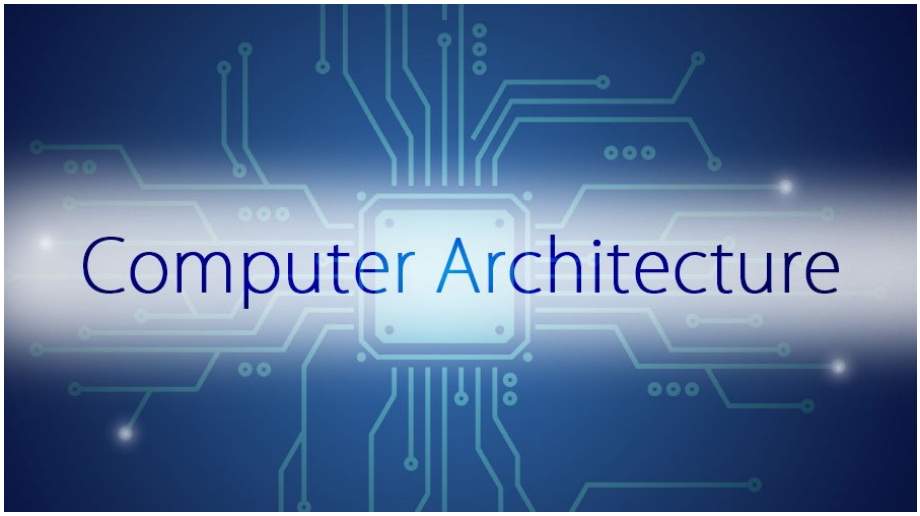
Computer Architecture and Organization Project

Two-Pass Assembler and Accumulator based CPU Design



SUBMITTED BY
Gaurav Singhal 2019UCO1571
Mayank Goel 2019UCO1558
Sandeep Jain 2019UCO1522

Nishant Goel 2019UCO1529

DESCRIPTION:

Assembly language is a low-level programming language which is converted into machine code with the help of an assembler program. We have designed a **Two -Pass Assembler** which translates the assembly program into its equivalent binary code and stores it in the memory. The CPU hence can access these instructions from the memory and process them to produce the desired output.

Language used to code the assembler: C++

MEMORY:

The size of the memory is **128* 16** bits. Memory is **byte addressable**.

CPU Design

Address bus: 7 bits

Data bus: 16 bits

Flags:

Zero flag: This flag is set if the data in accumulator is zero, else clear.

Carry flag: this flag is set if there is a carry bit

Sign flag: this flag is set if the value in AC is negative.

Overflow flag: this flag is set if the value in AC is overflowing

Instructions:

Size: 16 bit

Operations: 20 distinct operations

INSTRUCTION FORMAT

- Size of each instruction is 16 bits.
- First 2 bits are always 00.
- 3rd Bit is use for determining the type of instruction as Memory reference or Non-memory reference.
- 4th bit is used for determining the mode of instruction as direct or indirect
- Next 5 bits are reserved for operation code.
- Remaining bits are reserved for address.

16-Bit Instruction Format

00	MRI or Non-MRI (1-bit)	Mode Bit (1-bit)	Opcode(5-bit)	Address(7-bit)

INSTRUCTION SET

Sno	mnemonic	opcode	Register Transfer Language (RTL)	Function	Example
1	ldan	00001	AC<- M[xxxxxxx]	Load AC with data present at given address from memory	ldan ads
2	cman	00010	AC<- AC’	Complement the value stored at AC	cman
3	incn	00011	AC<- AC +1	Increment the value stored in AC by 1	incn
4	addn	00100	AC <- AC+M[xxxxxxx]	Add value stored at AC with data given by given address	addn ptr I
5	stan	00101	M[xxxxxxx]<-AC	Store value given by AC at the given address in memory	stan ptr

6	bunn	00110	PC<-AR	Branch unconditionally	bunn 15
7	clan	00111	AC <-0	Set data present in AC to 0	clan
8	iszn	01000	DR<-M[AR],DR<-DR+1,M[AR]<-DR,if(DR=0) then PC<-PC+1	Increment and skip if zero	iszn ptr
9	andn	01001	AC <-(AC&M[xxxxxxx])	Logical and operation between AC and the value present at given address	andn x
10	orn	01010	AC <-(AC|M[xxxxxxx])	Perform bitwise OR	orn x
11	xorn	01011	AC <-(AC^M[xxxxxxx])	Perform bitwise xor operation on AC with value stored at given address	xorn x
12	setn	01100	AC <-(AC&(1<<(M[xxxxxxx]-1)))	Set the ith bit in AC to 1 (Zero-based indexing)	setn i
13	lsln	01101	AC<-lsln(AC)	Logical left shift the value stored in AC	lsln x
14	lsrn	01110	AC<-lsln(AC)	Logical right shift the value stored in AC	lsrn x
15	asln	01111	AC<-lsln(AC)	Arithmetic left shift the value stored in AC	asln x
16	asrn	10000	AC<-lsln(AC)	Arithmetic right shift the value stored in AC	asrn x
17	csln	10001	AC<-lsln(AC)	Circular left shift the value stored in AC	csln x
18	csrn	10010	AC<-lsln(AC)	Circular right shift the value stored in AC	csrn x
19	subn	10011	AC<- AC-M[xxxxxxx]	Subtract value stored at given address from AC	subn ptr

20	decn	10100	AC<- AC-1	Decrement the value stored at AC by 1	decn

Pseudo-Instructions:

- hlt
- org
- end

Internal processor register:

Program counter: 7 bits

Address register: 7 bits

Instruction register: 5 bits

Data register: 16 bits

Types of Instructions:

- **Memory Reference Instructions**
  These instructions refer to memory address as an operand. The other operand is always accumulator.
- **Non-Memory Reference Instructions**
  These instructions does not require any address location for retrieval of data , they do not operate with memory .

Addressing Modes:

- **Direct Addressing Mode**

Address part of the instruction contains the address of the operand.

- **Indirect Addressing Mode**
  Address part of the instruction contains the address of the address where operand is stored.

How does the assembler works?

1. Assembler performs its action in two phases.
2. In first pass, Assembler retrieves all the data from symbolic addresses and prepares an address symbol table in the memory.
3. In second pass , Assembler converts each instruction line by line into its binary equivalent and replaces symbols with their equivalent data from address symbol table.
4. Assembler stops converting on reaching halt condition.

What does your assembler returns?

Our assembler returns the memory of size 128*16 having the instruction in machine level language that can be fed to the CPU of above mentioned specifications.

At each step our assembler shows the content of each register used in CPU design.

So, user can check the step by step execution of the instructions of the assembly program.

There is a separate space where complete Equivalent binary code is shown for the given

Assembly Program.

USER INTERFACE :-

For creating user interface we have used Qt software that uses C++ library to integrate our code with an interface to emulate the working of cpu.

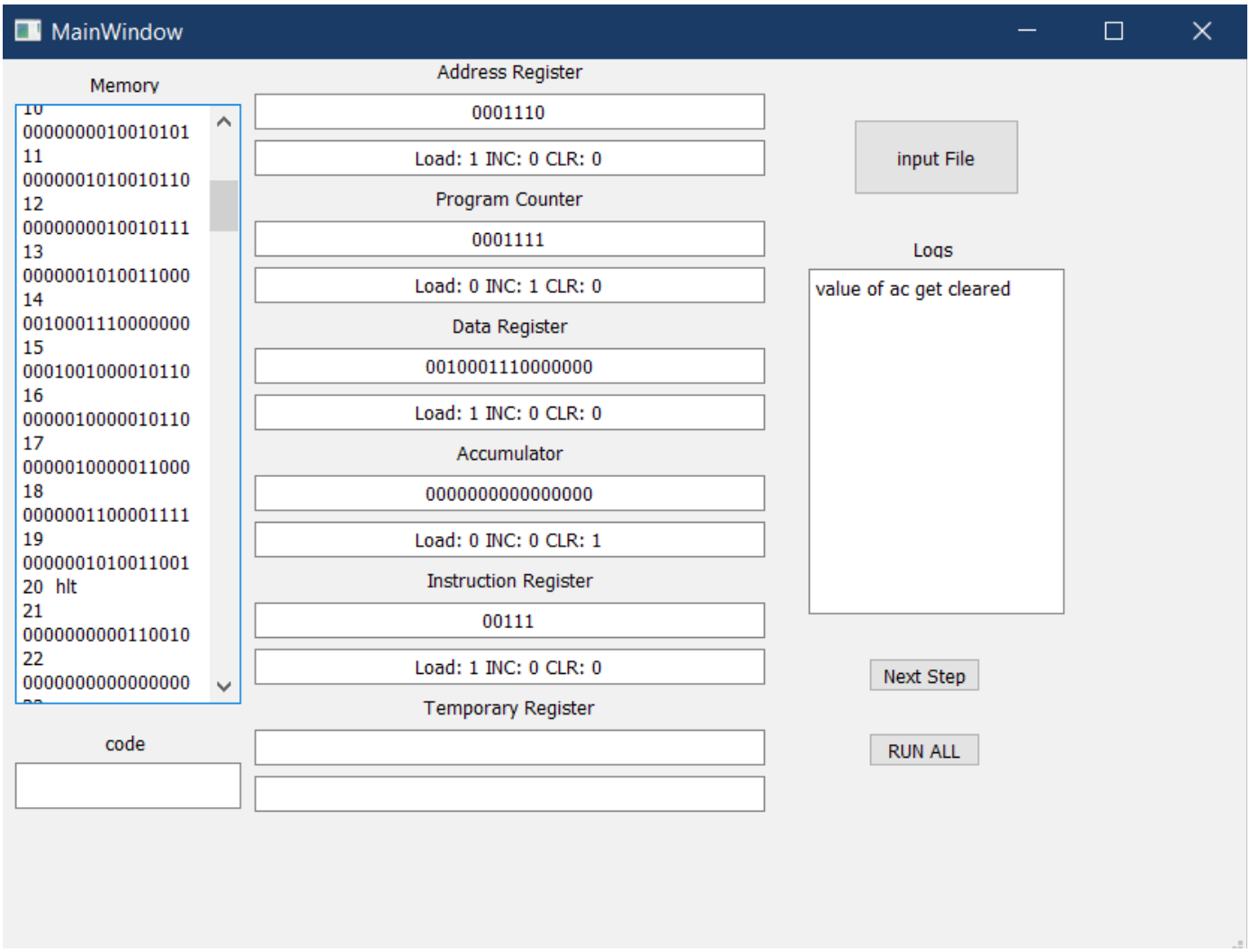
Inbuilt C++ library is provided by the Qt company which enabled us to display the output on a new screen.

This development environment at the same time created an equivalent application of the assembler.

For references for Qt , visit Qt documentation here: https://doc.qt.io/

SAMPLE RUN:-


sample.txt - Notepad
File Edit Format View Help
org 10
ldan ads
stan ptr
ldan nbr
stan ctr
clan
addn ptr I
iszn ptr
iszn ctr
bunn 15
stan sum
hlt
ads, hex 50
ptr, hex 0
nbr, dec -5
ctr, hex 1
sum, hex 2
org 50
dec 75
dec 25
dec 15
dec 10
dec 5
end
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8


MainWindow
Memory
0000000010010101
11
0000001010010110
12
0000000010010111
13
0000001010011000
14
0010001110000000
15
0001001000010110
16
0000010000010110
17
0000010000011000
18
0000001100001111
19
0000001010011001
20 hlt
21
0000000000110010
22
0000000000000000
code
Address Register
0001110
Load: 1 INC: 0 CLR: 0
Program Counter
0001111
Load: 0 INC: 1 CLR: 0
Data Register
0010001110000000
Load: 1 INC: 0 CLR: 0
Accumulator
0000000000000000
Load: 0 INC: 0 CLR: 1
Instruction Register
00111
Load: 1 INC: 0 CLR: 0
Temporary Register
input File
Logs
value of ac get cleared
Next Step
RUN ALL