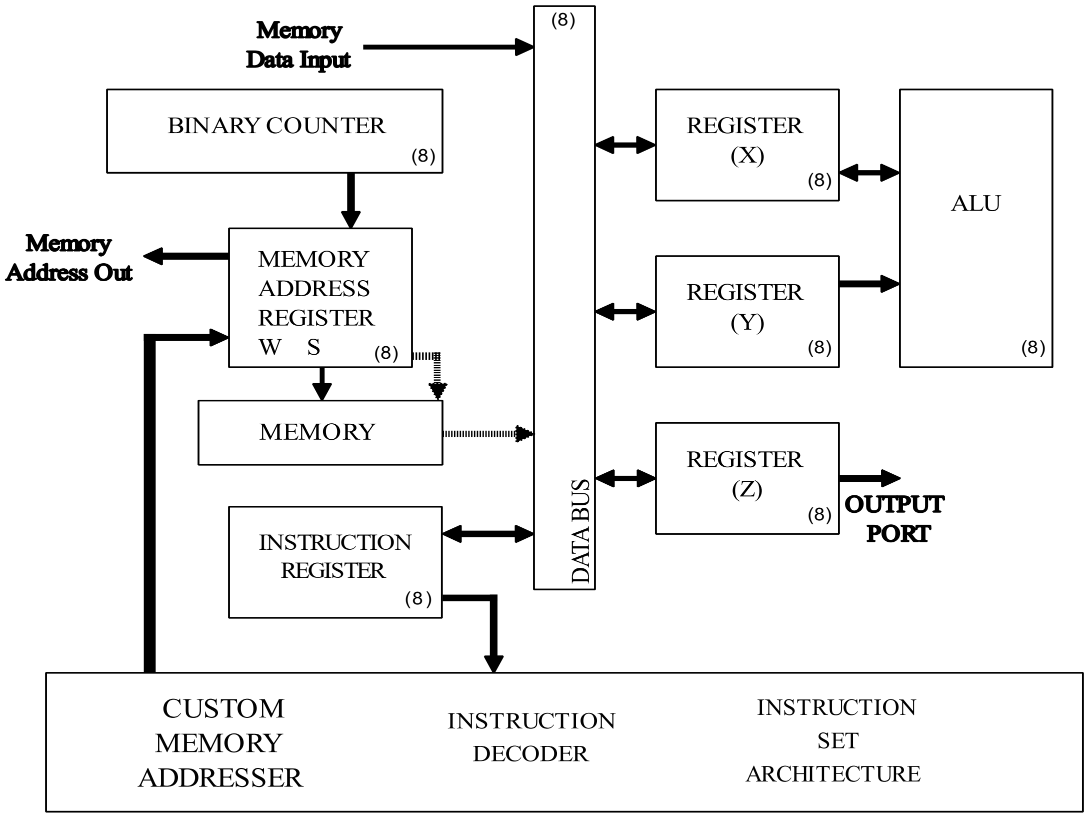
TMP8 (8-Bit Microprocessor)

The TMP8 is a complete computer system central processing unit which may be interfaced with memory having capacity of 16 Bytes. The processor communicates over 8-Bit data and 4-Bit Address Bus. The CPU contains an 8-Bit parallel arithmetic unit, a dynamic RAM (three 8-Bit multipurpose data registers), and a complete memory driven instruction decoding and control logic unit.

Features

- 8-Bit Parallel Microprocessor
- 15 Instructions (ISA)
- Directly Addresses 16 Byte Memory (16 x 8) Bits (4-Bit Address Bus)
- Contains 3x Multipurpose Registers
- Opensource ISA



Block Diagram

CONTENT

Page.

I. INTRODUCTION

The TMP8 is an 8-bit parallel central processor unit a temporary and early development design for implementation of RISA-MPD (Reprogrammable Instruction Set Architecture for Microprocessing Devices).

The processor communicates over an 8-bit Data bus and a 4-bit Address bus. Since Processor is in early development stage it doesn’t have any external control lines for memory addressing but fully ISA driven design for memory management, 8-Bit Data can be transmitted between the CPU and external Memory between 0x0 to 0xF Address possibility.

This Architecture contains a total of four (4) 8-Bit Multipurpose Registers, an Instruction Register, three (3) General Purpose Registers named as X, Y and Z. Here Register-X is accumulator, Register-Z is output latch Register and Register-Y is for Register to Memory data communicator, an 8-Bit Binary Counter for addressing Memory through Memory Address Register (MAR).

The control portion of the architecture use programmed (Flashed) ISA to implement variety of register transfer, memory transfer, basic arithmetic control. All the instructions are coded in one byte (8 bits); Presently Immediate data load, RAM Reverse transfer is unavailable.

Input and Output are in binary format and not encoded or decoded on input or output operations.

The Instruction set of the TMP8 consists of 16 instructions including data manipulation, basic binary arithmetic, and jump.

The normal program flow of TMP8 may be paused (HALT) using external inputs and Resettable using external inputs.

II. Basic Functional Blocks

The four basic functional blocks of TMP8 processor are the Registers, Arithmetic logic unit, Memory, Instruction Set Architecture (Control Unit).

A. Registers

TMP8 have 2 types of registers MP-Registers (Multipurpose Registers) and Instruction Register.
MP-Registers: - There are total three (3) MP-Registers X, Y and Z; Here Register X is accumulator, Register Y is memory communicator and Register Z is output latch, since those registers X, Y and Z can also be used for General Purpose operations they are Multipurpose registers.

Instruction Register: - Instructions are fetched from memory, stored in the instruction register, and decoded for control of ALU, Memories or Data transfers, It also responsible for dividing instruction into opcode and operand (generally address for transfer functions).

B. ALU (Arithmetic Logic Unit)

All arithmetic operations are ADDITION and SUBTRACTION are carried out in the 8-bit parallel arithmetic unit. Two temporary register "X" and register "Y" are used to store the accumulator and next followed byte for ALU operations. After any arithmetic operation the respective result override the register-X data.

C. Memory

Two separate memories are used in the complete system of TMP8, Program Memory (Flash Memory) and RAM (optional). These external memories are instruction specific and can be accessible through specific instructions only.

1. Program Memory (Flash Memory)

The main programming instructions and constants can be stored in this memory, A maximum of 16 Bytes can be directly accessible via address bus (4-Bits) and instructions can be manipulated via data bus (8-bits), This is an Read only Memory (ROM). Instructions can only be fetched from Flash memory, any new instruction or modification of instruction is not possible in runtime, Hence DFT (Dynamic Fetch Function) can be used as substitute.

2. **RAM (Optional Memory)**
   An external RAM unit can be used for temporary data storage or runtime data storage unit, A maximum of 16 Bytes can be directly accessible via same data and address bus as followed by flash memory. This memory provides reading as well as writing of data on specific address via register-Y only. An extra instruction DFT can fetch the next instruction from RAM instead of ROM for dynamic programming and execution.

D. <u>Control Unit</u>

The Control Unit consists of Instruction Register, Instruction Decoder and Instruction Set, all three main components made TMP8 to execute instructions on its own.

1. Instruction Register:
   Stores 8-bit instruction in it fetched from ROM or RAM (DFT) specific.

2. Instruction Decoder:
   Accepts the incoming instruction form instructions register split it into opcode (Main instruction) and operand (additional address to be changed) and generates an address of 8-bit by decoding, that specific instruction is now ready to be execute.

3. Instruction Set:
   Accepts the Instruction address from instruction decoder and fetch the data (16-Bit Data line) from main instruction firmware memory for that instruction and control the inputs and outputs control of different components.

III. BASIC INSTRUCTION SET

The following section presents the basic instruction set of the TMP8, Its data formats and execution formats.

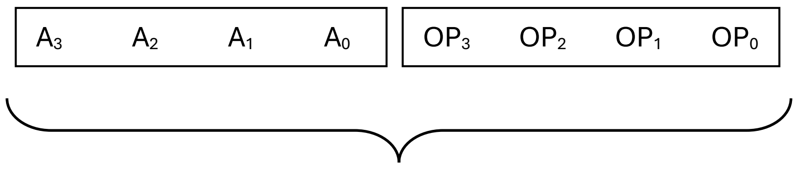
Data and Instruction Formats

Data in the TMP8 is stored in the form of 8-bit binary integers. All data transfers to the system Will be in the same format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

DATA BYTE

The program instruction is of one byte in length. Every instruction is designed to be operated in Single byte.
Program instruction is split into two parts Memory address and opcode.



Complete Program Instruction Format

A_3	A_2	A_1	A_0	Address for addressing mode.
OP_3	OP_2	OP_1	OP_0	OP CODE

For standard and default configuration logic "1" is defined as a high-level logic or ON state and a logic "0" is defined as a low-level logic or OFF state.

Instruction Set Architecture

TMP8 is a simplistic processing unit which contains 16 Instructions in total which are categorized as:

1. Register Load Instructions (3)
2. Arithmetic Instructions (2)
3. R2R Transfer Instructions (Register to Register) (6)
4. Memory Transfer Instructions (2)
5. Jumping Instruction (1)
6. Dynamic Fetching Instruction (1)

Register Load Instructions

Loding data directly from program memory (ROM) to either of the registers through passing an address of memory.

REPRESENT.	**INSTRUCTION CODE**								**DESCRIPTION OF OPERATION**
	D_7	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**	
LDX	**A**	**A**	**A**	**A**	0	0	0	1	Load data into register – X from address AAAA of Program Memory.
LDY	**A**	**A**	**A**	**A**	0	0	1	0	Load data into register – Y from address AAAA of Program Memory.
LDZ	**A**	**A**	**A**	**A**	0	0	1	1	Load data into register – Z from address AAAA of Program Memory/ Outputting Memory Data.

Arithmetic Instructions

Performing arithmetic operation between Register-X and Register-Y and storing its result back to register-X.

REPRESENT.	**INSTRUCTION CODE**								**DESCRIPTION OF OPERATION**
	D_7	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**	
ADD	**X**	**X**	**X**	**X**	0	1	0	0	Adding data of register – X and register – Y and store the result in register – X.
SUB	**X**	**X**	**X**	**X**	0	1	0	1	Subtracting data of register – X and register – Y and store the result in register – X.

*INC and DEC (Increment and Decrement) instructions are yet to be implement in default instruction set architecture, but function INC or DEC can be performed using the default instruction set architecture, Digital implementation of INC and DEC requires controlled Carry and Borrow of ALU units.

Register to Register Transfer Instructions
Data in either of the following register X, Y or Z can be transferred to any Multipurpose register, R2R doesn't allow to manipulate the data of instruction register by any means.

REPRESENT.	INSTRUCTION CODE D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0	DESCRIPTION OF OPERATION
XTY	**X X X X**	0 1 1 0	Direct Data Transfer from register – X to register – Y.
YTZ	**X X X X**	0 1 1 1	Direct Data Transfer from register – Y to register – Z.
XTZ	**X X X X**	1 0 0 0	Direct Data Transfer from register – X to register – Z.
YTX	**X X X X**	1 0 0 1	Direct Data Transfer from register – Y to register – X.
ZTY	**X X X X**	1 0 1 0	Direct Data Transfer from register – Z to register – Y.
ZTX	**X X X X**	1 0 1 1	Direct Data Transfer from register – Z to register – X.

Memory Transfer Instructions
To transfer the data from register to memory or memory to register can be performed by providing address on which data have to read or write and instruction, Data affected in using these instructions are being performed only in external RAM unit, only register – Y have access to communicate with memory either read or write.

REPRESENT.	INSTRUCTION CODE D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0	DESCRIPTION OF OPERATION
MOV	**A A A A**	1 1 0 1	Direct data transfer from register – Y to RAM (memory) address AAAA.
STY	**A A A A**	1 1 1 0	Store to Y, Store data from memory (RAM) address AAAA to register – Y.

Jumping and Dynamic Fetch
To jumping back to a previous memory address to form a loop condition or subroutine condition jumping instruction can be used and to fetch an instruction from RAM instead of ROM, Dynamic fetch or DFT can be used.

REPRESENT.	INSTRUCTION CODE D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0	DESCRIPTION OF OPERATION
JMP	**A A A A**	1 1 0 0	Jumping to any address (AAAA+1), after jumping all instructions afterwards repeats and forms loop.
DFT	**A A A A**	1 1 1 1	Dynamically fetch an instruction from RAM (Memory) address AAAA to instruction register for extra instruction execution between main program execution.

Invalid Instruction (The First Beginning)

When the processor firstly starts up itself and start fetching first instruction the doesn't increments the program counter, which makes a circular dependency condition on NOP instruction, this leads to the HALT condition of the program, hence whenever NOP instruction occurs processor undergoes an infinite loop condition which freezes it at that address forever which makes NOP instruction a HALT.

REPRESENT.	**INSTRUCTION CODE**								**DESCRIPTION OF OPERATION**
	D_7	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**	
HALT	**X**	**X**	**X**	**X**	0	0	0	0	Pause the processing at that address of MAR (Memory Address Register) forever.

*There is no such instruction as NOP for the time being as soon as processor encounter HALT condition it pauses processor at that address by repeatedly fetching same instruction HALT over and over again.

IV. Control Signals

The TMP8 has a control output of 16-Bit (2 Bytes) length generated by instruction set architecture, this 2-byte control line control the Register(s), ALU, PC (Program counter), and other components of processing unit to work on its own.

Decoded instruction is used as the input for a permanent memory unit which contains data as the control lines output.

Bit Weight	Line Name	Description
0	M_OUT	Enable ROM data output to data bus.
1	RX_OUT	Enable Register – X data output to bus.
2	RY_OUT	Enable Register – Y data output to bus.
3	RZ_OUT	Enable Register – Z data output to bus.
4	M_IN	Enable RAM data input from Register – Y.
5	RX_IN	Enable Register – X data input from data bus.
6	RY_IN	Enable Register – Y data input from data bus.
7	RZ_IN	Enable Register – Z data input from data bus.
8	R2M	Enable transfer of 4-Bit Jumping address from ISA to ROM address for addressing mode.
9	RINX_IN	Enable Instruction Register data input from data bus.
10	A_OUT	Enable Arithmetic Addition output to data bus.
11	S_OUT	Enable Arithmetic Subtraction output to data bus.
12	CI	Enable Program Counter for up counting.
13	RAM_OUT	Enable RAM data output to Register – Y for selected address.
14	JMP	Enable Counter to jump to certain address.
15	RAM_IN	Enable RAM input from Register – Y for selected address.

Table. x ISA Control BUS

Instructions Execution

TMP8 uses a maximum of six (6) steps to execute an instruction, if number of steps are less then six (6) ISA automatically fetches the next instruction to instruction register, this leads to optimize time complexity of instruction execution and next instruction fetching time.

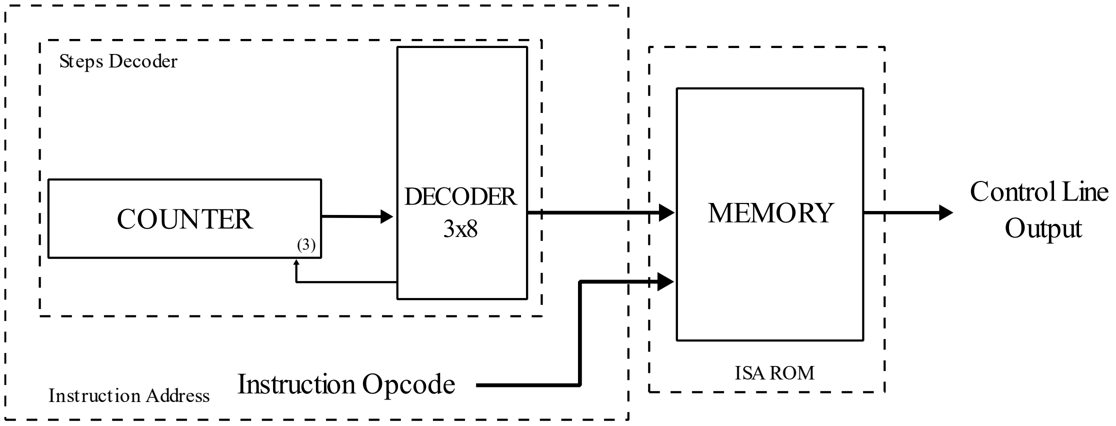


Fig. x Main Instruction Executor

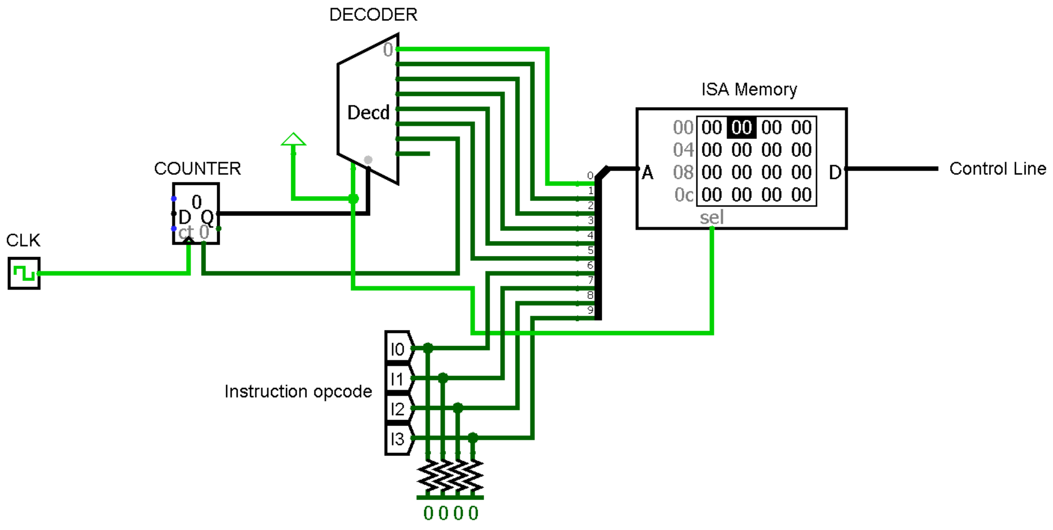


Fig. x Main Instruction Decoding and ISA Unit

Control BUS/Line

As the given instruction is decoded and send to memory address its steps start executing as the counter increases because of different decoder bits, As the instruction opcode changes it redirects the address to a totally new step.

V. Basic Interfacing

TMP8 can access external ROM (as program flash memory) and *RAM (as temporary storage memory), both can be interfaced via direct parallel communication with common data bus and address bus.

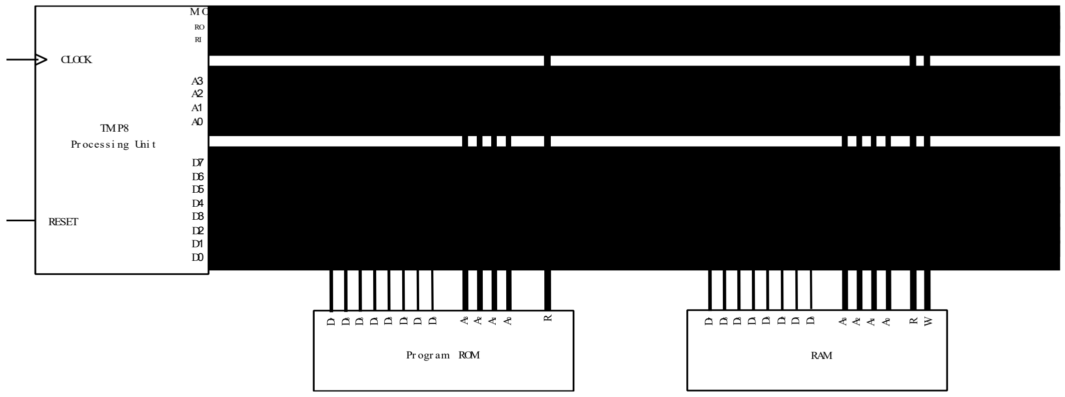


Fig. x TMP8 interfacing with Flash ROM and RAM module.

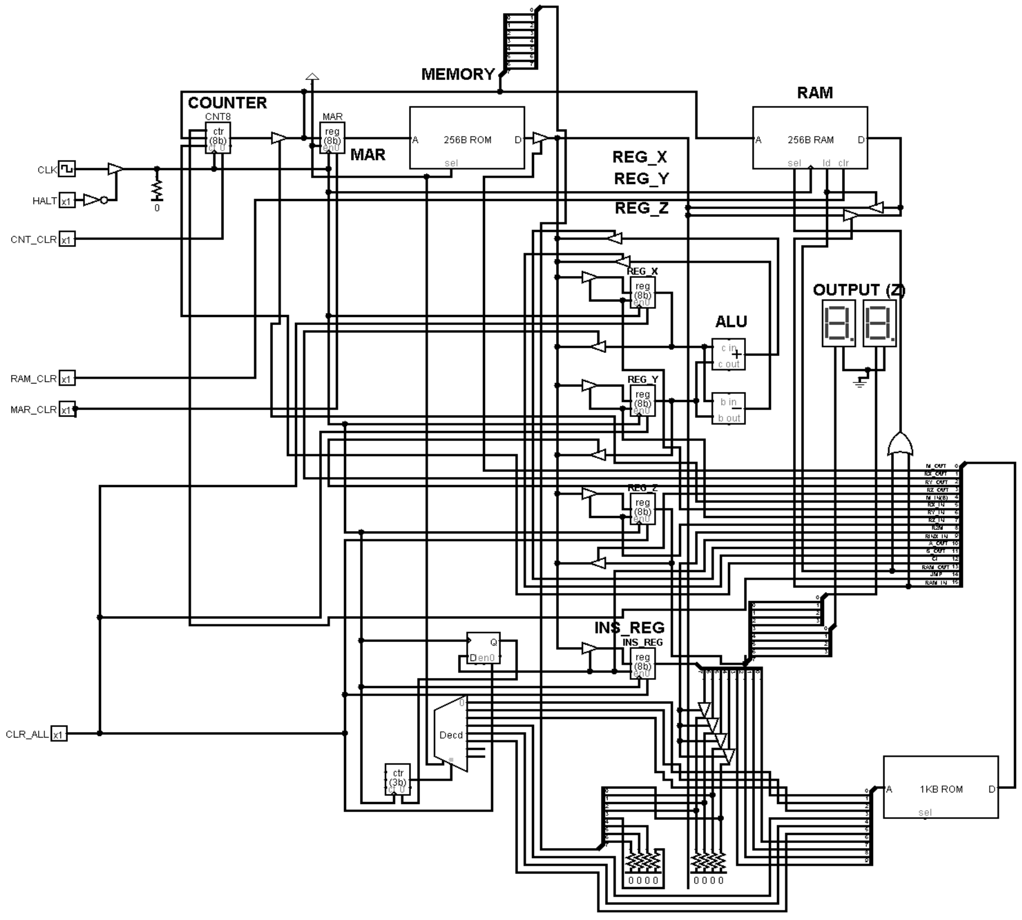


Fig. x Complete Interfaced Architecture

VI. Program Assembly and Execution

TMP8 is a simplistic processing unit which contains 16 Instructions in total, Instruction Decoder working as a six (6) steps sequencer to execute multiple steps. This execution of multiple steps is the execution of a complete instruction.

Execution of an instruction can have different steps but as soon as the first instruction, it initiates the steps to fetch next instruction.

1.) <u>LDX:</u>

When Instruction for LDX is once fetched onto instruction register where output of instruction register is always enabled, it will redirect the ISA Memory to the sequence address of LDX instruction.

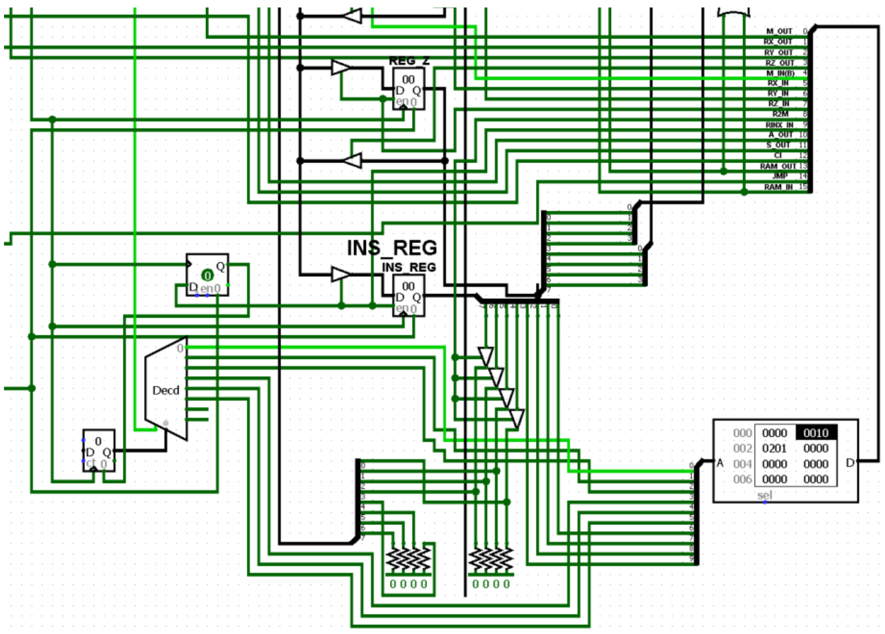


Fig. x Instruction Execution

As given the Fig. x the instruction output is split into two (2) buses, here lower half bits (0-3) are used to recognize the opcode and higher bits (4-7) are used to address back to the MAR (Memory address Resister), as the counter of decoder counts address for ISA memory changes in step wise and executes the steps.

Steps for LDX opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable buffers which transfers higher bits (4-7, working as address AAAA) of instruction register to memory, and AAAA Address is selected.	0x041H	0x0100H
2.	Enables the Main Memory to output the data selected at address AAAA. Also enables the Register – X Input (Load).	0x042H	0x0021H
3.	Enable Counter which increments the current address so the next instruction can be fetched.	0x044H	0x1000H
4.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x048H	0x0010H
5.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x050H	0x0201
6.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

2.) <u>LDY:</u>

Steps for LDY opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable buffers which transfers higher bits (4-7, working as address AAAA) of instruction register to memory, and AAAA Address is selected.	0x081H	0x0100H
2.	Enables the Main Memory to output the data selected at address AAAA. Also enables the Register – Y Input (Load).	0x082H	0x0041H
3.	Enable Counter which increments the current address so the next instruction can be fetched.	0x084H	0x1000H
4.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x088H	0x0010H
5.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x090H	0x0201
6.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

3.) <u>LDZ:</u>

Steps for LDZ opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable buffers which transfers higher bits (4-7, working as address AAAA) of instruction register to memory, and AAAA Address is selected.	0x0C0H	0x0100H
2.	Enables the Main Memory to output the data selected at address AAAA. Also enables the Register – Z Input (Load).	0x0C2H	0x0081H
3.	Enable Counter which increments the current address so the next instruction can be fetched.	0x0C4H	0x1000H
4.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x0C8H	0x0010H
5.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x0D0H	0x0201
6.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

4.) <u>ADD:</u>

Addition control line used as tri-state buffered control for addition output control, Whenever A_OUT signal enable it enables binary adder's output control.

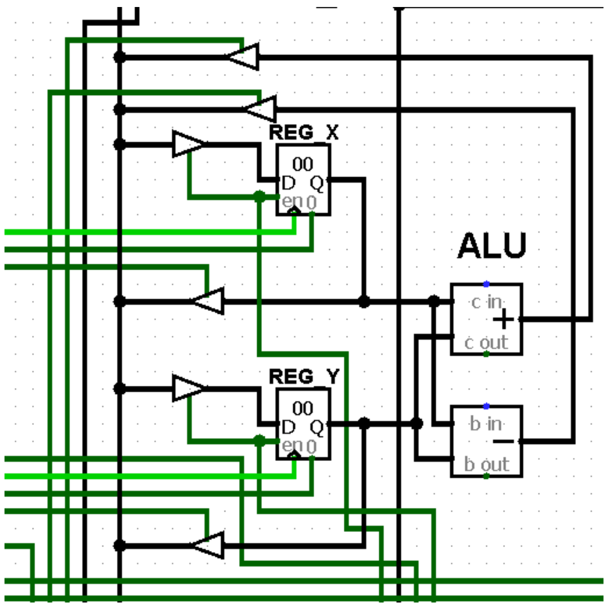


Fig. x ALU Controls and Execution.

As shown in Fig .x, Adder always keep on adding the data of Reg. X and Reg. Y and when the ADD instruction is called it enables the Binary Adder's output to enable tri-state buffer.

Steps for ADD opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable Reg. X Input (Load) and enable Adder to Bus tri-state buffer.	0x102H	0x0420H
2.	Enable Counter which increments the current address so the next instruction can be fetched.	0x104H	0x1000H
3.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x108H	0x0010H
4.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x110H	0x0201H
5.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

5.) <u>SUB:</u>

Steps for ADD opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable Reg. X Input (Load) and enable Subtractor to Bus tri-state buffer.	0x102H	0x0820H
2.	Enable Counter which increments the current address so the next instruction can be fetched.	0x104H	0x1000H
3.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x108H	0x0010H
4.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x110H	0x0201H
5.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

6.) Register to Register (R2R) Transfer (unoptimized):

a.) Reg. X -> Reg. Y:

Steps for XTY opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable Reg. X Output to Bus and enable Reg. Y Input (Load).	0x182H	0x0042H
2.	Enable Counter which increments the current address so the next instruction can be fetched.	0x184H	0x1000H
3.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x188H	0x0010H
4.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x190H	0x0201H
5.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

b.) Reg. Y -> Reg. Z:

Steps for YTZ opcode (Instruction is fetched):

STEP	WORKING (Status)	ISA ADDRESS	ISA DATA
1.	Enable Reg. Y Output to Bus and enable Reg. Z Input (Load).	0x1C2H	0x0420H
2.	Enable Counter which increments the current address so the next instruction can be fetched.	0x1C4H	0x1000H
3.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x1C8H	0x0010H
4.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x1D0H	0x0201H
5.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*

*Data and Address is according to the next fetched instruction.

Further, Instructions under R2R – Data Transfer are working on same principle which are:

1.) XTZ
2.) YTZ
3.) YTX
4.) ZTY
5.) ZTX

7.) **Jumping (Subroutine):**

Jumping or subroutine to different addresses can be a solution for looping and function calling.

Steps for YTZ opcode (Instruction is fetched):

STEP	**WORKING (Status)**	**ISA ADDRESS**	**ISA DATA**
1.	Enable Higher bits (4-7) of Instruction register and enable binary counter input (load), this allows counter to jump to that address.	0x302H	0x4100H
2.	Enable Counter which increments the current address so the next instruction can be fetched.	0x304H	0x1000H
3.	Counter's Output enables and MAR's input (Load) is enabled, which allows main memory to move to next instruction.	0x308H	0x0010H
4.	Next instruction is now in Bus and ready to be fetched by Loaded by Instruction Register.	0x310H	0x0201H
5.	Instruction is fetched into IR and repeat the process for next instructions until it catches 0x0 opcode.	*	*