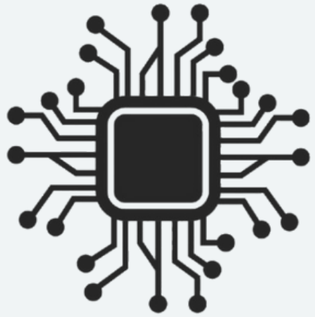
भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर
Indian Institute of Information Technology, Nagpur
An Institution of National Importance By An Act of Parliament

Dyna-85 v1

Intel 8085 inspired Dynamic Custom Processor



Anubhav Rathore | BT23ECE068

Hardware Description Languages

DATE: 14/11/2025

DYNA-85 V1

A dynamically parameterized 8-bit microprocessor inspired by 8085, implementing a unified control logic through field-based instruction decoding. Unlike extensive number of similar opcodes in 8085 for same operation, this processor is featured by compressed instruction space by parameterizing opcodes.

ANUBHAV RATHORE

MOTIVATION

Having studied Intel 8085 microprocessor closely, it appeared that:

1. **Redundant instructions for identical operations:** many similar instructions have distinct opcodes, only differing by their operands. Example, MOV A B and MOV B C
2. **Non-parameterized decoding:** 8085 uses hard coded bit patterns for opcodes, rather than efficient field-based decoding.
3. **Wasted Instruction Memory Space:** redundant instructions use unnecessary more space, which can be utilized for other important operations.
4. **Limited addressing capabilities:** direct and indirect addressing capabilities are weak. Example, LDA STA and such can be operated on Accumulator register only.
5. **High Accumulator Dependency:** most of the instructions use accumulator as default result storage unit, for ALU operations and more, which can be made rather more flexible.

Thus, these drawbacks in the architecture can be overcame by a much more efficient dynamic parameterized architecture based on field-based instruction decoding.,

SOLUTION

Parameterized Architecture based on Field-Based Decoding.

Say like a combination of CISC+RISC but completely independent architecture.

What does it means?

Example:

MOV opcode is passed as 8'b01DDDSSS

where 2'b01 (TWO MSB) is the MOV operation identifier in control unit.

3'bDDD - specifies destination register selected

3'bSSS - specifies source register selected

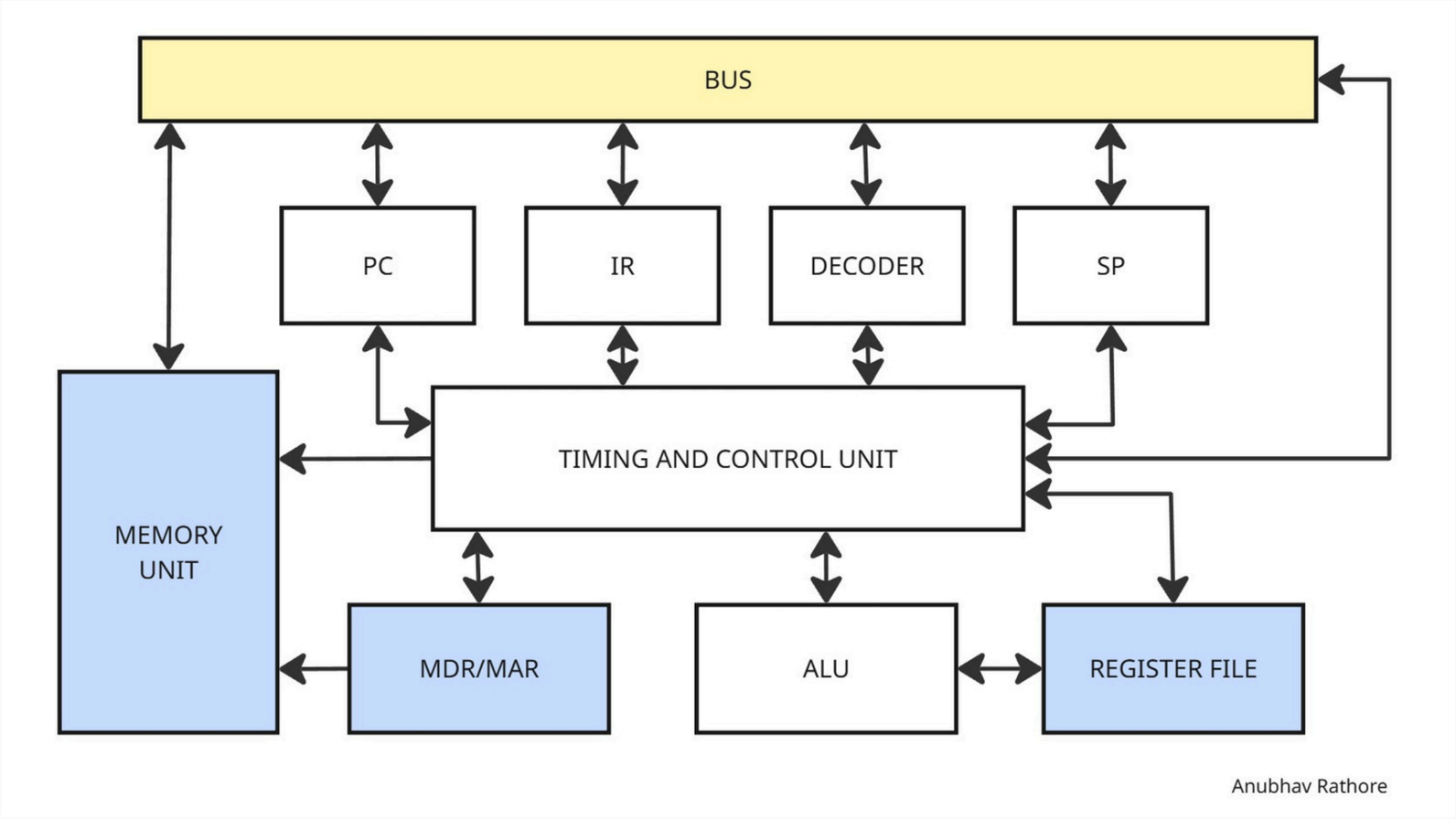
```
// Single-byte instruction formats (top 2 bits)
`define MOV_OP      2'b01

// Multi-byte instruction identifiers (top 5 bits)
`define LDA_OP      5'b00001
`define STA_OP      5'b00010
`define LXI_OP      5'b00011
`define JMP_OP      5'b00100

// ALU operations (for future use)
`define ALU_ADD     2'b00
`define ALU_SUB     2'b01
`define ALU_CMP     2'b10
`define ALU_INR     2'b11
```

Similarly optimized other operations like LDA, STA with addressing capabilities of using not just A, rather other internal registers as well. 5'b0001DDD OPERAND – DDD specifies destination register

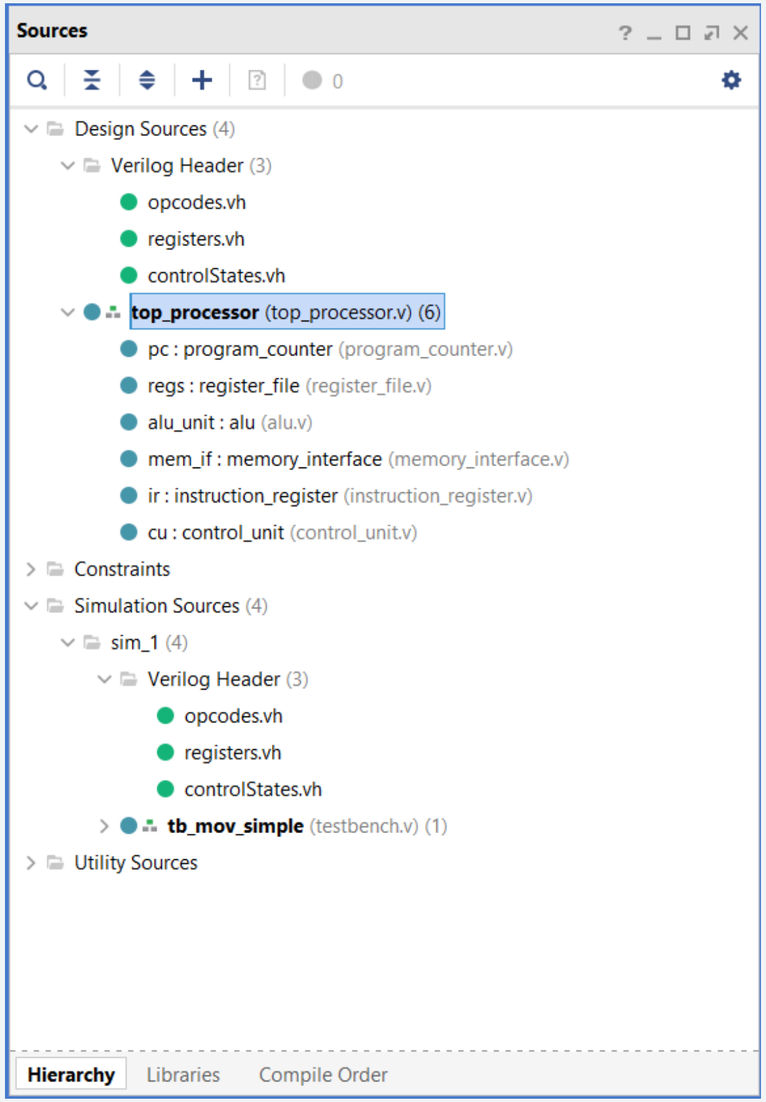
ARCHITECTURE

Parameterized Architecture based on Field-Based Decoding.

Say like a combination of CISC+RISC but completely independent architecture.



HEIRARCHY



TOP_PROCESSOR

The main module (wrapper) of all other important subsequent modules.
Instantiates and connects all modules, inter-relating their outputs and inputs

ANUBHAV RATHORE

MODULES

PROGRAM COUNTER

Keeps track of current instructions and opcode.
Feeds memory interface with program address

```
module program_counter(
    input clk,
    input pc_inc, pc_load, reset,
    input [15:0] jmp_add,
    output reg [15:0] pc_out
);
```

MODULES

MEMORY INTERFACE

Reads data in memory based on PC, then feeds instructions register or operands or other data based on control signals.

```
module memory_interface(
    input clk, rst,
    // Address sources
    input [15:0] pc_addr,
    input [15:0] hl_addr,
    // Data from registers (for stores)
    input [7:0] reg_data_out,
    // Control signals
    input mem_rd, mem_wr,
    input addr_sel,  // 0=PC, 1=HL
    // Outputs to processor
    output reg [7:0] mem_data_in,
    output reg [7:0] instruction_data,
    // External memory bus
    output reg [15:0] addr_bus,
    inout [7:0] data_bus,
    output reg rd_n, wr_n
);
```

MODULES

INSTRUCTION REG

Instructions register updates opcode based on control signals (load IR, etc.)
Keeps current opcodes for processor (DECODE stage).

```
module instruction_register(
    input clk, reset, ir_load,
    input [7:0] data_in,
    output reg [7:0] opcode,
    output reg ir_ready
);
```

MODULES

ALU

To perform arithmetic and logical operations, on operands
Generate flags based on results

```
module alu(
    input [7:0] a, b,
    input carry_in,
    input [3:0] alu_sel,
    output reg [7:0] result,
    output reg [3:0] flags
);
```

ANUBHAV RATHORE

MODULES

REGISTER FILE

Internal register bank of the processor.
8 bits wide - A B C D E H L
HL pairs as address holder also

registers.vh acts as global registers address file

```
module alu(
    input [7:0] a, b,
    input carry_in,
    input [3:0] alu_sel,
    output reg [7:0] result,
    output reg [3:0] flags
);
```

```
// General Purpose Registers
`define REG_A       3'b000
`define REG_B       3'b001
`define REG_C       3'b010
`define REG_D       3'b011
`define REG_E       3'b100
`define REG_H       3'b101
`define REG_L       3'b110
```

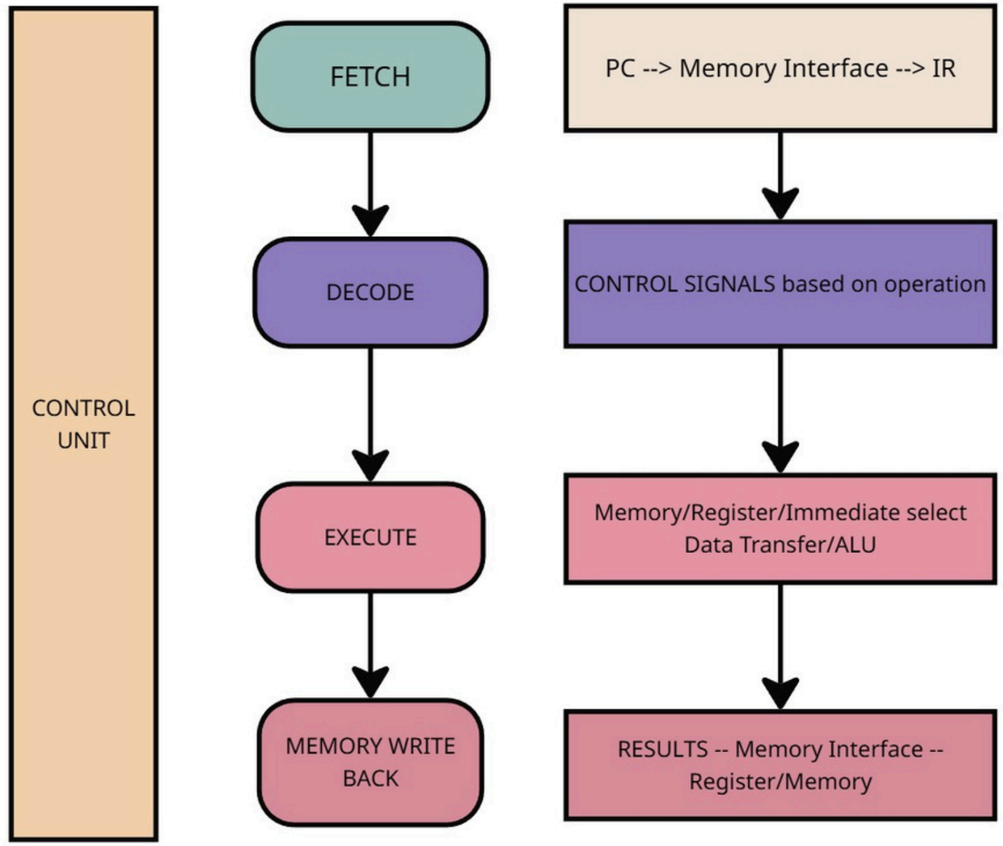
MODULES

CONTROL UNIT

Decodes opcodes and operands.
Generate control signals for other modules to update, execute and function.
Manages machine execution cycle - Fetch Decode Execute

```
module control_unit(
    input clk, reset,
    input [3:0] flags,
    input [7:0] opcode,

    // Program Counter control
    output reg pc_inc,
    output reg pc_load,
    output reg [15:0] jump_addr,

    // Instruction Register control
    output reg ir_load,

    // Register File control
    output reg reg_we,
    output reg [2:0] reg_read_sel,
    output reg [2:0] reg_write_sel,

    // ALU control
    output reg [3:0] alu_sel,
    output reg alu_carry_in,
    output reg alu_op1_sel,
    output reg alu_op2_sel,

    // Memory Interface control
    output reg mem_rd, mem_wr, addr_sel,

    // Temporary registers
    output reg temp_low_load, temp_high_load,
    output reg [7:0] temp_data,

    // Data path control
    output reg data_path_sel,

    output wire [3:0] state_debug
);
```

FLOW OF EXECUTION




```
// Main states
`define STATE_FETCH      4'b0000
`define STATE_DECODE     4'b0001

// Single-byte instruction states
`define STATE_MOV_EXEC   4'b0010

// Multi-byte instruction states
`define STATE_READ_LOW   4'b0100
`define STATE_READ_HIGH  4'b0101
`define STATE_LDA_DATA   4'b0110
`define STATE_STA_DATA   4'b0111
`define STATE_JMP_EXEC   4'b1000

`define STATE_HALT       4'b1111
```

TESTBENCH

```
=== Simple MOV Test ===
Monitoring bus activity...
Time=35000: FETCH addr=0x0000, opcode=0x41
Time=55000: FETCH addr=0x0001, opcode=0x4a
Time=75000: FETCH addr=0x0002, opcode=0x5c
Time=95000: FETCH addr=0x0003, opcode=0x00
Time=115000: FETCH addr=0x0004, opcode=0x00
Time=135000: FETCH addr=0x0005, opcode=0x00
Time=155000: FETCH addr=0x0006, opcode=0x00
Time=175000: FETCH addr=0x0007, opcode=0x00
Time=195000: FETCH addr=0x0008, opcode=0x00
Time=215000: FETCH addr=0x0009, opcode=0x00
Test completed successfully!
```

RESULT

Thus, this is dynamic parameterized processor inspired by 8085.
The instructions decoding and execution, along with dynamic opcodes are the key feature making the processor novel and different.
v1 in Dyna-85 v1 stands for version 1 build, has only MOV instruction right now and more are being added and scaled.

THANK YOU

ANUBHAV RATHORE