CPU DESIGN

INTRODUCTION

Designing CPU from microarchitecture level and knowing the features about how their functionalities are implemented in hardware is just mesmerizing. The CPU (Central Processing Unit) often referred to as the brain of the computer, handling all instructions it receives from hardware and software running on the computer. It is just like how our brain controls our body and processes information. The CPU carries out various tasks by performing basic arithmetic, logic, control, and input/output (I/O) operations specified by the instructions in the program. Modern CPUs are complex devices that contain millions of transistors integrated on a single chip, capable of executing billions of instructions per second. This complexity allows them to perform a wide range of functions and support advanced computing capabilities. It consists of two units:

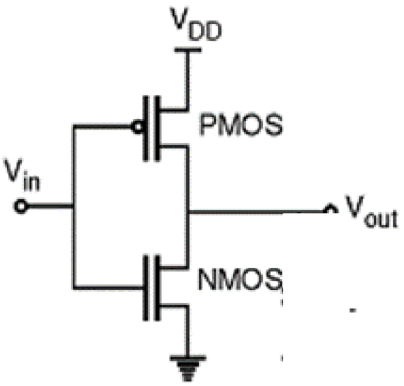
1. **Memory Unit:** It is a component of a computer that stores data and instructions for processing.
2. **Control Unit:** It directs the operation of the processor. It manages and coordinates all the activities within the computer.

Digital signals are preferred than analog because processing of noise is easy (e.g.: NOT gate).

CMOS INVERTER:

A Complementary metal oxide semiconductor (CMOS) inverter is a basic building block in digital electronics, primarily used in integrated circuits for creating complex digital circuits.

V_{DD}= 5V = logic 1 , Gnd = 0V = logic 0

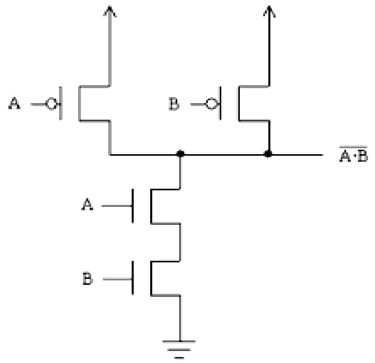


Input	PMOS transistor	NMOS transistor	Output
0	ON	OFF	1
1	OFF	ON	0

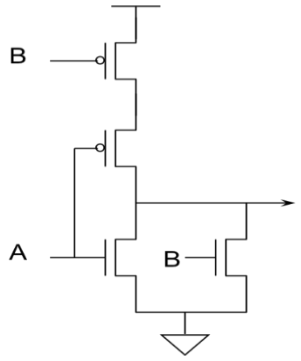
Although we know NAND and NOR gates are universal gates. Using these gates we can build all other elementary gates.

- NAND gates are used mostly than NOR gates.
  Reason: NAND gate using cmos acts as symmetric network. NOR gate using cmos acts as Asymmetric network.

Point to be noted, NMOS has lower resistance (R) where PMOS has higher resistance (2R). This is based on majority charge carriers.



NAND gate



NOR gate

In this article we basically understand the actual hardware.

- Implementation of combinational logic
- Implementation of ALU
- Implement of CPU & Memory
- Implementation of computer

COMBINATIONAL LOGICS:

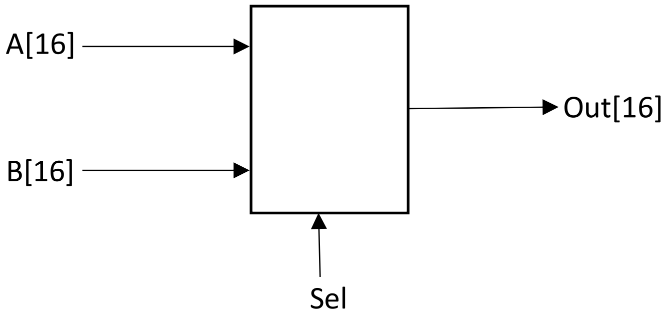
Combinational logic circuits like Multiplexer (MUX) and Demultiplexer (DMUX) are fundamental for analysing and implementation of hardware components.

A multiplexer (MUX) is a combinational circuit that selects one of many input signals and forwards the selected input to a single output line.

A demultiplexer (DMUX) is a combinational circuit that takes a single input signal and routes it to one of many output lines. It essentially performs the inverse operation of a multiplexer.

We are aware of using 1-bit inputs and outputs. Similarly for building a 16-bit processor we need 16-bit inputs and outputs. It is just by using 16-bit inputs instead of 1-bit inputs.

MUX16

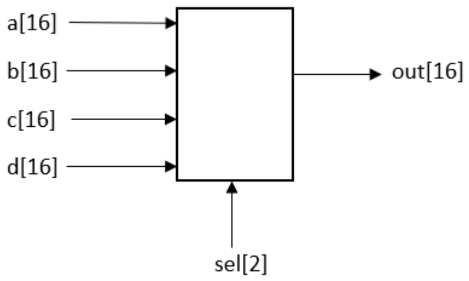


Or 8 way

It will add all the bits and verifies the presence of 1-bit logic. If the sum is 0 then there is no 1-bit. If the sum is not 0 then there is 1-bit.

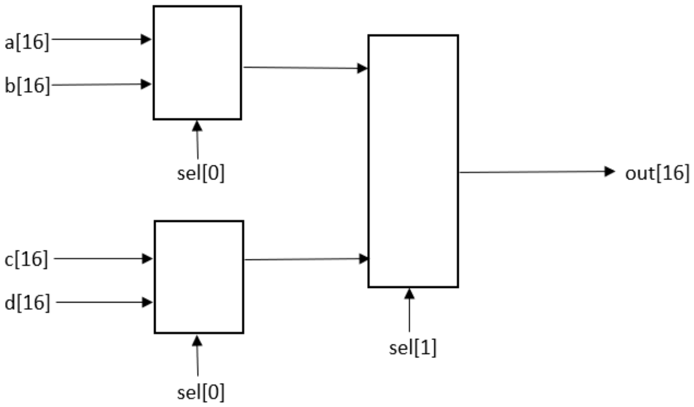
Syntax: in[0]+in[1]+in[2]+ in[3]+in[4]+in[5]+ in[6]+in[7]

Mux 4 way 16:

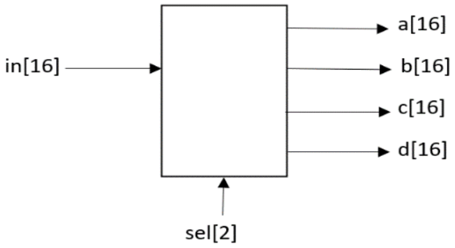


sel[1]	sel[0]	output
0	0	a[16]
0	1	b[16]
1	0	c[16]
1	1	d[16]

Its equivalent

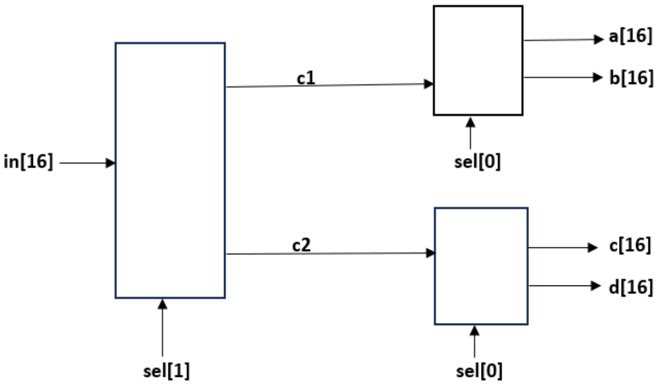


Dmux 4 way 16:



sel[1]	sel[0]	a[16]	b[16]	c[16]	d[16]
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Its equivalent



Inc 16

It indicates increment 16. It increments 16-bit value **‘a’** by 1.

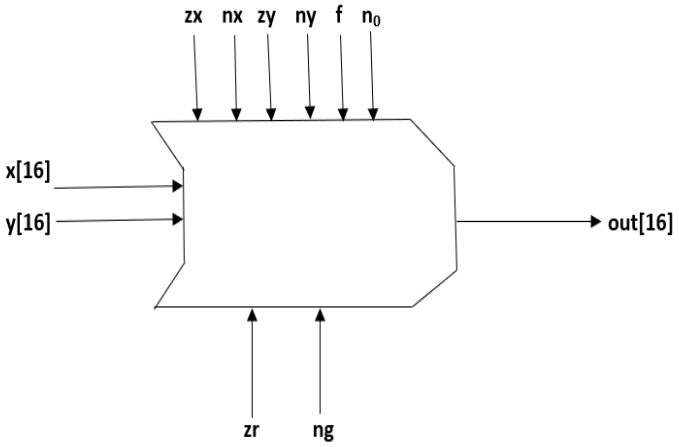
a[15] a[14] a[13] a[12] a[11] a[10] a[9] a[8] a[7] a[6] a[5] a[4] a[3] a[2] a[1] a[0] + 1

TYPES OF ARCHITECTURE:

- Harvard Architecture: Harvard Architecture is the digital computer architecture whose design is based on the concept where there are separate storage and separate buses (signal path) for instruction and data.
- Von neumann Architecture: Von Neumann Architecture is a digital computer architecture whose design is based on the concept of stored program computers where program data and instruction data are stored in the same memory.

ALU (ARITHMETIC LOGIC UNIT):

Arithmetic logic unit (**ALU**) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating point unit (FPU), which operates on floating point numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing unit (GPUs). The inputs to an ALU are the data to be operated on, called operands; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.



Input pins – x[16], y[16]

Output pins – out[16]

Select pins – zx, nx, zy, ny, f, n_0

Status pins – zr, ng

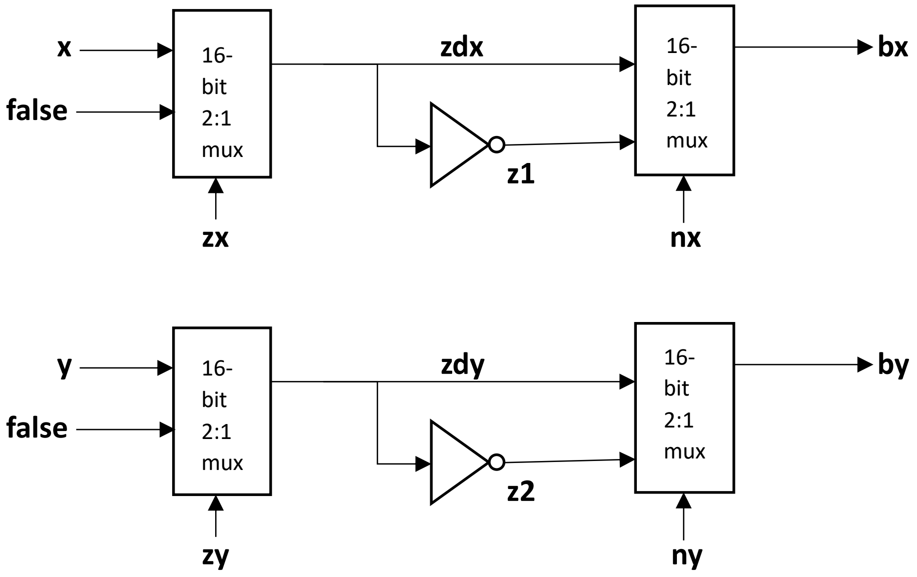
zr: if output is '0' then 'zr'=1, else 'zr'=0.

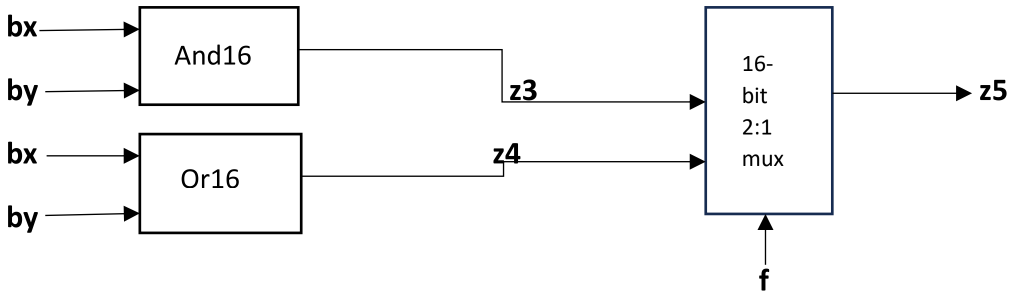
ng: if output is negative then 'ng'=1, else 'ng'=0.

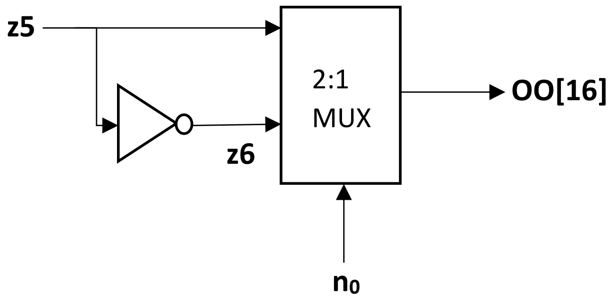
zx	**nx**	**zy**	**ny**	**f**	**n_0**	**out**
zx=1, x=0	nx=1, x=x!	zy=1, y=0	ny=1, y=y!	f=1, x+y (OR)	n_0=1, out=out!	output is obtained
zx=0, x=same	nx=0, x=same	zy=0, y=same	ny=0, y=same	f=0, x.y (AND)	n_0=0, out=same	

Pre-setting the 'x' input		***Pre-setting the 'y' input***		***Selecting between computing + or &***	***Post-setting the output***	***Resulting ALU output***
zx	**nx**	**zy**	**ny**	**f**	**n_0**	**out**
If 'zx' then x=0	**If 'nx' then x=x!**	**If 'zy' then y=0**	**If 'ny' then y=y!**	**If 'f' then out=x+y else out=x&y**	**If 'n_0' then out=out!**	**out(x,y)**
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

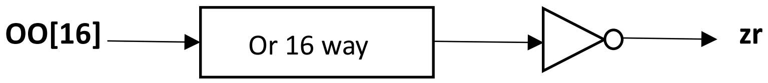
Internal architecture of ALU:



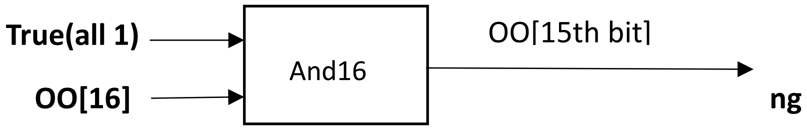


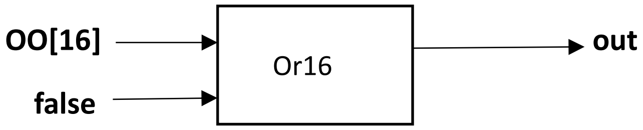


If OO[16] = 0 then zr=1 else zr=0.



If OO[16] < 0 then ng=1 ,else when OO[16] ≥ 0 then ng=0. This is done by checking the MSB bit. If MSB bit is 1 number is negative else positive.





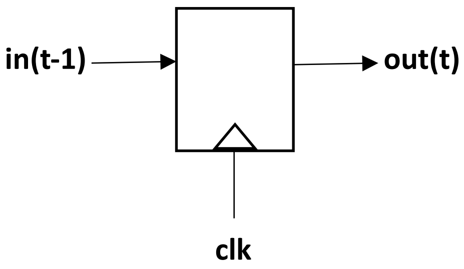
Finally output of ALU is obtained.

RAM (RANDOM ACCESS MEMORY):

Memory stores information, such as data and programs. Memory can be volatile and non-volatile. RAM(Random Access Memory) is a volatile memory, whenever power gets OFF the entire data will be erased. It can store data and instructions into the memory. We can read and write the data.

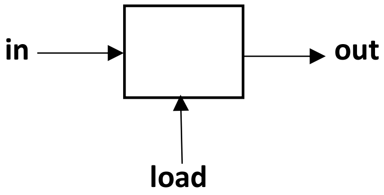
D – Flipflop is used as a storing element.

out(t) = in(t-1)



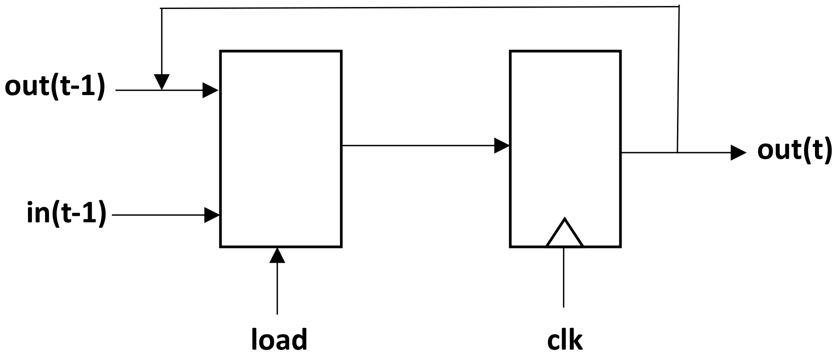
in(t-1)	out(t)
0	0
1	1

BIT:

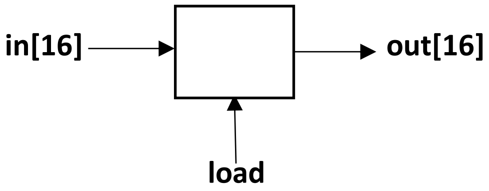


load(t-1)	out(t)
0	out(t-1)
1	in(t-1)

Load is having two conditions either 0 or 1.

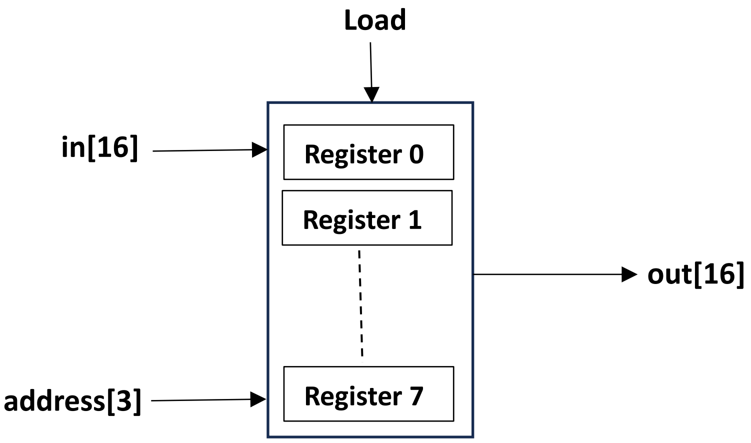


REGISTER: It is a device used to store information.



RAM8

(collection of registers)



If load=0 ,read value from register. If load=1 ,write value from register.

To read the data from any one of register, set desired address to address[3] pin.

Eg: Read data from Register-5.

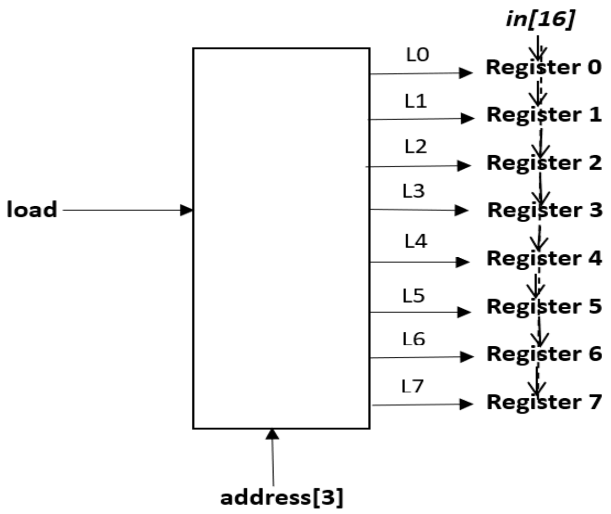
Give load=0 and set address[3]=101.Then out[16]= data from Register-5.

To write/set the data into RAM to a register.

Eg: Write data in Register-4.

Give load=1 and set address[3]=100.Then in[16]= set data.

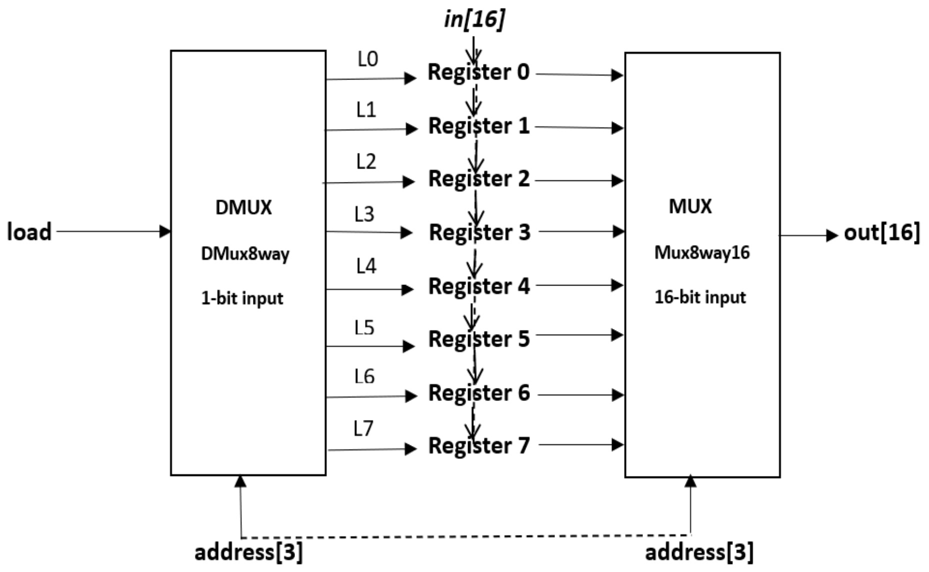
Internal structure of load pin:



Output of dmux is given to register as a load pin. Now by setting the address[3] pin we can activate the desired register load pin.

in[16] input of RAM8 is applied to all the registers. Because, it read/write based on load pin. So, at a time only one pin is activated such that pin read/write from in[16].

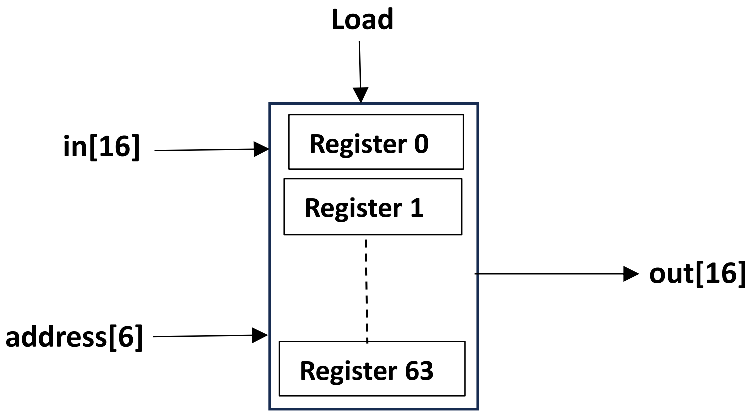
RAM8 Internal Architecture:



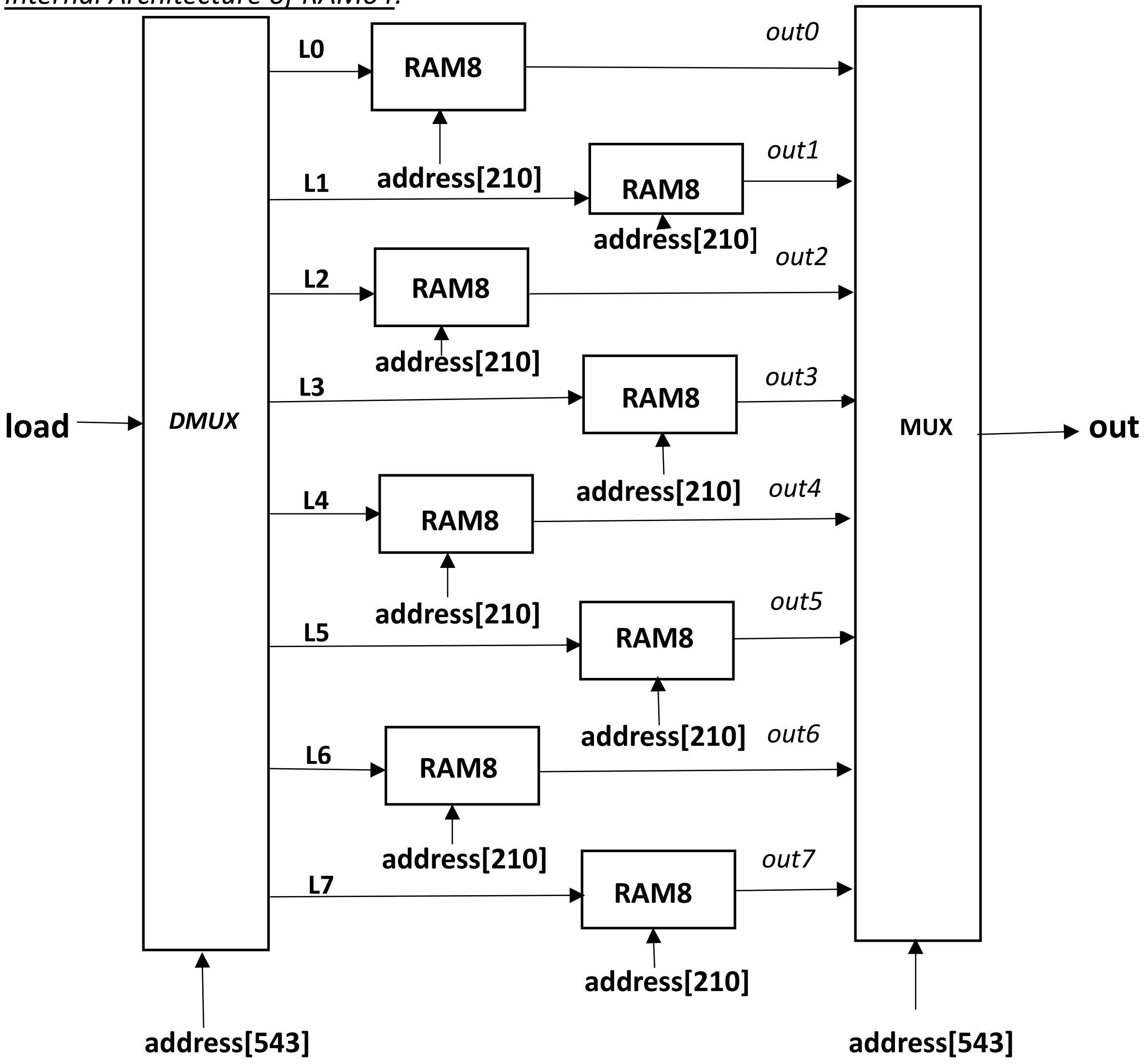
RAM64

(*8RAM X 8RAM = 64 Registers*), RAM64 has 6-bits address.

- 3-bits to activate – L0, L1, L2, L3, L4, L5, L6, L7. (address pins : 543)
- 3-bits to address pin for RAM. (address pins : 210)



Internal Architecture of RAM64:



address[5]	address[4]	address[3]	
0	0	0	L0
0	0	1	L1
0	1	0	L2
0	1	1	L3
1	0	0	L4
1	0	1	L5
1	1	0	L6
1	1	1	L7

RAM512

RAM512 has 9-bits address.

- 3-bits to activate – L0, L1, L2, L3, L4, L5, L6, L7.
- 6-bits to address pin for RAM.

RAM 4K

RAM 4K has 12-bits address.

- 3-bits to activate – L0, L1, L2, L3, L4, L5, L6, L7.
- 9-bits to address pin for RAM.

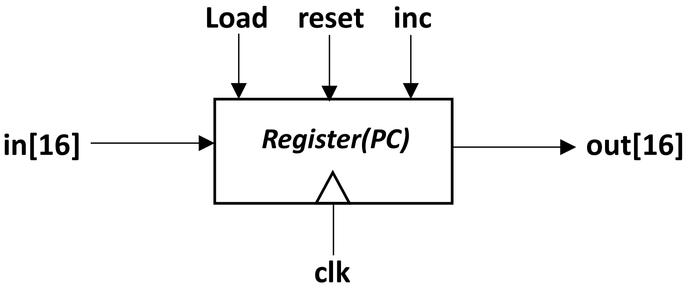
RAM 16K

RAM 16K has 14-bits address.

- 3-bits to activate – L0, L1, L2, L3, L4, L5, L6, L7.
- 12-bits to address pin for RAM.

PROGRAM COUNTER (PC):

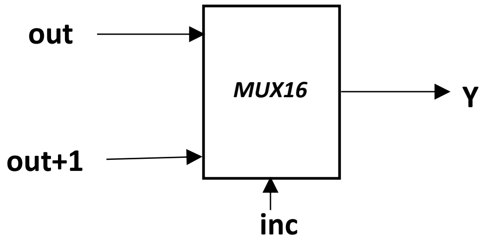
It counts the next instruction.



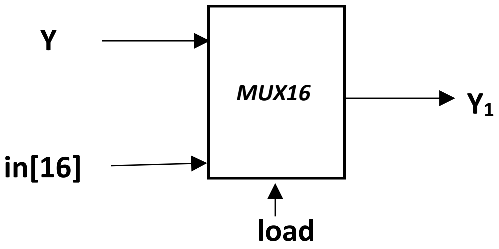
reset: When reset is ‘1’ it resets the PC, then output is ‘0’.

Inc: When inc is ‘1’ it increases the count by 1 i.e. increases output by 1.

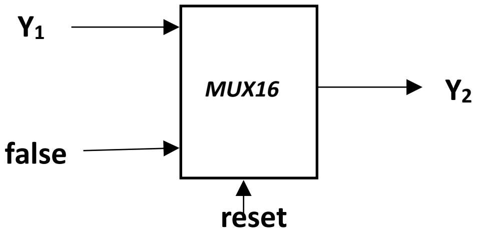
Load: When load is ‘1’ it jumps from particular/present value of output to destination point i.e. input goes to output. Else output is incremented by 1.



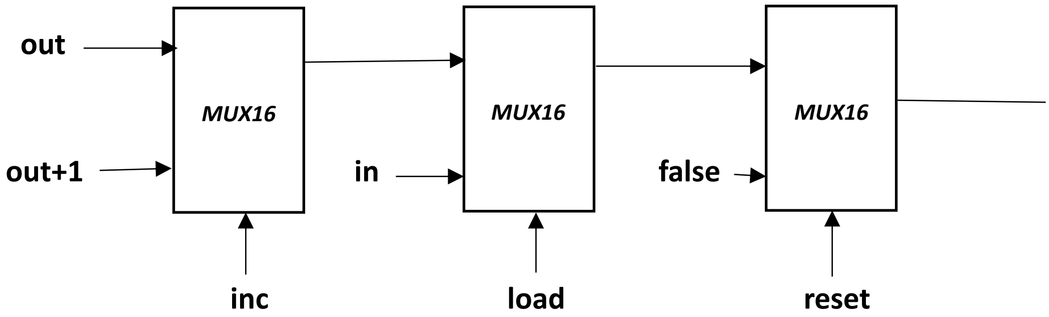
reset	Y_2
0	Y_1
1	false

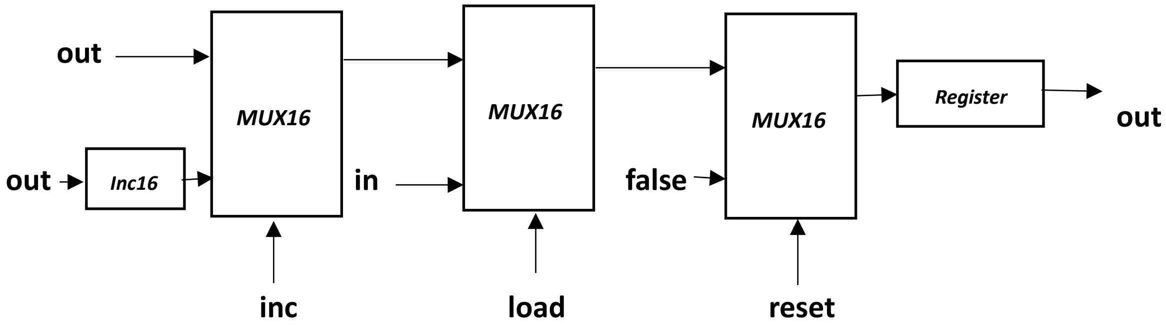


inc	Y
0	out
1	out+1



load	Y_1
0	Y
1	in[16]





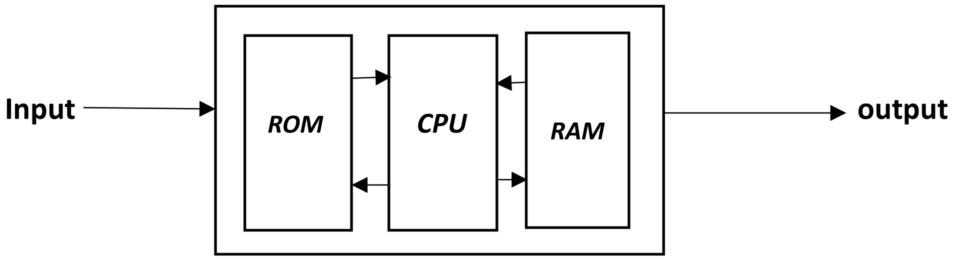
CPU (CENTRAL PROCESSING UNIT):

Instructions are stored in ROM and Data is stored in RAM.

We have 32,000 locations in ROM each location having unique instructions. Which location we want to choose is decided by the CPU (CPU→ROM) with the help of PC, then ROM delivers that particular instruction (ROM→CPU).

RAM is also having number of Registers, which register we want to use is also decided by CPU(CPU →RAM). Now RAM send that data to CPU (RAM →CPU).

Computer only understand the machine language.



ROM has 32K instructions.

TYPES OF INSTRUCTIONS:

1. A instruction
2. C instruction

15th bit = 0	A instruction
15th bit = 1	C instruction

A-instruction always defined first after that C-instruction is defined.

All these instructions are 16-bit.

A-INSTRUCTION:

- **0000000000010101** $A=(29)_D$
- 15th bit indicates type of instruction i.e. **‘0’**.
- 14th – 0th bits indicates value that has to be stored in A register.

Note:

In a processor we have to perform multiple operations for that we have PC, if we want all this in memory then it is waste of time (like in 8085 PC are is internally presented). So, to make it faster CPU itself having 2 Registers A&D.

Temporary data is stored in A & D Registers.

A-register tells the location of RAM. Instructions stored in ROM goes to the CPU , CPU collect that from A-register then it convert to know it's value (29) then it goes to that location in RAM for either read/write the data which is present inside the 29th location.

A=29 ;   CPU → M = RAM [29]

C – INSTRUCTION:

This is for **ALU activation.**

ALU is having 3 input registers. They are A-register, D-register and RAM register.

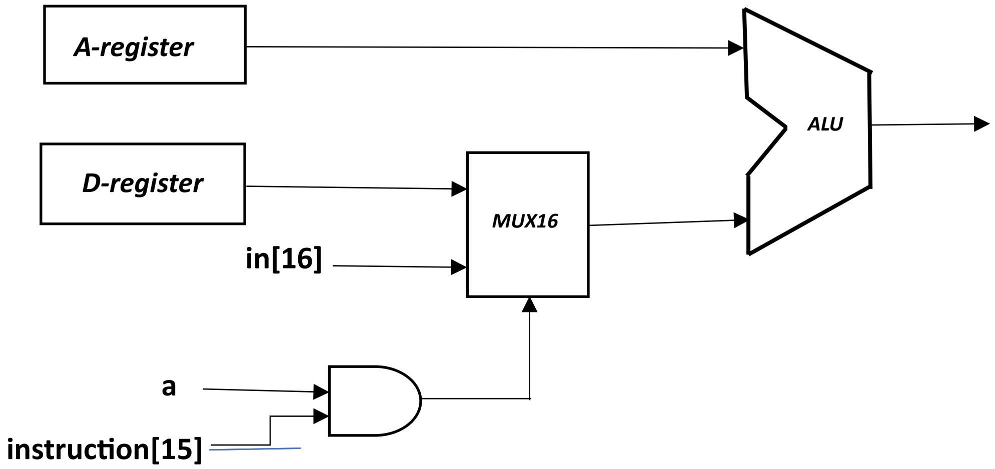
- **Example : 1xxa011111010111**
- 15th bit indicates type of instruction i.e. **‘1’**.
- 14th,13th bits **xx** indicates don’t care.
- 12th bit **‘a’** indicates when a=0 , ALU use contents of D-register and A-register.
  a=1 , ALU use contents of D-register and M memory register specified by A.
  M= RAM[A].
- 11th-6th bits indicates select pins of ALU zx, nx, zy, ny, f, n$_0$.
- 5th-3rd bits indicates temporary destination used by ALU (D-register).
  ALU is having 3 outputs mainly Memory(RAM), D-register and A-register.
  Here in this **example,*5th-3rd bits are ‘010’.***
  5th bit is 0 indicates that it does not store any data from output into A-register.
  4th bit is 1 indicates that it store data in D-register.
  3rd bit is 0 indicates that it doesn’t change the output data/ doesn’t write that data into data memory.
- 2nd-0th bits indicates for jump condition.

ALU inputs:

ALU takes inputs.

a=0 , A and D registers are used.

a=1 ,M and D registers are used.



A and C are 2 instructions. But 12th bit ‘a’ is from C-instruction. So for C type instruction 15th bit must be 1.When 15th bit(MSB) is 1 and a= 0 or 1 based on the select pin MUX is activated.

But MSB bit must be 1 if not MUX doesn’t activate and this operation can’t be from C-instructon.

ALU output:

5th-3rd bits acts as a load.

5th bit A-register load.

4th bit D-register load.

3rd bit writes output of ALU to data memory.

JUMP condition:

2nd-0th bits indicates for jump condition.

When, **111 → Unconditional jump**

001 → out > 0

010 → out ≥ 0

100 → out < 0

If the condition is satisfied then jump directly to a particular instruction.

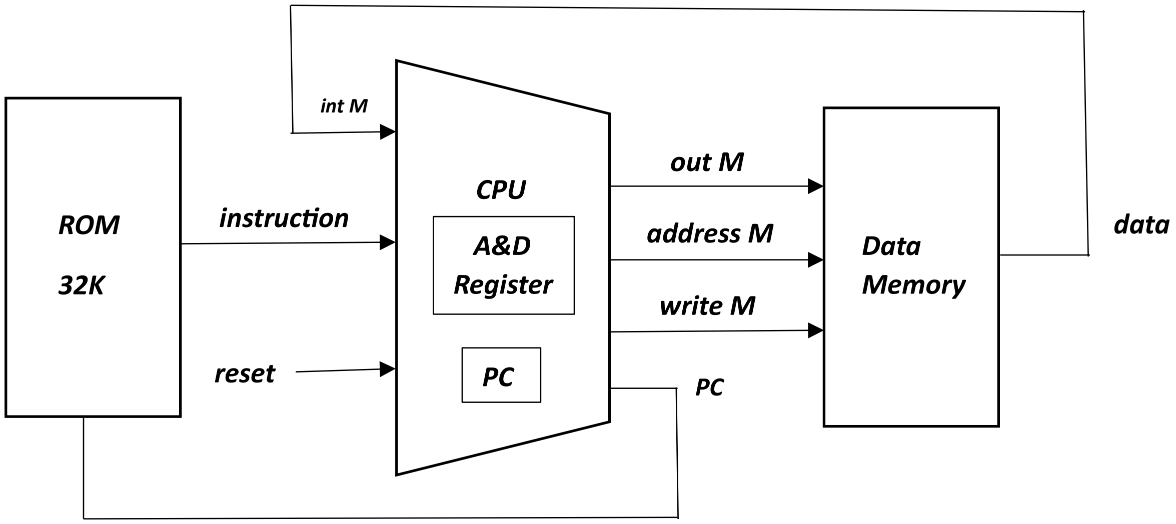
In the example, it is 111

If A=29 then it directly jump to 29th location of ROM.

A-instruction → A = Value → RAM [Value].

C-instruction → ALU computation.

COMPUTER IMPLEMENTATION:

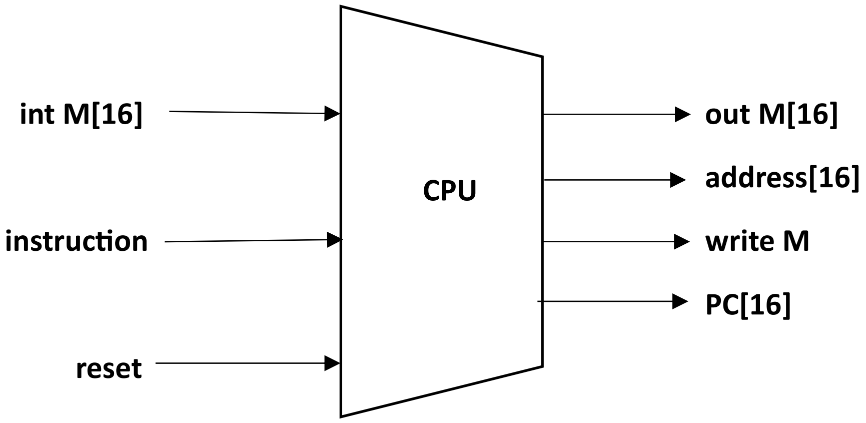


PC updates the instructions sequentially in ROM.

Reset the CPU so that PC starts count from 0.

First A type instruction is given to the CPU, based on that address M is selected. Which is at which register Location of Data memory should be selected and that data is given to CPU.

Then C-type instruction is given to the CPU, based on that operation was performed and based on "010" bits, write M performed or not is decided if then out M is activated and change the data by CPU into Data Memory (write operation in data memory).

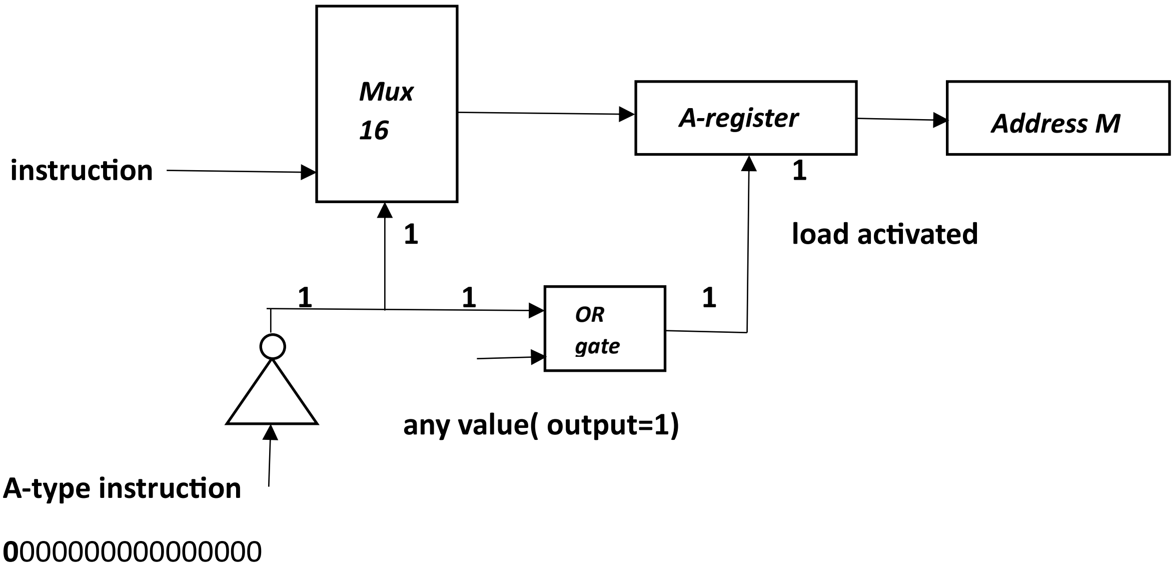


We have 2 types of instructions (A&C type) for that use 1 Mux16.

To send A-type instruction from second input(b) of Mux16 use 1 NOT gate.

A-instruction stored is in A-register. But it is stored or not is decided by the load pin for that use OR gate. To activate load pin and to store in A-register 1 is connected.

Whatever stored in A-register tells us which register(location) from the RAM is selected. And address of RAM is obtained.

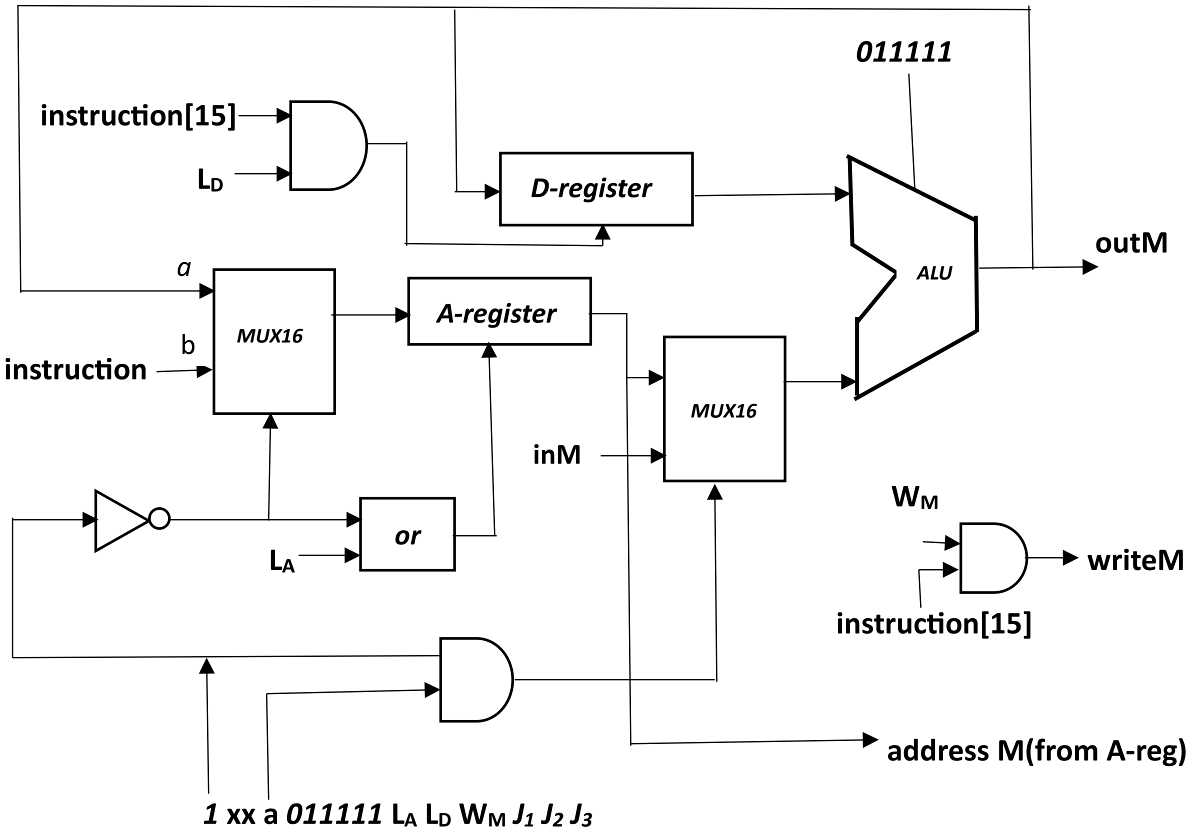


When C-type instruction is given to NOT gate, then NOT gate produces 0 at the output. This is when select pin is 0, ‘a’ input is selected. But we want to give all the instructions through ‘b’ input only.

Whenever C-type instruction comes into picture ALU is activated.

011111 these bits are taken as select pins of ALU.

Based on **‘a’** bit the inputs are connected to ALU.



Output goes to 3 places:

RAM (W_M), A-register (L_A) and D-register (L_D)

out M connected to RAM and connected to D-register directly. But A-register connected through MUX.

When C-type instruction is given to the (MSB bit) NOT gate will produce '0' due to this select pin becomes '0' and it selects 'a' input internally through that out M data goes to the A-register. But it is stored or not is decided by 'L_A' bit so, 2nd input of OR gate is 'L_A' bit.

If ‘L_A' is 1 then OR gate will produce '1' which activate the load of A-register and directly outM is stored, if 'L_A' is 0 then outM data is reached but as load of A-register is not activated so it doesn't store the data.

Similarly ‘L_D’ is applied to the D-register. But just remember these all are from C-type instruction. So, ‘L_D’ is same as MSB-bit of instruction i.e. 15th bit.

For writeM, we have to activate through the ‘W_M’ bit which is also from C-type so one more AND gate is used.

PC IMPLEMENTATION: (Along with jump/sequential)

From 32K combinations, which register we want to choose is decided by the PC.

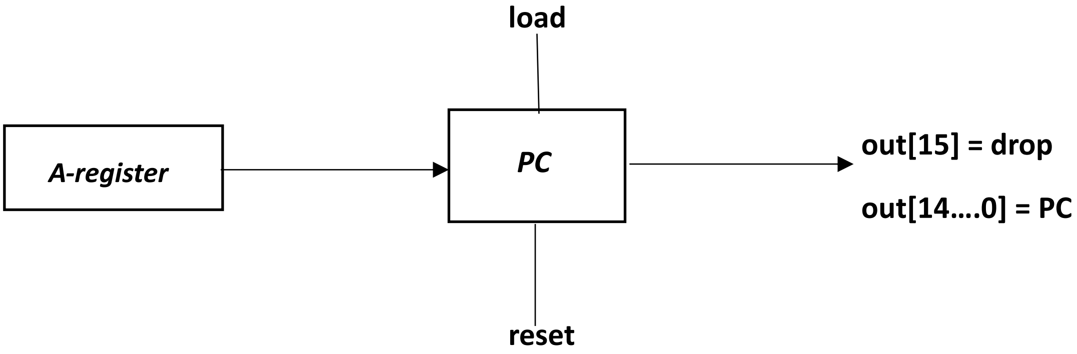
So, for 32,000 combinations 15-bit address is required.

How to convert 16-bit PC to 15-bit address?

We drop the MSB-bit i.e. 15th-bit as reset pin for initiating from CPU.

But to jump at new location(register) of the ROM we require load pin. So, that new address is fetched to the PC(input). Because of load=1, this address becomes new address for the PC. This new address is stored at A-register which is connected to PC.

So, whenever we get a ***conditional jump, load pin is activated*** based on that PC is updated.

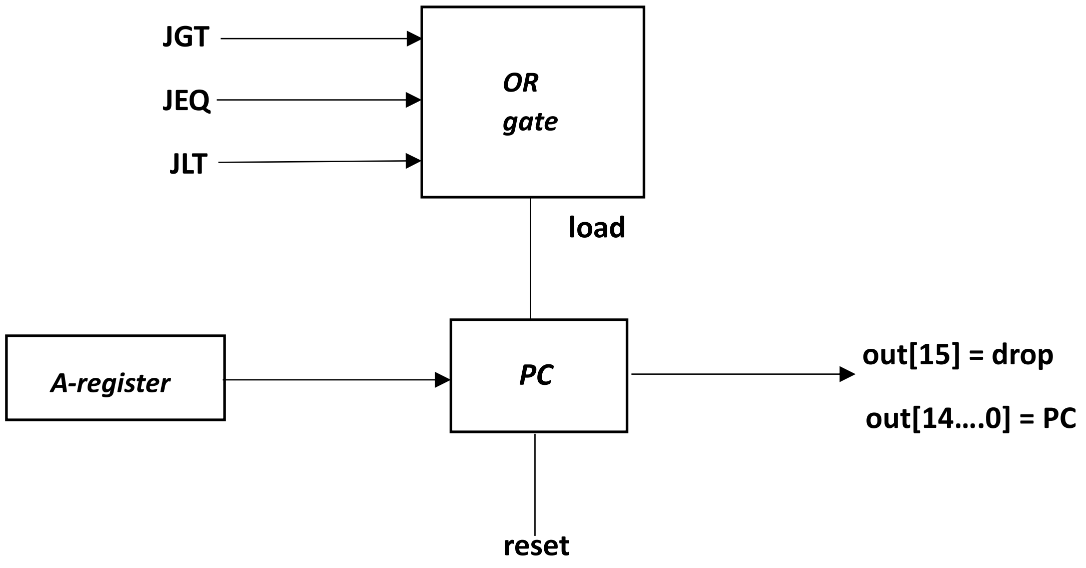


BUILDING LOAD PIN: (Based on Jump {Conditional Jump})

J_1 J_2 J_3

1. **JGT** → output greater than zero
2. **JEQ** → output equal to zero
3. **JLT** → output less than zero

If any one of the above condition is true then make load pin true(activated).



Example:1

A=29, D=0 condition D,JEQ

Since D=0 is true so jump at 29, for that load must be 1. OR gate produces 1 then PC is updated. A = 29 → PC = 29

Example:2

A=29, D=1 condition D,JGT

Since D>0 is true so jump at 29, for that load must be 1. OR gate produces 1 then PC is updated. A = 29 → PC = 29

JGT or JEQ have no difference they are different because of D value.

Main goal is to bring A = data to PC = data whatever may be the condition JGT or JEQ or JLT.

1.JGT:

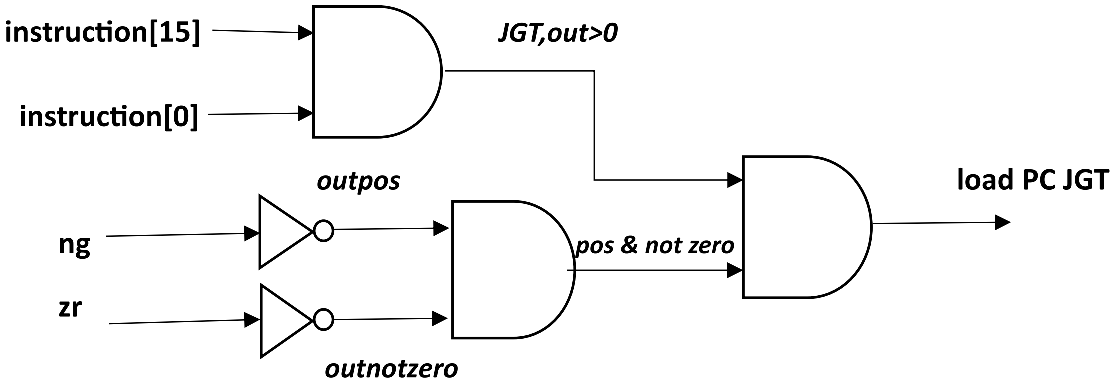
1xxa011111010***001***

This is a condition when out > 0.

To check output greater than 0 or not:

- First of all it should be a C-type instruction and LSB bit should be 1 then we can say it is having jump or not.
- Checking ALU outputs weather it is positive or negative, zero or non-zero, give these pins to AND gate they tell us output greater than zero or not equal to zero.

When both the above conditions are true then it produce 1 to load pin and the operation is activated.



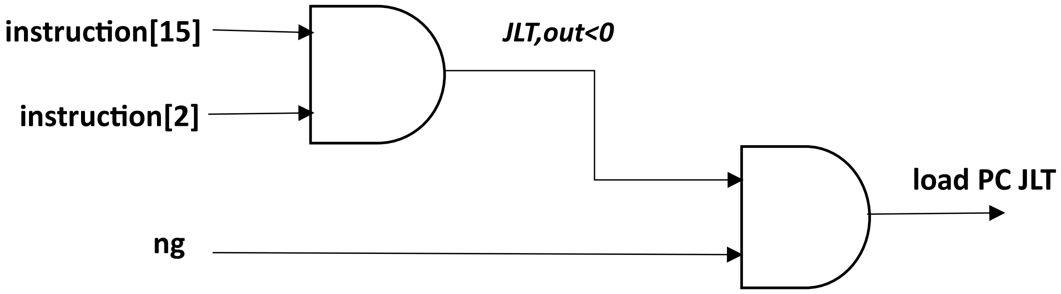
2.JLT:

1xxa011111010***100***

This is a condition when out < 0.

It should be C-type instruction and 2nd-bit of instruction is 1. So that jump is activated.

Since out < 0 negative pin is directly connected.



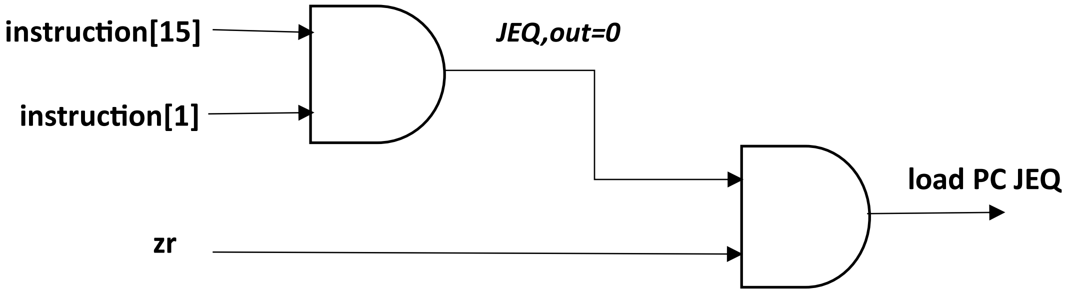
3.JEQ:

1xxa011111010***010***

This is a condition when out = 0.

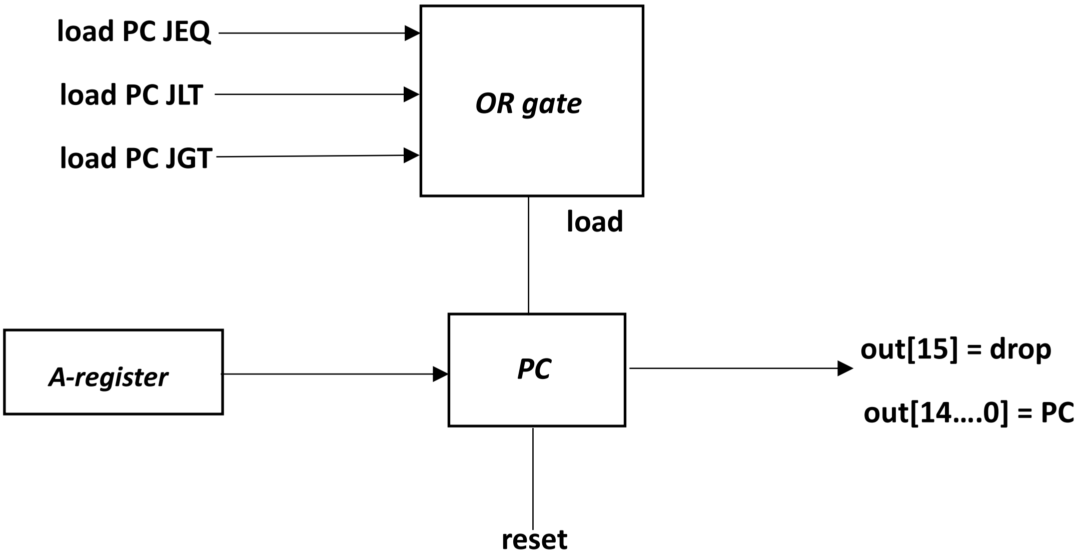
It should be C-type instruction and 1st-bit of instruction is 1. So that jump is activated.

Since out = 0 zero pin is directly connected.



4.UNCONDITIONAL JUMP:

When any one of the input to OR gate is 1, output of OR gate is 1.


load PC JEQ
load PC JLT
load PC JGT
OR gate
load
A-register
PC
out[15] = drop
out[14….0] = PC
reset