MBR ← ALU**
		M(X) ← MBR
		*ALU inputs both AC, MBR
		**The new address, calculated by ALU, transfers to MBR

21	STOR M(X,28:39)	MBR ← M(X)
		ALU ← MBR*
		ALU ← AC*
		MBR ← ALU**
		M(X) ← MBR
		*ALU inputs both AC, MBR
		**The new address, calculated by ALU, transfers to MBR

22	HALT	Ends the program

The next page of this document describes the programs implemented in IAS machine ...

8 Programs Implemented in IAS Machine

8.1 Calculating sum of first 10 natural numbers

The data was stored at the following memory addresses:

Address	Data Stored	Comments
100	10	We first start adding number 10, then 9 and so on upto 1
101	1	This is used to decrement the number at Address 100
102	0	This address stores the result of the calculation

The following set of instructions were used to perform the calculation:

Address	LHS Instruction	RHS Instruction	Comments
0	LOAD M(102)	ADD M (100)	Since we are storing result at address 102, we first load it in AC, and then add the next natural number to be added to the sum.
1	STOR M(102)	LOAD M(100)	Since the result of the above operation is stored in AC, we first store the contents of AC into address 102, then we have to subtract the new number to be added. So we load it to AC.
2	SUB M(101)	STOR M(100)	We subtract 1 from the number to be added next, and store it back to address 100
3	JUMP+ M(0, 0:19)	HALT	As the number to be added next is still in AC, we check if the number is greater than or equal to 0, if it is, we again execute LHS instruction at address 0, else we halt the machine.

8.2 Calculating first 10 Fibonacci Numbers

The Fibonacci Sequence is a well known sequence that can be constructed as follows:

$$F(i) = \begin{cases} 0, & \text{if } i = 0 \\ 1, & \text{if } i = 1 \\ F(i-1) + F(i-2), & \text{if } i \geq 2 \end{cases}$$

The data was stored at the following memory addresses:

Address	Data Stored	Comments
100	0	This is the first number of the Fibonacci Sequence. This address holds $F(i-2)$ for future calculations.
101	1	This is the second number of the Fibonacci Sequence. This address holds $F(i-1)$ for future calculations
102	7	As we have to calculate 10 Fibonacci numbers, out of which 2 have already been calculated (i.e. 0 & 1), we have to calculate 8 more Fibonacci numbers. When the number at this address becomes negative (i.e. -1), the program will stop, as from 7 to 0, we would have calculated 8 Fibonacci numbers.
103	1	Each time a number in the sequence has been calculated, we would need to decrement the number in address 102, this 1 is used for decrementing the number at address 102.
104	202	Stores the address of the location at which the next Fibonacci number will be stored.
200 - 209	See Comments	Fibonacci Numbers are stored at these memory locations. The first Fibonacci Number will be at address 200, and the last one will be at address 209.

The following set of instructions were used to perform the calculation:

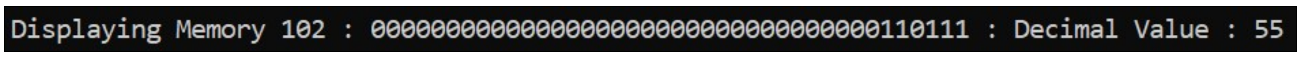
Address	LHS Instruction	RHS Instruction	Comments
0	LOAD M(100)	STOR M(200)	We load the first number of the sequence into AC and store it in address 200
1	LOAD M(101)	STOR M(201)	We then load the second number of the sequence into AC and store it in address 201
2	LOAD M(102)	JUMP+ (4, 0:19)	We now have to check if we have to calculate more Fibonacci numbers or not. For that, we load the number at address 102, which tells us if there are more numbers that we've to calculate. If it is greater than or equal to 0, we proceed to memory location 4
3		HALT	If we have no more numbers to calculate, we halt the machine
4	LOAD MQ M(101)	LOAD M(100)	Since we have more numbers to calculate, we load $F(i-1)$ to MQ and load $F(i-1)$ to AC
5	ADD M(101)	STOR M(202)	As AC contains $F(i-2)$, we add $F(i-1)$ to AC, which gives us $F(i)$ and store the result at its desired location. Please note that the number 202 in the RHS instruction will be later modified by the address modify instructions to point to the address at which we are going to store the next Fibonacci Number.

This table is continued at the next page...

Address	LHS Instruction	RHS Instruction	Comments
6	STOR M(101)	LOAD M(Q)	For calculating the next number i.e. $F(i+1)$, we will need the value of $F(i)$. We already have $F(i)$ in AC, so we store the value of AC (i.e. $F(i)$) at address 101, as it contains the term $F(j)$ for calculating $F(j+1)$ We also load contents of MQ (which had $F(i-1)$) to AC.
7	STOR M(100)	LOAD M(102)	AC now contains $F(i-1)$. As $F(i+1) = F(i) + F((i+1)-2)$, we store $F(i-1)$ to the location containing $F(j-2)$ which is used for calculating $F(j)$. Also, we need to decrement the number of Fibonacci numbers left to calculate, so we load it into AC.
8	SUB M(103)	STOR M(102)	We decrement the number of Fibonacci Numbers left to calculate, and put it back to address 102.
9	LOAD M(104)	ADD M(103)	We need to increment the address for storing the next number. So, we load the number at address 104 (which stores the address at which next Fibonacci Number will be stored), and increment it.
10	STOR M(104)	STOR M(5, 28:39)	We store this new address at address 104, and modify the RHS instruction of address 5, so that it points to the address for storing the next number
11		JUMP M(2, 0:19)	We go back to our original loop

9 Result of Execution of Programs

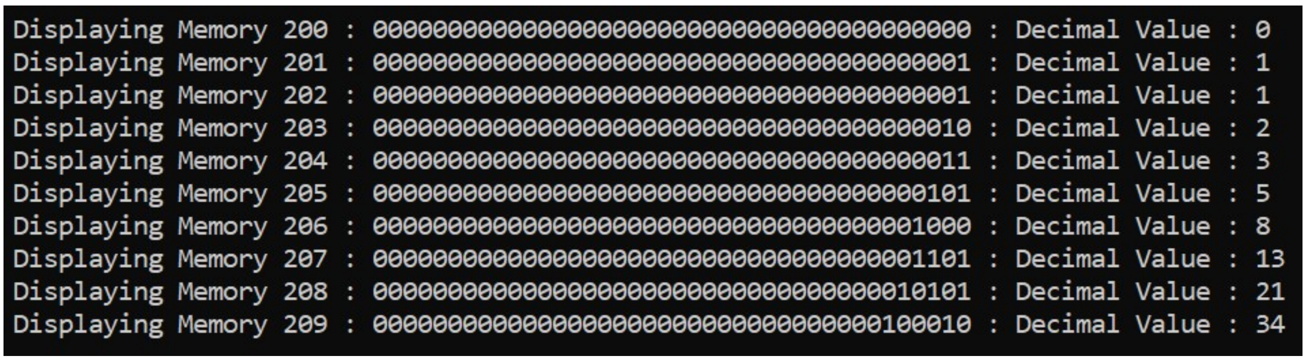
9.1 Result of the first Program: calculating sum of first 10 natural numbers



```
Displaying Memory 102 : 0000000000000000000000000000000000110111 : Decimal Value : 55
```

Figure 1: Result of program for calculating Sum of first 10 natural numbers

9.2 Result of the second program : calculating Fibonacci numbers



```
Displaying Memory 200 : 0000000000000000000000000000000000000000 : Decimal Value : 0
Displaying Memory 201 : 0000000000000000000000000000000000000001 : Decimal Value : 1
Displaying Memory 202 : 0000000000000000000000000000000000000001 : Decimal Value : 1
Displaying Memory 203 : 0000000000000000000000000000000000000010 : Decimal Value : 2
Displaying Memory 204 : 0000000000000000000000000000000000000011 : Decimal Value : 3
Displaying Memory 205 : 0000000000000000000000000000000000000101 : Decimal Value : 5
Displaying Memory 206 : 0000000000000000000000000000000000001000 : Decimal Value : 8
Displaying Memory 207 : 0000000000000000000000000000000000001101 : Decimal Value : 13
Displaying Memory 208 : 0000000000000000000000000000000000010101 : Decimal Value : 21
Displaying Memory 209 : 0000000000000000000000000000000000100010 : Decimal Value : 34
```

Figure 2: Result of program for calculating Fibonacci Numbers

10 References and Acknowledgements

It would have been impossible for me to program the IAS Computer without the help and support I received from **Prof. Nanditha Rao** and **TA's**.

Also, I would like to mention the following resources from the Internet, which helped me to better understand the underlying principles of IAS Computer:

1. Simplified IAS Language: Link
2. Basic Concepts and Computer Evolution: Slide 16 for an updated picture of the IAS machine: Link
3. Paper on IAS machine for providing useful insights: Link