Elements of Computing

Assembly program to define an array with N numbers stored in an array which down counts to N

Deepthi Sudharsan (CB.EN.U4AIE19022)
Isha Indhu S (CB.EN.U4AIE19030)
Jayashree O (CB.EN.U4AIE19031)
Meghna Menon (CB.EN.U4AIE19043)

Table of Contents

Acknowledgment

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We are highly indebted to Mr Vijay Krishna Menon and Dr Govind for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

Objective

To write an assembly program to define an array with N numbers stored in an array which down counts to N

Introduction

Every computer has its own microprocessor that will maintain its arithmetical, logical and control activities. Each kind of processor has its own set of instructions for taking care of various operations such as getting input from the user through the keyboard, displaying information on screen and many other tasks. These set of instructions are known as 'machine language instructions'.

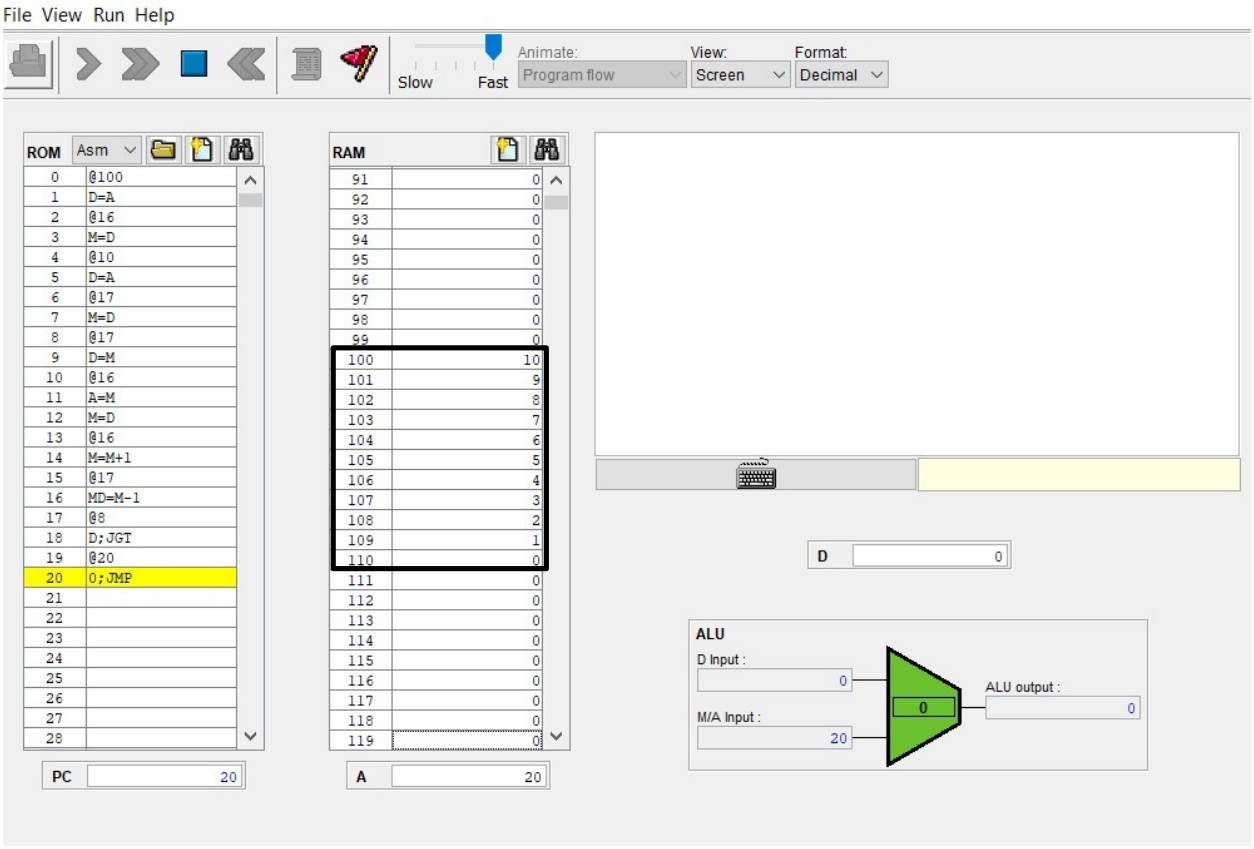
A microprocessor can grasp only machine language instructions which are binary values '1' and '0'.But,unfortunately the machine language is too difficult to comprehend and for the usage of software development. Hence, the low level assembly language is designed for a specific kind of processors that represent different instructions in symbolic code and a more sensible form.

PART 1

Assembly Language Code

```
@100
D=A
@arr
M=D

@10
D=A
@n
M=D
(LOOP)
@n
D=M
@arr
A=M
M=D

@arr
M=M+1

@n
MD=M-1

@LOOP
D;JGT

@END
(END)
0;JMP
```

Output



In the above obtained output, the program down counts from 10 to 0 and stores the values at array locations starting from 100 in RAM.

PART 2

Assembler Code Explanation:

Inside Main Function:

```
if __name__ == "__main__":
    afile = "C://Isha//AI//semester 1//2019 AI//GOVIND SIR//New folder//DOWNCOUNT
    hfile=afile.replace(".asm",".hack");
    hf=open(hfile,"w")
    af=open(afile,"r")
    for line in af:
        if line.strip():
            if not line.startswith("//"):
                label(line.strip());
    for line in open(afile,"r"):
        #print(line)
        if line.strip():
            if not line.startswith("//"):
                s = InstructionType(line.strip());
                #print(s)
                if(InstructionType(line.strip())=="A"):
                    A(line.strip());
                if(InstructionType(line.strip())=="C"):
                    C(line.strip());
```

afile: input asm file
hfile: creates new hack file
opens both files

for loop :
reads entire asm file:
ignoring the comments :
performs function label

another for loop:
checks(Function(InstructionType)):
if(return of (InstructionType) – A):
performs function A
if(return of (InstructionType) – C):
performs function C

Global Variables Declared:

```
import re

Labels={}
Variables={}

dest = {'M': '001', 'D': '010', 'MD': '011', 'A': '100', 'AM': '101',
               'AD': '110', 'AMD': '111'}

cmp={'0':'0101010','1':'0111111','-1': '0111010', 'D': '0001100',
               'A': '0110000', '!D': '0001101', '!A': '0110001', '-D': '0001111',
               '-A': '0110011', 'D+1': '0011111', 'A+1': '0110111',
               'D-1': '0001110', 'A-1': '0110010', 'D+A': '0000010',
               'D-A': '0010011', 'A-D': '0000111', 'D&A': '0000000',
               'D|A': '0010101', 'M': '1110000', '!M': '1110001', '-M': '1110011'
               'M+1': '1110111', 'M-1': '1110010', 'D+M': '1000010',
               'D-M': '1010011', 'M-D': '1000111', 'D&M': '1000000',
               'D|M': '1010101'}
jmp = {'JGT': '001', 'JEQ': '010', 'JGE': '011', 'JLT': '100', 'JNE': '101',
        'JLE': '110', 'JMP': '111'}

Symbols = {'R0': 0, 'R1': 1, 'R2': 2, 'R3': 3, 'R4': 4, 'R5': 5,
                  'R6': 6, 'R7': 7, 'R8': 8, 'R9': 9, 'R10': 10, 'R11': 11,
                  'R12': 12, 'R13': 13, 'R14': 14, 'R15': 15, 'SP': 0,
                  'LCL': 1, 'ARG': 2, 'THIS': 3, 'THAT': 4, 'SCREEN': 16384,
                  'KBD': 24576}
varcount=16
linecount=0
```

dictionary Labels – to store label names and a integer value associated to it
dictionary Variables – to store variables and a integer value associated to it

dictionary dest - to store the corresponding three destination bits

dest	d1	d2	d3
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

dictionary jmp – to store the corresponding three jump bits

jump	j1	j2	j3
null	0	0	0
JGT	0	0	1
JEQ	0	1	0
JGE	0	1	1
JLT	1	0	0
JNE	1	0	1
JLE	1	1	0
JMP	1	1	1

dictionary cmp – to store the corresponding six computation bits

comp (when a=0)	c1	c2	c3	c4	c5	c6	*comp* (when a=1)
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Function label:

declares global labels
declares global linecount
ignores spaces
splits the line according to special characters - ins
if (ins(1) is ‘(‘):
    and if (ins(2) is digit):
     nothing happens
if (ins(2) is already not in Labels:
     new label is added and is mapped to linecount
linecount is incremented

```
def label(line):
    global labels
    global linecount
    line = line.replace(" ","")
    ins = re.split('(\(|\)|=|;|@|,|/)', line)
    if ins[1] == '(':
        if ins[2].isdigit():
            return
        if ins[2] not in Labels:
            Labels[ins[2]] = linecount
        return
    linecount += 1
```

Function InstructionType:

ignores spaces
splits special characters and line – comd
if comd(1) is ‘(‘:
  its label
if comd(1) is ‘@’:
    it’s A instruction
else :
 its C instruction

```
def InstructionType(line):
    line=line.replace(" ","")
    comd=re.split('(\(|\)|=|;|@|,|/)',line)
    if comd[1]=='(':
        return "L"
    elif comd[1]=="@":
        return "A"
    else:
        return "C"
```

Function A:

```
def A(line):
    global varcount
    global Variables
    line=line.replace(" ","")
    ins=re.split('(@)',line)
    #print(ins)
    if ins[2].isdigit():
        hf.write('0' + format(int(ins[2]), '015b'))
        #print('0' + format(int(ins[2]), '015b'))
        hf.write("\n");
        return
    if ins[2] in Labels:
        hf.write('0'+ format(int(Labels[ins[2]]), '015b'))
        #print('0'+ format(int(Labels[ins[2]]), '015b'))
        hf.write("\n");
        return
    if ins[2] in Symbols:
        hf.write('0'+ format(Symbols[ins[2]], '015b'))
        #print('0'+ format(int(Symbols[ins[2]]), '015b'))
        hf.write("\n");
        return
    if ins[2] in Variables:
            hf.write('0' + format(Variables[ins[2]], '015b'))
            #print('0' + format(Variables[ins[2]], '015b'))
            hf.write("\n");
            return
    Variables[ins[2]]=varcount;
    hf.write('0' + format(varcount, '015b'))
    hf.write("\n");
    varcount+= 1
    return
```

declares global varcount - integer variable count

declares global Variables - list of variable names
ignores spaces and splits @ , remaining line - ins

if Condition1 – checks if ins(2) is digit:
concatenates **0** and **15 bit binary equivalent of digit**
writes above string in hackfile
if Condition1 – checks if ins(2) is in Labels:
concatenates **0** and **15 bit binary equivalent of integer mapped to Label**
writes above string in hackfile
if Condition1 – checks if ins(2) is in Symbols:
concatenates **0** and **15 bit binary equivalent of integer mapped to Symbol**
writes above string in hackfile
if Condition1 – checks if ins(2) is in Variables:
concatenates **0** and **15 bit binary equivalent of integer mapped to variable**
writes above string in hackfile
else
add ins as new variable to list Variables and is mapped to varcount
concatenates **0** and **15 bit binary equivalent of integer mapped to variable**
writes above string in hackfile
varcount is incremented

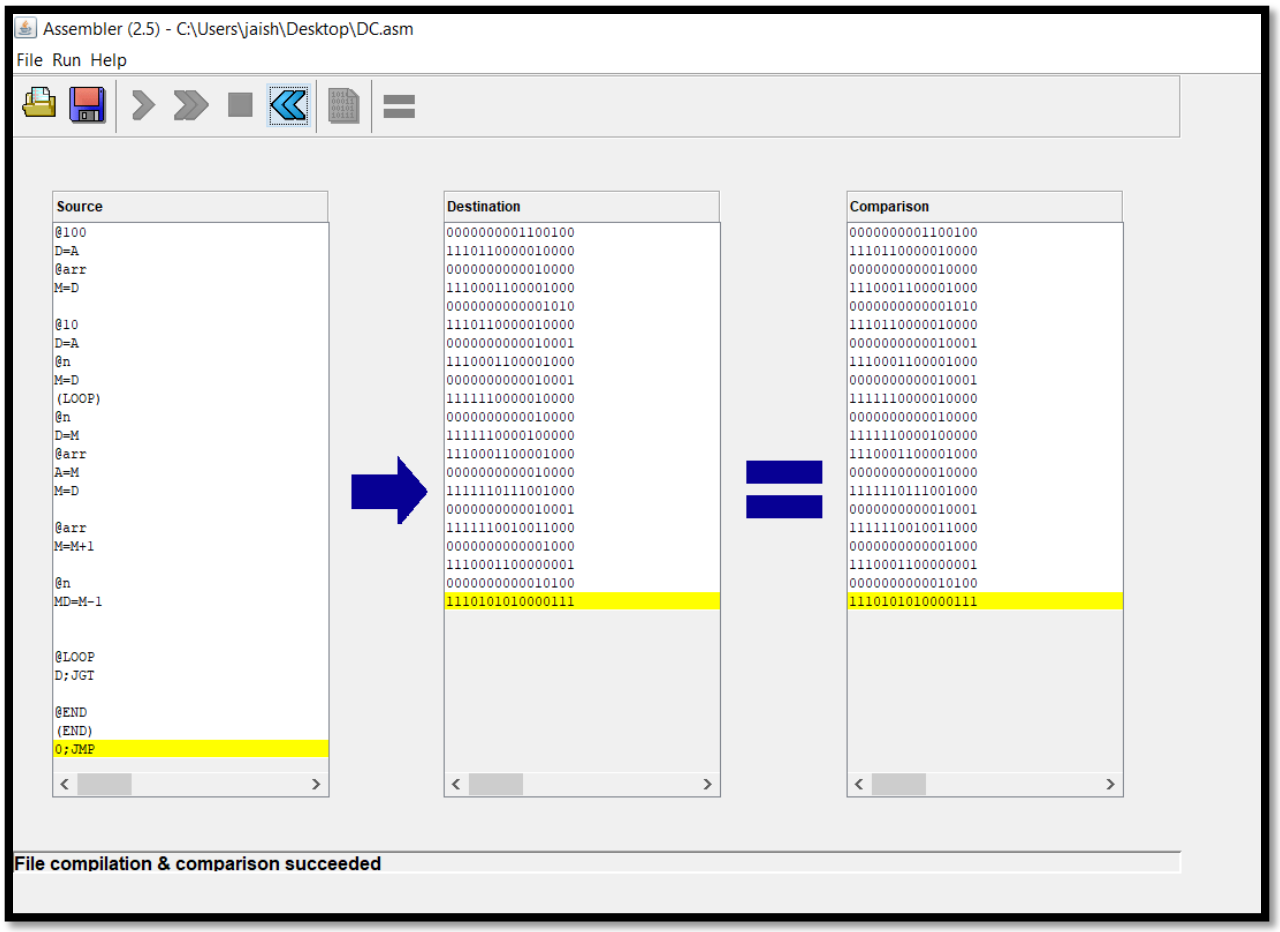
Function C:

```
def C(line):
    line=line.replace(" ","")
    ins=re.split('(\(|\)|=|;|@|,|/)',line)
    if ins[1] == '=':
        hf.write('111' + cmp[ins[2]] + dest[ins[0]] + '000')
        #print('111' + cmp[ins[2]] + dest[ins[0]] + '000')
        hf.write("\n");
        return
    if ins[1] == ';':
        hf.write('111' +  cmp[ins[0]] + '000' + jmp[ins[2]])
        #print('111' +  cmp[ins[0]] + '000' + jmp[ins[2]])
        hf.write("\n");
        return
    if ins[1] == ',':
        hf.write('111' +  cmp[ins[0]] + '000' + jmp[ins[2]])
        #print('111' +  c[ins[0]] + '000' + j[ins[2]])
        hf.write("\n");
        return
```

ignores spaces
splits special characters and line - ins
if(ins(1) is ‘=’)
concatenates **111** and **mapped value of cmp[ins(2)]** and **mapped value ofdest[ins(0)]** and **000**
writes above string in hackfile
if(ins(1) is ‘;’)
concatenates **111** and **mapped value of cmp[ins(0)]** and **000** and**mapped value ofjmp[ins(2)]**
writes above string in hackfile
if(ins(1) is ‘,’)
concatenates **111** and **mapped value of cmp[ins(0)]** and **000** and **mapped value ofjmp[ins(2)]**
writes above string in hackfile

OUTPUT OF PYTHON CODE:

```
0000000001100100
1110110000010000
0000000000010000
1110001100001000
0000000000001010
1110110000010000
0000000000010001
1110001100001000
0000000000010001
1111110000010000
0000000000010000
1111110000100000
1110001100001000
0000000000010000
1111110111001000
0000000000010001
1111110010011000
0000000000001000
1110001100000001
0000000000010100
1110101010000111
```



The above figure shows the output and verification of the hack file obtained from the assembler code**.**

PART 3

Computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. It represents the means of interconnectivity between the computer's hardware components and the mode of data transfer and processing exhibited. Different computer architecture configurations have been developed to speed up the movement of data, allowing for increased data processing. The basic architecture has the CPU at the core with a main memory and input/output system on either side of the CPU.

Computer architecture consists of three main categories.

- System design – This includes all the hardware parts, such as CPU, data processors, multiprocessors, memory controllers and direct memory access. This part is the actual computer system.
- Instruction set architecture – This includes the CPU's functions and capabilities, the CPU's programming language, data formats, processor register types and instructions

used by computer programmers. This part is the software that makes it run, such as Windows or Photoshop or similar programs.

- Micro Architecture – This defines the data processing and storage element or data paths and how they should be implemented into the instruction set architecture.

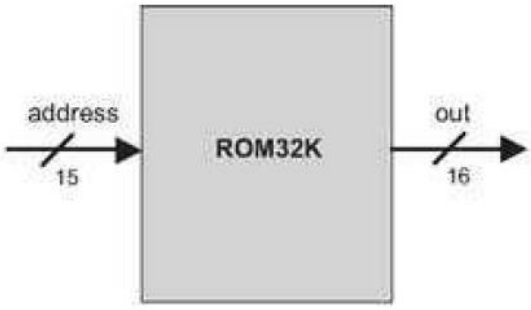
The computer architecture consists of three main components mainly:

- Instruction Memory (ROM32K)
- CPU
- Data Memory

To use a human analogy, the CPU is your brain, the RAM is your short-term memory, and the ROM is your long-term memory. CPU is large and in-charge. The CPU (Central Processing Unit) is, in fact, the logical centre of a computer.

Instruction Memory

Architecture uses separate memory for instruction and data. Instruction memory is read-only; a programmer cannot write into the instruction memory. Instructions tell the programme what to do and are usually stored in the text portion of the binary.

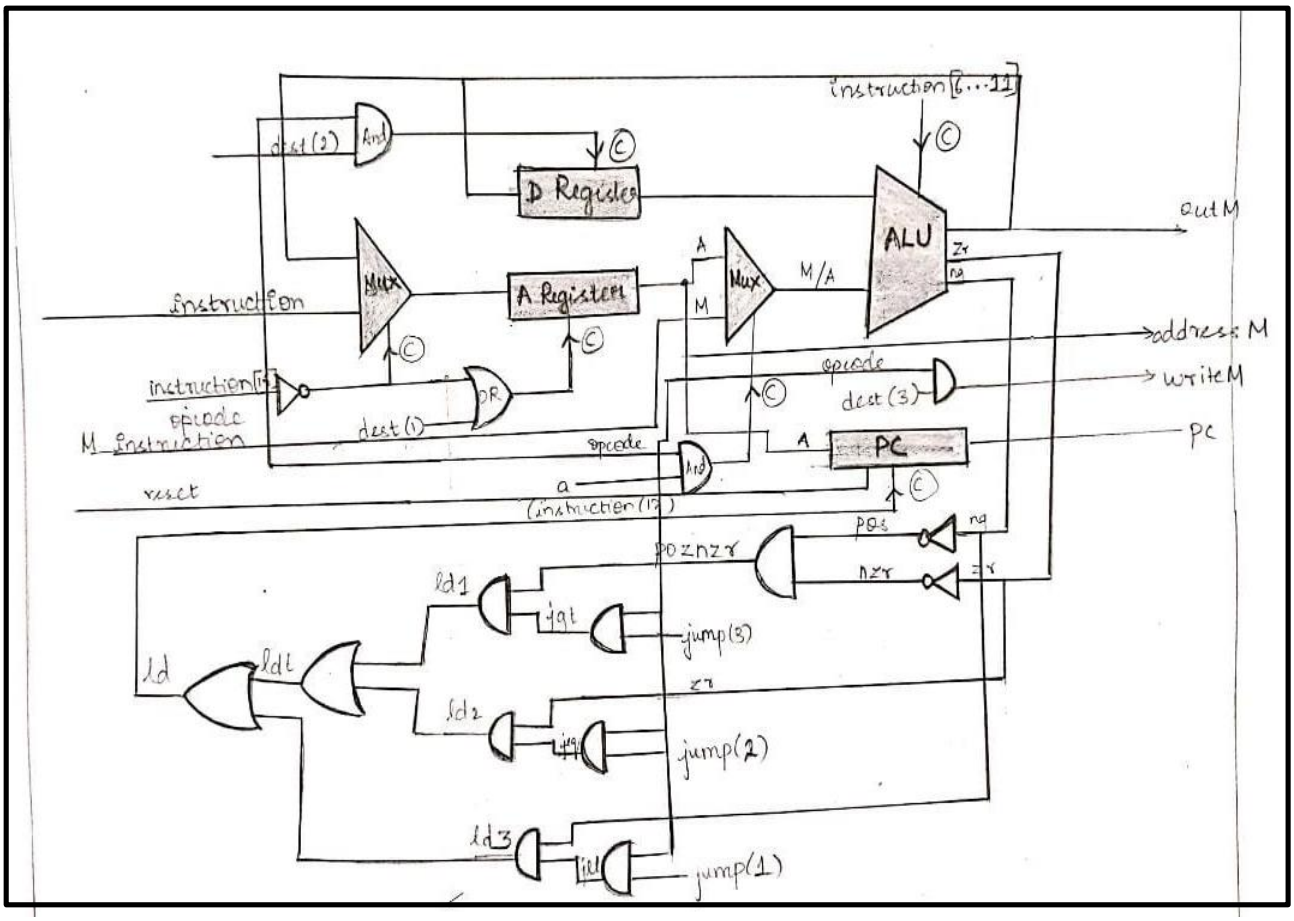



```
CHIP InstructionMemory {
   IN ad[15];
   OUT output[16];

   PARTS:

   ROM32K(address=ad,out=output);

}
```

The chip name is ROM32K. It is a 16 bit read only 32k memory. It takes address (in the ROM) as its input and the value of the ROM (address) as its output. The ROM is preloaded with a machine language program. The hardware implementations can treat the ROM as a built in chip. The software simulators must supply a mechanism for loading a program into the ROM

Central Processing Unit (CPU)

The objective of the CPU implementation is to create logic gate architecture capable of executing a given Hack instruction and getting the next instruction to be executed. Naturally, the CPU will include an ALU capable of executing Hack instructions, a set of registers, and some control logic designed to fetch and decode instructions.



The figure above shows the architecture that has been used to build the CPU chip.

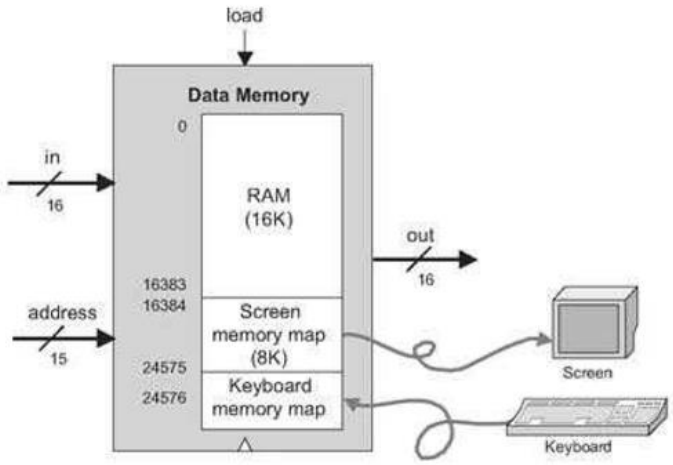
```
CHIP CPU {

  IN  inM[16],     // M value input  (M = contents of RAM[A])
      instruction[16], // Instruction for execution
      reset;           // Signals whether to re-start the current
                       // program (reset=1) or continue executing
                       // the current program (reset=0).

  OUT outM[16],       // M value output
      writeM,         // Write into M?
      addressM[15],   // Address in data memory (of M)
      pc[15];         // address of next instruction

  PARTS:
            Not(in=instruction[15],out=ni);
            Mux16(a=outtM,b=instruction,sel=ni,out=i);

            Or(a=ni,b=instruction[5],out=intoA);
            ARegister(in=i,load=intoA,out=A,out[0..14]=addressM);

            And(a=instruction[15],b=instruction[12],out=AorM);
            Mux16(a=A,b=inM,sel=AorM,out=AM);

            ALU(x=D,y=AM,zx=instruction[11],nx=instruction[10],zy=instruction[9],ny=instruction[8],f=instruction[7],no=instruction[6],out=outtM,out=outM,zr=zr,ng=ng);

            And(a=instruction[15],b=instruction[4],out=intoD);
            DRegister(in=outtM,load=intoD,out=D);

            And(a=instruction[15],b=instruction[3],out=writeM);

            Not(in=ng,out=pos);
            Not(in=zr,out=nzr);
            And(a=instruction[15],b=instruction[0],out=jgt);
            And(a=pos,b=nzr,out=posnzr);
            And(a=jgt,b=posnzr,out=ld1);

            And(a=instruction[15],b=instruction[1],out=jeq);
            And(a=jeq,b=zr,out=ld2);

            And(a=instruction[15],b=instruction[2],out=jlt);
            And(a=jlt,b=ng,out=ld3);

            Or(a=ld1,b=ld2,out=ldt);
            Or(a=ld3,b=ldt,out=ld);

            PC(in=A,load=ld,inc=true,reset=reset,out[0..14]=pc);

}
```

Central Processing Unit (CPU) consists of the following features:

- CPU is considered as the brain of the computer.
- CPU performs all types of data processing operations.
- It stores data, intermediate results, and instructions (program).
- It controls the operation of all parts of the computer.

Data Memory

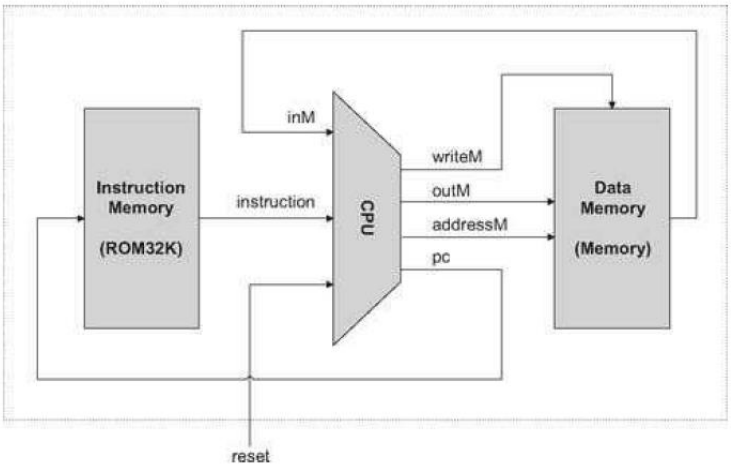
Memory architecture describes the methods used to implement electronic computer data storage in a manner that is a combination of the fastest, most reliable, most durable, and least expensive way to store and retrieve information. Data memory (RAM) is used for temporary storing and keeping the required data for the proper operation of the programs.




```
CHIP Memory {
IN in[16],load,address[15];
OUT out[16];

PARTS:
RAM16K(in=in,load=load,address=address[0..13],out=out);


}
```

Computer Chip

The architecture of the Hack computer is rather minimal. Typical computer platforms have more registers, more data types, more powerful ALUs, and richer instruction sets. However, these differences are mainly quantitative. From a qualitative standpoint, Hack is quite similar to most digital computers, as they all follow the same conceptual paradigm: the von Neumann architecture.




```
CHIP ComputerChip {
 IN reset;

PARTS:

ROM32K(address=pc,out=instruction);
CPU(inM=inM,instruction=instruction,reset=reset,outM=outM,writeM=writeM,addressM=addressM,pc=pc);
Memory(in=outM,load=writeM,address=addressM,out=inM);


}
```

How are Computer Chips designed?

The process of building transistors into a chip starts with a pure silicon wafer. It is then heated in a furnace to grow a thin layer of silicon dioxide on the top of the wafer. A light-sensitive photoresist polymer is then applied over the silicon dioxide. Each chip contains many transistors making up a processor. There can be tens of millions of transistors on one chip. These pieces are aligned together to create an electrical signal. Several chips are placed together with different amounts of memory storage space on them in a central processing unit.

Control Signals

A pulse or frequency of electricity or light that represents a control command as it travels over a network, a computer channel or wireless is known as control signal. In the data communications world, control signals typically travel the same path as the data either as separate packets or contained within the data packets.

Control signals go to three separate destinations:

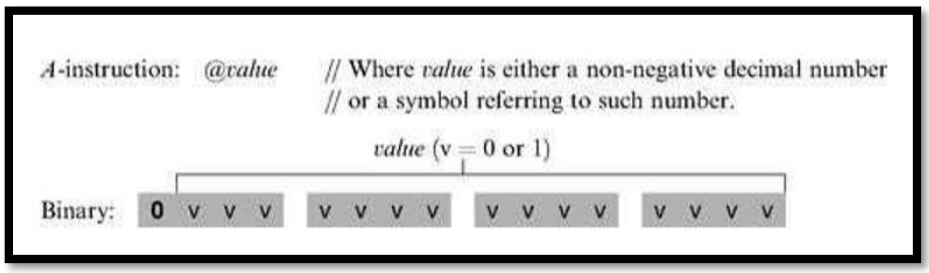
• Data paths: The control unit controls the internal flow of data. For example, on instruction fetch, the contents of the memory buffer register are transferred to the instruction register. For each path to be controlled, there is a switch (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.

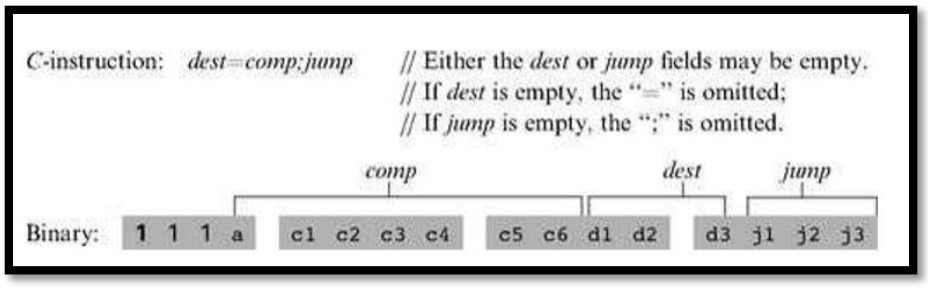
• ALU: The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic circuits and gates within the ALU.

• System bus: The control unit sends control signals out onto the control lines of the system bus (e.g., memory READ).

Components used in Central Processing unit(CPU):

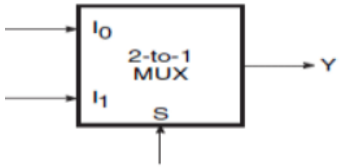
- MUX 1 (16 bit)
- A-register
- MUX 2(16 bit)
- ALU
- D register
- PC
- Control signals





MUX1:

Control signal	Negation of instruction[15]
0	Input A //output of ALU
1	Input B//instruction



(a)

S	Y
0	I_0
1	I_1

(b)

A-Register:

CONTROL SIGNAL (load)	Negation (instruction [15] i.e the opcode + instruction [5] i.e d1) • **d1 for A-instruction =1**
INPUT	Output from the MUX1
OUTPUT	The output of A-register is stored in addressM

MUX2:

Control signal	**AND OPERATION OF** instruction[15]-i.e the opcode and instruction[12]-i.e a • IF A =0 IT'S A, WHERE A IS THE ADDRESS • IF A=1 ITS M, WHERE M = CONTENTS OF RAM[A]
0	Input A // output from A-register
1	Input B //M value input (M = contents of RAM[A])

D-register:

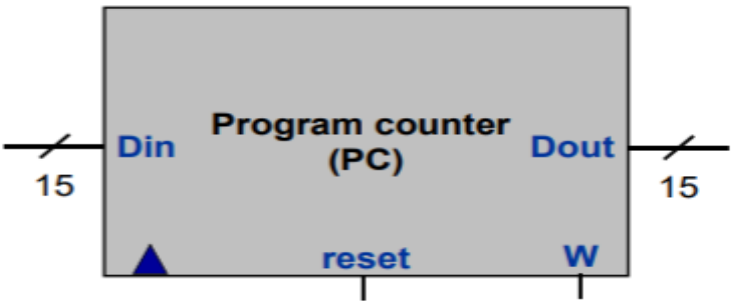
CONTROL SIGNAL (load)	**And operation of** Instruction[15]-i.e the opcodeand instruction[4]-i.e d2 • If d2=1 the instruction should enter D-register • If d2=0 the instruction shouldn't enter D-register
INPUT	Output of ALU
OUTPUT	Output of D-register, which acts as one of the input of ALU.

ALU:

(when a=0) comp mnemonic	c1	c2	c3	c4	c5	c6	(when a=1) comp mnemonic
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

INPUT	X[16]=Output of D-register
	Y[16]=Output of Mux2
	Zx(Zero the x input) =Instruction[11] (c1)
	nx(Negate the x input)= Instruction[10] (c2)
	zy(Zero the y input) = Instruction[9] (c3)
	ny(Negate the y input) = Instruction[8] (c4)
	f= Instruction[7] (c5)
	no= Instruction[6] (c6)
OUTPUT	Output of ALU is outM • outM=M value output
OUTPUT FLAG	zr, // 1 if out=0, 0 otherwise
	ng; // 1 if out<0, 0 otherwise

PC:




```
CHIP PC {

    IN  in[16], load, inc, reset;
    OUT out[16];

    BUILTIN PC;
    CLOCKED in, load, inc, reset;
}
```

Here t-1 refers to current time -1

If reset(t-1) then out(t) = 0
else if load(t-1) then out(t) = in(t-1)
else if inc(t-1) then out(t) = out(t-1) + 1 (integer addition)
else out(t) = out(t-1)

References

- https://computersciencewiki.org/index.php/Architecture_of_the_central_processing_unit_(CPU)
- https://quickcse.wordpress.com/category/computer-organization-architecture/
- https://www.sciencedirect.com/topics/computer-science/instruction-memory#:~:text=The%20instruction%20is%20read%20from,and%20the%20other%20containing%20data.
- https://www.tutorialspoint.com/assembly_programming/assembly_introduction.htm
- https://www.pcmag.com/encyclopedia/term/control-signal#:~:text=A%20pulse%20or%20frequency%20of,a%20computer%20channel%20or%20wireless.&text=In%20the%20traditional%20telephone%20communications,See%20signal%20and%20signaling.
- The Elements of Computing Systems –Noah Nisan and Shimon Schocken