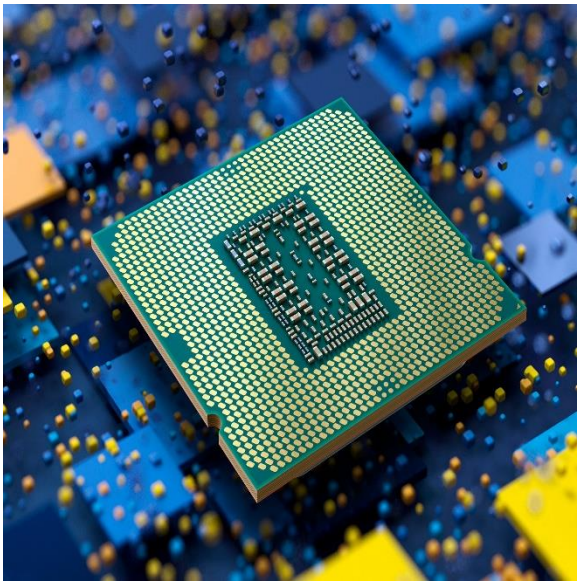
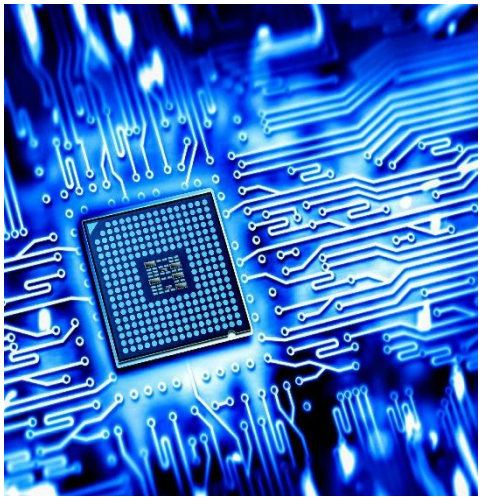
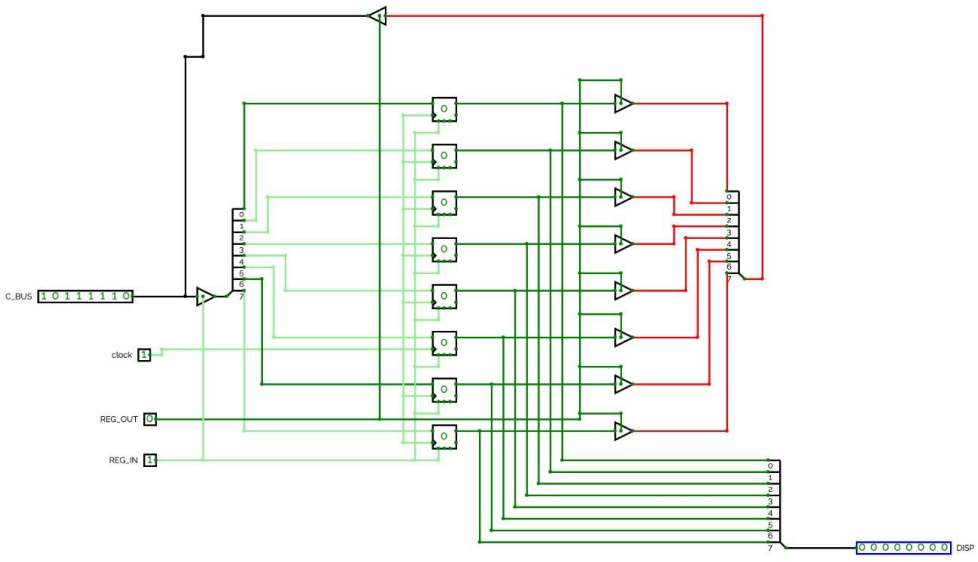


GAJENDRA-I





REGISTER:

- The register in our CPU operates bidirectionally, capable of both storing input from the DATA and outputting to the DATA. This bidirectional function is managed by three control signals:

- REG_IN: When active, it allows the register to store incoming data from the DATA input. This activation is facilitated through the individual D-flipflops Enable mechanism.

- REG_OUT: Activation of this signal allows the register to output its stored data to the DATA.

- CLK: This signal serves as the clock input for the individual D-flipflops within the circuit, coordinating their operations.

- Within our CPU, there exist three distinct general-purpose registers:

- Accumulator (REG_A): This register is specifically designated for carrying the summation or difference across multiple instructions.

- REG_B: Functionally, REG_B acts as the intermediary between the memory and the CPU, serving as the primary register for communicating with the memory.

- REG_C: This register serves a dual purpose. It acts as both an output register and facilitates the implementation of the SWAP instruction, allowing for the exchange of data between the Accumulator and REG_B.

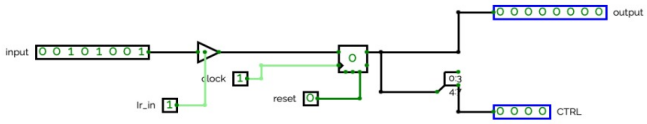
HOW NUMBER OF REGISTERS IMPROVE SOFTWARE FLEXIBILITY:

- Reduced Memory Access: More registers allow storing frequently used variables, reducing reliance on slower memory access and enhancing overall performance.

- Improved Function Handling: Increased registers aid in handling function calls, parameter passing, return values, and local variable storage, improving overall function efficiency.

- Improved Register Allocation: Additional registers offer the compiler more options for allocating variables to registers, optimizing the usage of available resources.

- Loop Performance: Greater register availability allows loop variables to be stored in registers, optimizing loop execution and reducing loop overhead.



Instruction register:

The Instruction Register, akin to general-purpose registers, shares the same bit size.

It operates unidirectionally and employs control words akin to those in general-purpose registers.

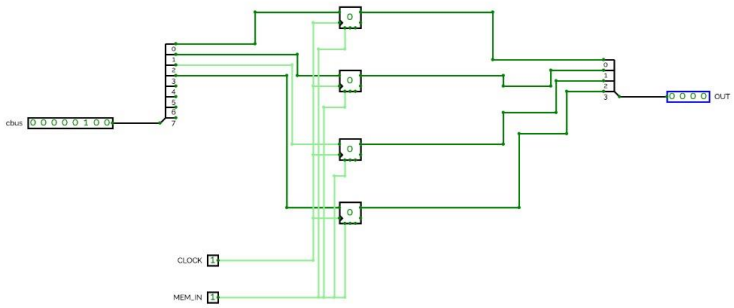
The Instruction Register acquires input from the processor's common bus and stores it.

Specifically responsible for relaying the 4 Most Significant Bits (MSBs) of the opcode to the Controller.

Sends the same 4 MSBs to the instruction decoder, aiding in the determination of operations for the Arithmetic Logic Unit (ALU).

Capable of providing addresses to the Memory Address Register (MAR) for certain instructions when required.

Utilizes D-flipflops for data storage, with CTRL serving as an unrestrained output.



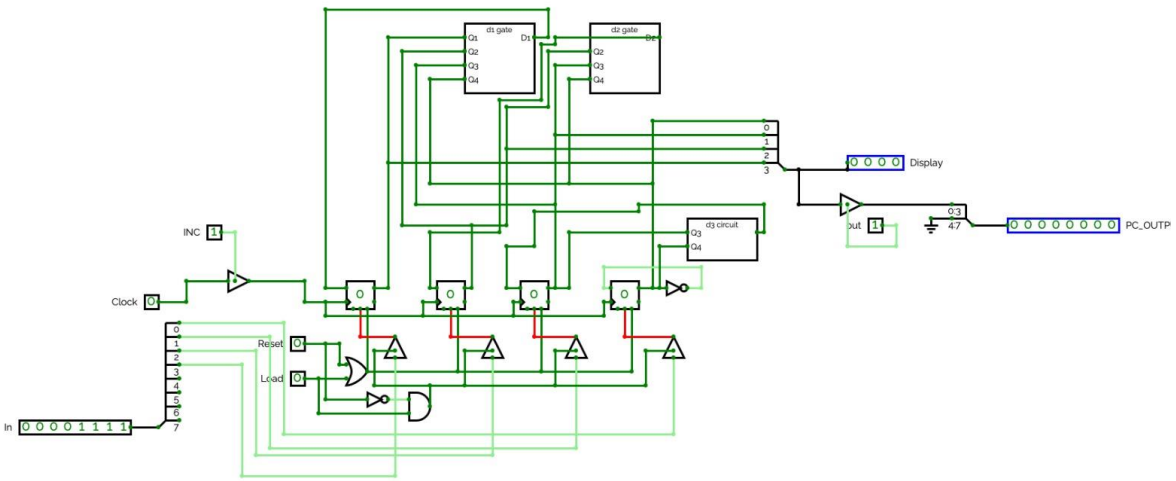
Memory address register:

The Memory Address Register (MAR) is a 4-bit register storing addresses destined for the ROM within the processor.

Constructed via a bidirectional 4-bit register module, it uses a 4-bit Reg module to capture and store the 4 Least Significant Bits (LSBs) when activated by MAR_IN control.

The MAR_ADD output remains consistently active, providing a 4-bit output that specifies the correct address within the ROM.

Crucially, the MAR serves as an intermediary for the Program Counter (PC) to send addresses to the ROM and simultaneously output them to the control bus. This intermediary role is vital as individual components cannot directly communicate with the common bus, necessitating the MAR's function of storing and providing addresses.



Program counter:

The Program Counter, a 4-bit register using the 4-bit Reg module akin to MAR, serves to dispatch the address of instructions stored in the ROM.

Load function allows it to receive addresses from the common bus.

Flipflops and combinational logic facilitate incrementing its value by one and storing it back into the register.

Bidirectional functionality allows for output to the common bus when OUT is active.

RESET control resets the counter to 0, initiating the program from the first instruction.

System reset, activated by the system reset control signal, also triggers the RESET function, effectively freezing the processor.

Exclusive control signals prevent multiple simultaneous activations.

Fundamentally, the Program Counter operates as a sequential counter, directing to the subsequent instruction in the program after each cycle, with the implementation ensuring the first 4 MSBs of its output are always 0.

	OPERATION	OPERANDS	DESCRIPTION	OPERATION
0000	NOP	-	No operation	-
0001	LDA	Ra,Ad	Load Accumulator	Ra←Ad(ROM)
0010	ADD	Ra,Rb,Ad	Add without carry	Rb←Ad(ROM) Ra←Ra+Rb
0011	SUB	Ra,Rb,Ad	Subtraction of positive numbers	Rb←Ad(ROM) Ra←Ra-Rb
0100	LDI	Ra,K	Load Immediate	Ra←K
0101	OUT	Rc	Display on Hex Display	Rc←Ra
0110	JMP	Ad	Jump to Address	PC←Ad(ROM)
0111	JNZ	Ad	Jump when not zero	PC←Ad(ROM)
1000	SWAP	Ra,Rb	Swap values in the registers	Rc←Rb Rb←Ra Ra←Rc
1001	MOVAC	Ra,Rc	Move the data	Rc←Ra
1010	MOVBA	Ra,Rb	Move the data	Ra←Rb
1011	MOVCB	Rb,Rc	Move the data	Rb←Rc
1100	MOVAB	Ra,Rb	Move the data	Ra←Rb
1101	MOVCA	Ra,Rc	Move the data	Rc←Ra
1110	MOVBC	Rb,Rc	Move the data	Rb←Rc
1111	HALT	-	Halt	RESET=1

LDA – Load Accumulator

Description:

This instruction loads the Accumulator with the contents from the memory by looking up at

the 4-bit address provided.

Operation:

Ra←XXXX(ROM)

Syntax:	Operands:	Program Counter:
LDA XXXX	X=0 or X=1	PC←PC+1

8-bit opcode:

0001	XXXX

ADD-Add without carry

Description

This instruction loads the Register-B with the contents from the memory by looking up at the

4-bit address provided and then adds up the contents of Accumulator and Register-B. And

stores the final sum in Accumulator.

Operation:

Rb←XXXX(ROM)

Ra←Ra+Rb

Syntax:	Operands:	Program Counter:
ADD XXXX	X=0 or X=1	PC←PC+1

8-bit opcode:

0010	XXXX

SUB – Subtraction of positive numbers

Description

This instruction loads the Register-B with the contents from the memory by looking up at the

4-bit address provided and then subtracts the contents of Register-B from Accumulator. And

stores the final difference in Accumulator, provided the difference non-negative.

Operation:

Rb←XXXX(ROM)

Ra←Ra-Rb

Syntax:	Operands:	Program Counter:
SUB XXXX	X=0 or X=1	PC←PC+1

8-bit opcode

0011	XXXX

LDI – Load Immediate

Description: Loads least significant 4 bit input into Accumulator of 8 bits with first 4 MSB as zeros.

Operation:

Ra←K

Syntax:	Operands:	Program Counter:
LDI XXXX	X=0 or X=1	PC←PC+1

8-bit opcode:

0100	XXXX

OUT – Display on Hex display:

Description: Copies the number from A to C to know what is there in an accumulator.

Operation:

Rc←Ra

Syntax:	Operands:	Program Counter:
OUT XXXX	X=0	PC←PC+1

8-bit opcode:

0101	XXXX

JMP – Jump to address

Description:

This instruction takes the control to the memory location specified.

Operation:

ROM address: XXXX

Syntax:	Operands:	Program Counter:
JMP XXXX	X=0 or X=1	PC←PC+1

8-bit opcode:

0110	XXXX

JNZ – Jump when Non-zero

Description:

This instruction takes the control to the given memory location when the value returned after

performing an operation is non-zero.

Operation:

If NZ=1,ROM address=XXXX

Syntax:	Operands:	Program Counter:
JNZ XXXX	X=0 or X=1	PC←PC+1

8-bit opcode:

0111	XXXX

SWAP(AB) – Swap A and B

Description

Swaps the contents present in Accumulator and Register-B by storing contents of

Reg_B in Reg_C during swapping.

Operation:

Rc←Rb

Rb←Ra

Ra←Rc

Syntax:	Operands:	Program Counter:
SWAP XXXX	X=0	PC←PC+1

8-bit opcode:

1000	XXXX

MOVAC - Copy data from A to C

- Description: The contents present in Register A are loaded into the Register C
- Operation : Rc ← Ra

Syntax	Operands	Program Counter
MOVAC XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1001	XXXX

MOVBA- Copy data from B to A

- Description: The contents present in Register B are loaded into the Register A
- Operation : Ra ← Rb

Syntax	Operands	Program Counter
MOVBA XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1010	XXXX

MOVCB - Copy data from C to B

- Description: The contents present in Register C are loaded into the Register B
- Operation : Rb ← Rc

Syntax	Operands	Program Counter
MOVCB XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1011	XXXX

MOVAB - Copy data from A to B

- Description: The contents present in Register A are loaded into the Register B
- Operation : Rb ← Ra

Syntax	Operands	Program Counter
MOVAB XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1100	XXXX

MOVCA - Copy data from C to A

- Description: The contents present in Register C are loaded into the Register A
- Operation : Ra← Rc

Syntax	Operands	Program Counter
MOVCA XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1101	XXXX

MOVBC - Copy data from B to C

- Description: The contents present in Register B are loaded into the Register C
- Operation : Rc ← Rb

Syntax	Operands	Program Counter
MOVBC XXXX	X=0	PC ← PC+1

- 8 - bit opcode

1110	XXXX

HALT – Halt the program

Description:

This instruction terminates the program by freezing the program counter. This is done by

resetting the program counter. The control is taken back to the first memory location.

Operation:

RESET=1

Syntax:	Operands:	Program Counter:
HALT XXXX	X=0	PC←PC+1

8-bit opcode:

1111	XXXX

Assembly Programs implemented using the instruction set

Add two numbers and displaying the result

ADDRESS	ASSEMBLY CODE	Machine code
0x0	LDA 0x5	0x15
0x1	ADD 0x6	0x36
0x2	MOVA 0x0	0x90
0x3	HALT 0x0	0xf0
0x4		0x00
0x5		0x34
0x6		0x12

- Value at address 0x5 -34 is loaded to the accumulator.
- Value at address 0x6 – 12 is added to the value in the accumulator (to give 46) and is stored in it.
- Value in accumulator is moved to REG_C to display.

Adding and subtracting four numbers in some combination

Address	Assembly code	Machine code
0x0	LDA 0x5	0x15
0x1	SUB 0x6	0x46
0x2	ADD 0x7	0x37
0x3	SUB 0x8	0x48
0x4	HALT 0xf	0xf0
0x5		0x17
0x6		0x08
0x7		0x25
0x8		0x12

- Value at address 0x5 -17 is loaded to the accumulator.
  Value at address 0x6 -8 is subtracted from the value of accumulator-to give 0f and is stored in it.

  Value at address 0x7- 25 is added to the value of accumulator-to give 34 and is stored in it.

  Value at address 0x8 -12 is subtracted from the value of accumulator- to give 22 and is stored in it.

Address	Assembly code	Machine code
0x0	LDA 0x9	0x19
0x1	SWAP 0x0	0x70
0x2		
0x3		
0x4		
0x5		
0x6		
0x7		
0x8		
0x9		
0xa		
0xb		

Sequence of control words for each instruction:

NOP:

NOP :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 0

T3 : 0

T4 :0

LDA :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << ir_out | 1 << mar_in

T3 : 1 << mem_out | 1 << rega_in

T4 : 0

ADD :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << ir_out | 1 << mar_in

T3 : 1 << mem_out | 1 << regb_in

T4 : 1 << alu_out | 1 << rega_in

SUB :

T0 : 1<<pc_out | 1<<mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << ir_out | 1 << mar_in

T3 : 1 << mem_out | 1 << regb_in

T4 : 1 << alu_out | 1 << rega_in

LDI :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << ir_out | 1 << rega_in

T3 :0

T4 :0

OUT :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << rega_out | 1 << regc_in

T3 : 1 << regc_out

T4 :0

JMP :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << ir_out | 1 << pc_load

T3 :0

T4 :0

JNZ (when Z ≠ 1) :

T0: 1 << pc_out | 1 << mar_in

T1: 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2: 1 << ir_out | 1 << pc_load

T3:0

T4:0

JNZ (when Z = 1) :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 :0

T3 :0

T4 :0

SWAP :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << rega_out | 1 << regc_in

T3 : 1 << regb_out | 1 << rega_in

T4 : 1 << regc_out | 1 << regb_in

MOV :

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 : 1 << alu_out | 1 << rega_in

T3 :0

T4 :0

HALT:

T0 : 1 << pc_out | 1 << mar_in

T1 : 1 << mem_out | 1 << ir_in | 1 << pc_inc

T2 :1<<reset

T3 :0

T4 :0