# Computer Architecture Assignment-1

Mannat Kaur Bagga(IMT2023071),Kavya Gupta(IMT2023016),Ananya Vundavalli(IMT2023537)

26 January 2024

## 1 Introduction:

In this project, we have written a program which checks whether the number entered by the user is an Armstrong Number or not. We have implemented this using the IAS instruction set architecture and have used the instructions which are required to implement our program. Also, we have added 2 instructions of our choice, the POWER instruction and the LEN instruction. The program prints 1 if the number is an Armstrong number, and prints 0 if not. The high level language program is written in C programming language , and the assembler and processor are implemented using python programming language. We have also attached the assembly language program.

## 2 What are Armstrong Numbers?

An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits. For example, the number 153 is an Armstrong number because $1^3 + 5^3 + 3^3 = 153$

## 3 C Program:

```c
#include <stdio.h>
int x_power_y(int x,int y){
	int result=1;
	for (int i=0;i<y;i++){
		result=result*x;
	}
	return result;
}
int len(int n){
	//finding the number of digits in the number
	int count=0;	//the final value of count will be the number of digits
	while (n!=0){
		n=n/10;
		count++;
	}
	return count;
}

int armstrong(int n){
	//finding the sum when each digit is raised to the power of the number of digits in the number
	int count=len(n);
	int sum=0;
	int temp=n;
	while (temp!=0){
		int units=temp%10;
		sum=sum+(x_power_y(units,count));
		temp=temp/10;
	}

	if (sum==n){
		return 1;
	}

	else{
		return 0;
	}

}
void main(){
	int n;
	printf("Enter a number:");
	scanf("%d",&n);
	printf("Final result (1: if Armstrong number, 0: if not)\n");
	printf("%d\n",armstrong(n));
}
```

Figure 1: C Program

# 4 Assembly Program:

LEN instruction: It calculates the number of digits in the number entered. Length is calculated as len(str(AC)) [we have used the len function for strings in python, so convert the number into a string and use the len function]. The length of the number is then stored in AC.

POWER instruction: The extracted digits would be raised to the power of the number of digits in the original number.This computation can be performed using the IAS machine's arithmetic capabilities. The MBR takes the value of the length(M[2]). The value in AC is raised to the power of the value in MBR.

```
M={n,10,l,s}                           # Here n is the number given, l is the length of the number and s is the final sum.

                        memory address
LOAD M(0) LEN            //0           # It takes n and stores in AC. Then it finds the length of the number and stores in AC.
STOR M(2) LOAD MQ,M(0)   //1           # Length is stored in M(2)=l. Then it stores n in MQ.

                                       # Loop starts

LOAD MQ DIV M(1)         //2           # Loads MQ to AC and divides the number by 10 (in M(1)). The remainder is stored in AC and quotient is in MQ.
POWER (M(2)) ADD M(3)    //3           # Gives value in AC to the power l. Then adds it to the value in (M(3)=s).
STOR M(3)                //4           # the value in AC is stored in M(3).
JUMP m(2,0:19)           //5           # Goes back to the memory location 2.
HALT
```

Figure 2: Assembly Program

# 5 Assembler:

Assembler converts the instructions in the Assembly Program to Binary. To do so we have used a bunch of if-else statements. The add0 function here just ensures that the address is 12 bits long.

```python
def op(ins):
    if ins == "LOAD":
        return "00000001"
    elif ins == "STOR":
        return "00100001"
    elif ins == "DIV":
        return "00001100"
    elif ins == "ADD":
        return "00000101"
    elif ins == "JUMP":
        return "00001101"
    elif ins == "POWER":
        return "00100011"
    elif ins == "LEN":
        return "01100011"
    elif ins == "LOAD MQ,M(0)":
        return "00001001"
    elif ins == "LOAD MQ":
        return "00001010"
    else:
        return "0"

def binary(n):
    if n == 0:
        return "0"
    elif n == 1:
        return "1"
    else:
        if n % 2 == 0:
            return binary(n // 2) + "0"
        else:
            return binary(n // 2) + "1"
```

Figure 3: Assembler 1

```python
def add0(a):
    n = 12 - len(a)
    return '0' * n + a

def Input(m):
    print("Assembly Program: ")
    while True:
        a = input()
        if a == "HALT":
            break
        else:
            y = ""
            strbin = ""

            if "JUMP" in a:
                y = op("JUMP")
                strbin = binary(int(a[7]))
                strbin = add0(strbin)
                y += strbin
                y = "0"*20 + y
            elif "STOR" in a:
                y = op("STOR")
                strbin = binary(int(a[7]))
                strbin = add0(strbin)
                y += strbin
                if "LOAD" in a:
                    y += op("LOAD MQ,M(0)")
                    strbin = binary(int(a[20]))
                    strbin = add0(strbin)
                    y += strbin
                else:
                    y = "0"*20 + y
```

Figure 4: Assembler 2

```
        elif "LEN" in a:
            y = op("LOAD")
            strbin = binary(int(a[7]))
            strbin = add0(strbin)
            y += strbin
            y += op("LEN")
            y += "0"*12

        elif "LOAD MQ" in a:
            y = op("LOAD MQ")
            y += "0"*12
            y += op("DIV")
            strbin = binary(int(a[14]))
            strbin = add0(strbin)
            y += strbin

        elif "POWER" in a:
            y = op("POWER")
            strbin = binary(int(a[9]))
            strbin = add0(strbin)
            y += strbin
            y += op("ADD")
            strbin = binary(int(a[19]))
            strbin = add0(strbin)
            y += strbin

        m.append(y)
    print("Binary :")
    for j in m:
        print(j)
```

Figure 5: Assembler 3

# 6 Processor:

This program contains functions to decode and process the information (in binary) stored in the memory. The decoder splits the binary to form Left Instruction and Right instruction, and further splits the instructions to it's respective opcode and address. Then the processor takes the opcode and address, and performs their respective operations. The final result is stored in M[3].

```
from ca_assembler_final import *

MAR, IR, PC, IBR = "", "", 0, ""
AC, MQ, MBR, M = 0, 1, 0, [1, 10, 0, 0]
memory = []
def processor(op, addr):
    global AC, MQ, MBR
    addr= "0b"+addr
    if op == "00000001":#load
        MBR = M[int(addr, 2)]
        AC = MBR
        print("load----------------------")
        print("AC: ", AC)

    elif op == "00100001":#stor
        MBR = AC
        M[int(addr, 2)] = MBR
        print("stor----------------------")
        print("AC: ", AC)
    elif op == "00000101":#add
        MBR = M[int(addr, 2)]
        AC = AC + MBR
        print("add----------------------")
        print("AC: ", AC)
    elif op == "00001100":#div
        MBR = M[int(addr, 2)]
        MQ = AC // MBR
        AC = AC % MBR
        print("div----------------------")
        print("AC: ", AC)
        print("MQ: ", MQ)
```

Figure 6: Processor 1

```python
    elif op == "00001001":#load mq m(0)
        MQ=M[int(addr, 2)]
        print("load mq m(0)----------------------")
        print("MQ: ",MQ)
    elif op == "00001010":#Load mq
        AC = MQ
        print("load mq----------------------")
        print("AC: ", AC)

    elif op == "00100011":#power
        MBR = M[int(addr, 2)]
        AC = AC**MBR
        print("power----------------------")
        print("AC: ", AC)

    elif op == "01100011":#Len
        AC=len(str(AC))
        print("len----------------------")
        print("AC: ", AC)

    elif op == "00001101":#jump
        print("jump----------------------")
        PC = int(addr,2)
        print("PC: ",PC)
```

Figure 7: Processor 2

```python
def decoder(inst):
    global MAR, IR, PC, IBR
    LHS, RHS, Lop, Rop, Laddr, Raddr = "", "", "", "", "", ""
    MAR = PC

    LHS = inst[0:20]
    Lop = LHS[0:8]
    Laddr = LHS[8:20]

    RHS = inst[20:40]
    Rop = RHS[0:8]
    Raddr = RHS[8:20]
    MAR,IBR,IR =Laddr,RHS,Lop
    processor(Lop, Laddr)
    PC+=1

    MAR, IR = Raddr, Rop
    processor(Rop, Raddr)


Input(memory)
n = int(input("Enter a number: "))
M[0] = n

decoder(memory[0])
decoder(memory[1])
```

Figure 8: Processor 3

```
N = 2
count=0
while count<len(str(n)) :
    decoder(memory[N])
    if "00001101" in memory[N]:
            N = 2
            count+=1
    else:
        N += 1
print("Final result (1: if Armstrong number, 0: if not)")
if M[3]==M[0]:
    print(1)
else:
    print(0)
```

Figure 9: Processor 4

# 7 Result:

```
kavya@kavya-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Documents/sem2_CA/ca_final_v2/ca_final$ gcc ca_c_final.c
kavya@kavya-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Documents/sem2_CA/ca_final_v2/ca_final$ ./a.out
Enter a number:153
Final result (1: if Armstrong number, 0: if not)
1
kavya@kavya-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Documents/sem2_CA/ca_final_v2/ca_final$ ./a.out
Enter a number:37
Final result (1: if Armstrong number, 0: if not)
0
```

Figure 10: Result 1

```
kavya@kavya-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Documents/sem2_CA/ca_final_v2/ca_final$ python3 ca_processor_final.py
Assembly Program:
LOAD M(0) LEN
STOR M(2) LOAD MQ,M(0)
LOAD MQ DIV M(1)
POWER (M(2)) ADD M(3)
STOR M(3)
JUMP m(3,0:19)
HALT
Binary :
0000000100000000000001100011000000000000
0010000100000000001000001001000000000000
0000101000000000000000001100000000000001
0010001100000000001000000101000000000011
0000000000000000000000100001000000000011
0000000000000000000000001101000000000011
```

Figure 11: Result 2

For input number = 153.

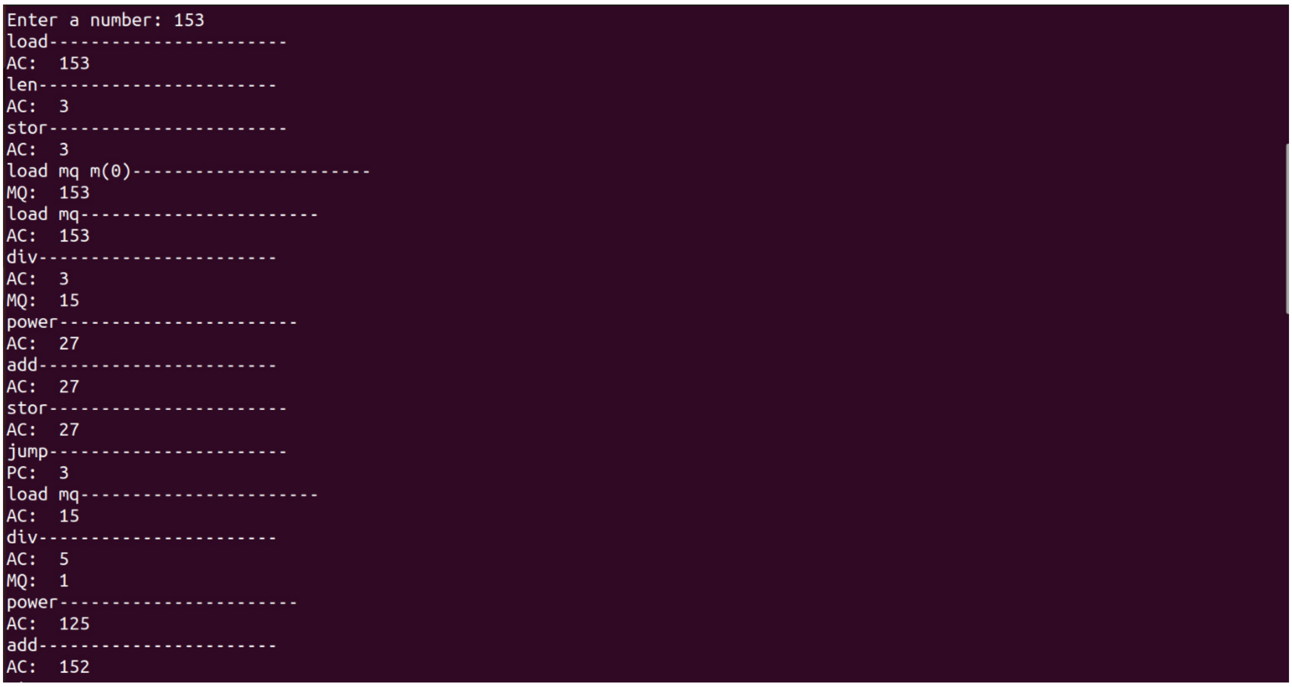
```
Enter a number: 153
load----------------------
AC:  153
len----------------------
AC:  3
stor----------------------
AC:  3
load mq m(0)----------------------
MQ:  153
load mq----------------------
AC:  153
div----------------------
AC:  3
MQ:  15
power----------------------
AC:  27
add----------------------
AC:  27
stor----------------------
AC:  27
jump----------------------
PC:  3
load mq----------------------
AC:  15
div----------------------
AC:  5
MQ:  1
power----------------------
AC:  125
add----------------------
AC:  152
```

Figure 12: Result 3

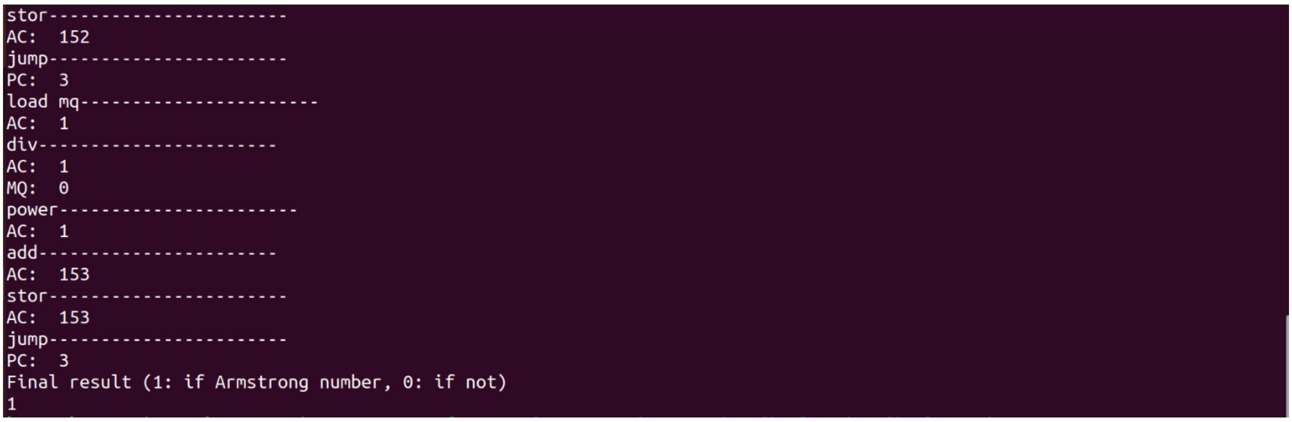
```
stor----------------------
AC:  152
jump----------------------
PC:  3
load mq----------------------
AC:  1
div----------------------
AC:  1
MQ:  0
power----------------------
AC:  1
add----------------------
AC:  153
stor----------------------
AC:  153
jump----------------------
PC:  3
Final result (1: if Armstrong number, 0: if not)
1
```

Figure 13: Result 4

For input number = 37.

```
Enter a number: 37
load----------------------
AC:  37
len----------------------
AC:  2
stor----------------------
AC:  2
load mq m(0)----------------------
MQ:  37
load mq----------------------
AC:  37
div----------------------
AC:  7
MQ:  3
power----------------------
AC:  49
add----------------------
AC:  49
stor----------------------
AC:  49
jump----------------------
PC:  3
load mq----------------------
AC:  3
div----------------------
AC:  3
MQ:  0
power----------------------
AC:  9
add----------------------
AC:  58
```

Figure 14: Result 5

```
stor----------------------
AC:  58
jump----------------------
PC:  3
Final result (1: if Armstrong number, 0: if not)
kavya@kavya-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Documents/sem2_CA/ca_final_v2/ca_final$
```

Figure 15: Result 6