A green and yellow logo

Description automatically generatedA close-up of a logo

Description automatically generated

**SAP-3 Processor Core**

**Specification Sheet**

Abderahman Mohamed Khalil V23010331

Ahmed Kamal V23010450

Mohaned Tarek V23010449

Mohamed Gamal V23010636

**Delivered to Dr. Islam Yehia**

Contents

[**Processor Overview** 3](#_Toc154016853)

[Specifications 3](#_Toc154016854)

[Programming model 4](#_Toc154016855)

[Hardware details 5](#_Toc154016856)

[**Memory** 6](#_Toc154016857)

[**ALU** 6](#_Toc154016858)

[**Register File** 7](#_Toc154016859)

[**Clock Controller** 7](#_Toc154016860)

[**Output Controller** 7](#_Toc154016861)

[**Controller and instruction register** 8](#_Toc154016862)

[**Peripheral interface** 8](#_Toc154016863)

[**Instruction set architecture (ISA)** 8](#_Toc154016864)

[**Instructions details** 9](#_Toc154016865)

[**Timing Diagram** 14](#_Toc154016866)

[**Area Analysis** 14](#_Toc154016867)

[**Testing Plan** 15](#_Toc154016868)

[Testing Each Module 15](#_Toc154016869)

[Testing the main instructions required 15](#_Toc154016870)

[Programming Examples 15](#_Toc154016871)

[Addition code 15](#_Toc154016872)

[Stack Testing Code 16](#_Toc154016873)

# **Processor Overview**

## Specifications

* **8-bit Simple Processor with SAP-3 Core**
  + Runs at **700 MHz** as a maximum frequency
  + Includes **basic arithmetic and logical** operations
  + Supports **conditional and unconditional branching** instruction
  + Supports **extended word size** operations (16-bit)
* **I/O**
  + It has I/Os to be connected to 16-bit address bus and 8-bit data bus
* **Low power**
  + Has low standby power consumption due to clock gating that cuts the clock after the execution end of the program
* **Memory**
  + Has **64kb RAM** memory
  + Follows **Von Neumann** architecture (instruction and data in the same memory)

## Programming model

The processor core has 8 registers with 8-bit width and 2 registers with 16-bits word width. The 8-bit registers are: B,C,D,E,H,L,W and Z. In extended instructions, each pair can be treated as one register to form 4 registers with 16-bit word size. H,L has special task when direct addressing mode is used which is storing the address mentioned in the instruction during a direct addressing mode. The PC (Program Counter) register is used to store the location of the next instruction. In basic designs, the PC is found outside the register file as an independent module, in our design, it is included in the register file to reduce control complexity and reduce control lines. The SP (Stack Pointer) register is used to store the last data location that is pushed to the stack memory.

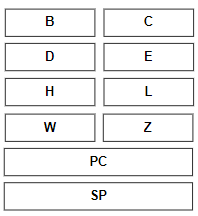


Figure 1 Programming model of the designed processor

## Hardware details

The design of the processor follows Von Neumann architecture. As a result, all sub blocks of the processor communicate through single 16-bit bus transferring both address and data as shown in figure 2. The controlling lines forms 35-bit word size controlled by the processor controller.

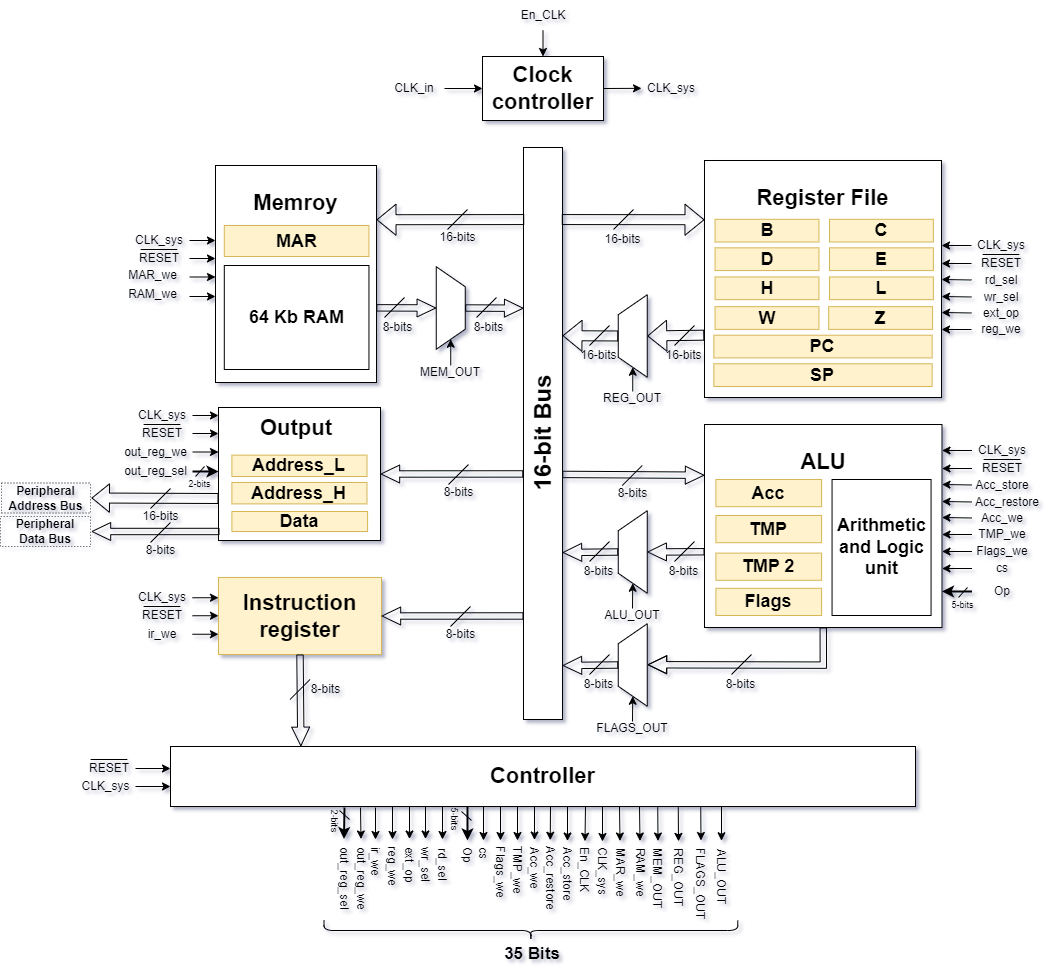


Figure 2 Processor Core Overall Block Diagram

### **Memory**

**I/O:**

* **input** clk
* **input** rst
* **input** mar\_we
* **input** ram\_we
* **input** bus
* **output** out

### **ALU**

**I/O:**

* **input** clk
* **input** rst
* **input** a\_store
* **input** a\_restore
* **input** a\_we
* **input** tmp\_we
* **input** flags\_we
* **input** cs
* **input** op
* **input** bus
* **output** flags
* **output** out

### **Register File**

**I/O:**

* **input** clk
* **input** rst
* **input** rd\_sel
* **input** wr\_sel
* **input** ext
* **input** we
* **input** data\_in
* **output** data\_out

### **Clock Controller**

**I/O:**

* **input** clk\_in
* **input** hlt
* **output** clk\_out

### **Output Controller**

**I/O:**

* **input** clk
* **input** rst
* **input** reg\_we
* **input** reg\_sel
* **input** bus
* **output** out\_address\_bus
* **output** out\_data\_bus

### **Controller and instruction register**

**I/O:**

* **input** clk,
* **input** rst
* **input** opcode
* **input** flags
* **output** reg ctrl\_word

# **Peripheral interface**

The processor can be connected to other peripheral to extend its functions or to communicate to outside world through the output controller. The output controller depends on the concept of address based i/o which means there will be address bus and data bus, and the peripheral with the address putted in the bus will be able to receive the data in the data bus. The address width is 16-bit word which means that 65535 peripherals can be connected to the processor.

# **Instruction set architecture (ISA)**

The ISA of the processor follows exactly the ISA of 8085

Flags abbreviation:

**S:** Sign Flag

**Z:** Zero Flag

**P:** Parity Flag

**C:** Carry Flag

## **Instructions details**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Opcode | Bytes | Flags affected | Clock cycles | Function | Addressing mode |
| Arithmetic instructions | | | | | | | |
| Increment and decrement instructions | | | | | | | |
| INR A | 3C | 1 | SZP- | 4 | A = A + 1 | Register |
| INR B | 04 | 1 | SZP- | 6 | B = B + 1 | Register |
| INR C | 0C | 1 | SZP- | 6 | C = C + 1 | Register |
| INR D | 14 | 1 | SZP- | 6 | D = D + 1 | Register |
| INR E | 1C | 1 | SZP- | 6 | E = E + 1 | Register |
| INR H | 24 | 1 | SZP- | 6 | H = H + 1 | Register |
| INR L | 2C | 1 | SZP- | 6 | L = L + 1 | Register |
| INR M | 34 | 1 | SZP- | 7 | [HL] = [HL] + 1 | Direct |
| DCR A | 3D | 1 | SZP- | 4 | A = A - 1 | Register |
| DCR B | 05 | 1 | SZP- | 6 | B = B - 1 | Register |
| DCR C | 0D | 1 | SZP- | 6 | C = C - 1 | Register |
| DCR D | 15 | 1 | SZP- | 6 | D = D - 1 | Register |
| DCR E | 1D | 1 | SZP- | 6 | E = E - 1 | Register |
| DCR H | 25 | 1 | SZP- | 6 | H = H - 1 | Register |
| DCR L | 2D | 1 | SZP- | 6 | L = L - 1 | Register |
| DCR M | 35 | 1 | SZP- | 7 | [HL] = [HL] - 1 | Direct |
| INX B | 03 | 1 | ---- | 4 | BC = BC + 1 | Register |
| INX D | 13 | 1 | ---- | 4 | DE = DE + 1 | Register |
| INX H | 23 | 1 | ---- | 4 | HL = HL + 1 | Register |
| INX SP | 33 | 1 | ---- | 4 | SP = SP + 1 | Register |
| DCX B | 0B | 1 | ---- | 4 | BC = BC - 1 | Register |
| DCX D | 1B | 1 | ---- | 4 | DE = DE - 1 | Register |
| DCX H | 2B | 1 | ---- | 4 | HL = HL - 1 | Register |
| DCX SP | 3B | 1 | ---- | 4 | SP = SP - 1 | Register |
| DAD B | 09 | 1 | ---C | 12 | HL = HL + BC | Register |
| DAD D | 19 | 1 | ---C | 12 | HL = HL + DE | Register |
| DAD H | 29 | 1 | ---C | 12 | HL = HL + HL | Register |
| DAD SP | 39 | 1 | ---C | 12 | HL = HL + SP | Register |
| Addition | | | | | | | |
| ADD A | 87 | 1 | SZPC | 4 | A = A + A | Register |
| ADD B | 80 | 1 | SZPC | 5 | A = A + B | Register |
| ADD C | 81 | 1 | SZPC | 5 | A = A + C | Register |
| ADD D | 82 | 1 | SZPC | 5 | A = A + D | Register |
| ADD E | 83 | 1 | SZPC | 5 | A = A + E | Register |
| ADD H | 84 | 1 | SZPC | 5 | A = A + H | Register |
| ADD L | 85 | 1 | SZPC | 5 | A = A + L | Register |
| ADD M | 86 | 1 | SZPC | 6 | A = A + [HL] | Direct |
| ADI byte | C6 | 2 | SZPC | 6 | A = A + byte | Immediate |
| ADC A | 8F | 1 | SZPC | 4 | A = A + A + FlagC | Register |
| ADC B | 88 | 1 | SZPC | 5 | A = A + B + FlagC | Register |
| ADC C | 89 | 1 | SZPC | 5 | A = A + C + FlagC | Register |
| ADC D | 8A | 1 | SZPC | 5 | A = A + D + FlagC | Register |
| ADC E | 8B | 1 | SZPC | 5 | A = A + E + FlagC | Register |
| ADC H | 8C | 1 | SZPC | 5 | A = A + H + FlagC | Register |
| ADC L | 8D | 1 | SZPC | 5 | A = A + L + FlagC | Register |
| ADC M | 8E | 1 | SZPC | 6 | A = A + [HL] + FlagC | Direct |
| ACI byte | CE | 2 | SZPC | 6 | A = A + byte + FlagC | Immediate |
| Subtraction | | | | | | | |
| SUB A | 97 | 1 | SZPC | 4 | A = A - A | Register |
| SUB B | 90 | 1 | SZPC | 5 | A = A - B | Register |
| SUB C | 91 | 1 | SZPC | 5 | A = A - C | Register |
| SUB D | 92 | 1 | SZPC | 5 | A = A - D | Register |
| SUB E | 93 | 1 | SZPC | 5 | A = A - E | Register |
| SUB H | 94 | 1 | SZPC | 5 | A = A - H | Register |
| SUB L | 95 | 1 | SZPC | 5 | A = A - L | Register |
| SUB M | 96 | 1 | SZPC | 6 | A = A - [HL] | Direct |
| SUI byte | D6 | 2 | SZPC | 6 | A = A - byte | Immediate |
| SBB A | 9F | 1 | SZPC | 4 | A = A - A - FlagC | Register |
| SBB B | 98 | 1 | SZPC | 5 | A = A - B - FlagC | Register |
| SBB C | 99 | 1 | SZPC | 5 | A = A - C - FlagC | Register |
| SBB D | 9A | 1 | SZPC | 5 | A = A - D - FlagC | Register |
| SBB E | 9B | 1 | SZPC | 5 | A = A - E- FlagC | Register |
| SBB H | 9C | 1 | SZPC | 5 | A = A - H - FlagC | Register |
| SBB L | 9D | 1 | SZPC | 5 | A = A - L - FlagC | Register |
| SBB M | 9E | 1 | SZPC | 6 | A = A – [HL] - FlagC | Direct |
| SBI byte | DE | 2 | SZPC | 6 | A = A - byte - FlagC | Immediate |
| Logical operation instructions | | | | | | | |
| AND | | | | | | | |
| ANA A | A7 | 1 | SZPC | 4 | A = A and A | Register |
| ANA B | A0 | 1 | SZPC | 5 | A = A and B | Register |
| ANA C | A1 | 1 | SZPC | 5 | A = A and C | Register |
| ANA D | A2 | 1 | SZPC | 5 | A = A and D | Register |
| ANA E | A3 | 1 | SZPC | 5 | A = A and E | Register |
| ANA H | A4 | 1 | SZPC | 5 | A = A and H | Register |
| ANA L | A5 | 1 | SZPC | 5 | A = A and L | Register |
| ANA M | A6 | 1 | SZPC | 6 | A = A and [HL] | Direct |
| ANI byte | E6 | 2 | SZPC | 6 | A = A and byte | Immediate |
| OR | | | | | | |
| ORA A | B7 | 1 | SZPC | 4 | A = A or A | Register |
| ORA B | B0 | 1 | SZPC | 5 | A = A or B | Register |
| ORA C | B1 | 1 | SZPC | 5 | A = A or C | Register |
| ORA D | B2 | 1 | SZPC | 5 | A = A or D | Register |
| ORA E | B3 | 1 | SZPC | 5 | A = A or E | Register |
| ORA H | B4 | 1 | SZPC | 5 | A = A or H | Register |
| ORA L | B5 | 1 | SZPC | 5 | A = A or L | Register |
| ORA M | B6 | 1 | SZPC | 6 | A = A or [HL] | Direct |
| ORI byte | F6 | 2 | SZPC | 6 | A = A or byte | Immediate |
| XOR | | | | | | |
| XRA A | AF | 1 | SZPC | 4 | A = A xor A | Register |
| XRA B | A8 | 1 | SZPC | 5 | A = A xor B | Register |
| XRA C | A9 | 1 | SZPC | 5 | A = A xor C | Register |
| XRA D | AA | 1 | SZPC | 5 | A = A xor D | Register |
| XRA E | AB | 1 | SZPC | 5 | A = A xor E | Register |
| XRA H | AC | 1 | SZPC | 5 | A = A xor H | Register |
| XRA L | AD | 1 | SZPC | 5 | A = A xor L | Register |
| XRA M | AE | 1 | SZPC | 6 | A = A xor [HL] | Direct |
| XRI byte | EE | 2 | SZPC | 6 | A = A xor byte | Immediate |
| Shift and Rotate | | | | | | |
| RLC | 07 | 1 | ---C | 4 | Shift A left, FlagC = A[7] | Register |
| RAL | 17 | 1 | ---C | 4 | Shift A left, shift FlagC into A[0] | Register |
| RAR | 1F | 1 | ---C | 4 | Shift A right, shift FlagC into A[7] | Register |
| RRC | 0F | 1 | ---C | 4 | Shift A right, FlagC = A[0] | Register |
| Comparison instructions | | | | | | |
| Control Carry Flag | | | | | | |
| CMA | 2F | 1 | ---- | 4 | A = ~A | Register |
| STC | 37 | 1 | ---C | 4 | FlagC = 1 | - |
| CMC | 3F | 1 | ---C | 4 | FlagC = ~FlagC | - |
| Compare | | | | | | |
| CMP A | BF | 1 | SZPC | 4 | FlagZ = 1 if A == A | Register |
| CMP B | B8 | 1 | SZPC | 5 | FlagZ = 1 if A == B | Register |
| CMP C | B9 | 1 | SZPC | 5 | FlagZ = 1 if A == C | Register |
| CMP D | BA | 1 | SZPC | 5 | FlagZ = 1 if A == D | Register |
| CMP E | BB | 1 | SZPC | 5 | FlagZ = 1 if A == E | Register |
| CMP H | BC | 1 | SZPC | 5 | FlagZ = 1 if A == H | Register |
| CMP L | BD | 1 | SZPC | 5 | FlagZ = 1 if A == L | Register |
| CMP M | BE | 1 | SZPC | 6 | FlagZ = 1 if A == [HL] | Direct |
| CPI byte | FE | 2 | ---- | 6 | FlagZ = 1 if A == byte | Immediate |
| Data Movement instructions | | | | | | |
| Load & Extended Load & Store | | | | | | |
| LDA addr | 3A | 3 | ---- | 11 | Load A with [addr] | Direct |
| LXI B, dble | 01 | 3 | ---- | 10 | Load BC with dble | Immediate |
| LXI D, dble | 11 | 3 | ---- | 10 | Load DE with dble | Immediate |
| LXI H, dble | 21 | 3 | ---- | 10 | Load HL with dble | Immediate |
| LXI SP, dble | 31 | 3 | ---- | 10 | Load SP with dble | Immediate |
| STA addr | 32 | 3 | ---- | 11 | Store A at [addr] | Direct |
| LHLD addr | 2A | 3 | ---- | 14 | Load HL with [addr] | Direct |
| SHLD addr | 22 | 3 | ---- | 14 | Store HL at [addr] | Direct |
| Move | | | | | | |
| MOV A, A | 7F | 1 | ---- | 4 | A = A | Register |
| MOV A, B | 78 | 1 | ---- | 4 | A = B | Register |
| MOV A, C | 79 | 1 | ---- | 4 | A = C | Register |
| MOV A, D | 7A | 1 | ---- | 4 | A = D | Register |
| MOV A, E | 7B | 1 | ---- | 4 | A = E | Register |
| MOV A, H | 7C | 1 | ---- | 4 | A = H | Register |
| MOV A, L | 7D | 1 | ---- | 4 | A = L | Register |
| MOV A, M | 7E | 1 | ---- | 5 | A = [HL] | Register |
| MOV B, A | 47 | 1 | ---- | 4 | B = A | Register |
| MOV B, B | 40 | 1 | ---- | 4 | B = B | Register |
| MOV B, C | 41 | 1 | ---- | 4 | B = C | Register |
| MOV B, D | 42 | 1 | ---- | 4 | B = D | Register |
| MOV B, E | 43 | 1 | ---- | 4 | B = E | Register |
| MOV B, H | 44 | 1 | ---- | 4 | B = H | Register |
| MOV B, L | 45 | 1 | ---- | 4 | B = L | Register |
| MOV B, M | 46 | 1 | ---- | 5 | B = [HL] | Register |
| MOV C, A | 4F | 1 | ---- | 4 | C = A | Register |
| MOV C, B | 48 | 1 | ---- | 4 | C = B | Register |
| MOV C, C | 49 | 1 | ---- | 4 | C = C | Register |
| MOV C, D | 4A | 1 | ---- | 4 | C = D | Register |
| MOV C, E | 4B | 1 | ---- | 4 | C = E | Register |
| MOV C, H | 4C | 1 | ---- | 4 | C = H | Register |
| MOV C, L | 4D | 1 | ---- | 4 | C = L | Register |
| MOV C, M | 4E | 1 | ---- | 5 | C = [HL] | Register |
| MOV D, A | 57 | 1 | ---- | 4 | D = A | Register |
| MOV D, B | 50 | 1 | ---- | 4 | D = B | Register |
| MOV D, C | 51 | 1 | ---- | 4 | D = C | Register |
| MOV D, D | 52 | 1 | ---- | 4 | D = D | Register |
| MOV D, E | 53 | 1 | ---- | 4 | D = E | Register |
| MOV D, H | 54 | 1 | ---- | 4 | D = H | Register |
| MOV D, L | 55 | 1 | ---- | 4 | D = L | Register |
| MOV D, M | 56 | 1 | ---- | 5 | D = [HL] | Register |
| MOV E, A | 5F | 1 | ---- | 4 | E = A | Register |
| MOV E, B | 58 | 1 | ---- | 4 | E = B | Register |
| MOV E, C | 59 | 1 | ---- | 4 | E = C | Register |
| MOV E, D | 5A | 1 | ---- | 4 | E = D | Register |
| MOV E, E | 5B | 1 | ---- | 4 | E = E | Register |
| MOV E, H | 5C | 1 | ---- | 4 | E = H | Register |
| MOV E, L | 5D | 1 | ---- | 4 | E = L | Register |
| MOV E, M | 5E | 1 | ---- | 5 | E = [HL] | Register |
| MOV H, A | 67 | 1 | ---- | 4 | H = A | Register |
| MOV H, B | 60 | 1 | ---- | 4 | H = B | Register |
| MOV H, C | 61 | 1 | ---- | 4 | H = C | Register |
| MOV H, D | 62 | 1 | ---- | 4 | H = D | Register |
| MOV H, E | 63 | 1 | ---- | 4 | H = E | Register |
| MOV H, H | 64 | 1 | ---- | 4 | H = H | Register |
| MOV H, L | 65 | 1 | ---- | 4 | H = L | Register |
| MOV H, M | 66 | 1 | ---- | 5 | H = [HL] | Register |
| MOV L, A | 6F | 1 | ---- | 4 | L = A | Register |
| MOV L, B | 68 | 1 | ---- | 4 | L = B | Register |
| MOV L, C | 69 | 1 | ---- | 4 | L = C | Register |
| MOV L, D | 6A | 1 | ---- | 4 | L = D | Register |
| MOV L, E | 6B | 1 | ---- | 4 | L = E | Register |
| MOV L, H | 6C | 1 | ---- | 4 | L = H | Register |
| MOV L, L | 6D | 1 | ---- | 4 | L = L | Register |
| MOV L, M | 6E | 1 | ---- | 5 | L = [HL] | Register |
| MOV M, A | 77 | 1 | ---- | 5 | [HL] = A | Direct |
| MOV M, B | 70 | 1 | ---- | 5 | [HL] = B | Direct |
| MOV M, C | 71 | 1 | ---- | 5 | [HL] = C | Direct |
| MOV M, D | 72 | 1 | ---- | 5 | [HL] = D | Direct |
| MOV M, E | 73 | 1 | ---- | 5 | [HL] = E | Direct |
| MOV M, H | 74 | 1 | ---- | 5 | [HL] = H | Direct |
| MOV M, L | 75 | 1 | ---- | 5 | [HL] = L | Direct |
| Move immediate | | | | | | |
| MVI A, byte | 3E | 2 | ---- | 6 | A = byte | Immediate |
| MVI B, byte | 06 | 2 | ---- | 6 | B = byte | Immediate |
| MVI C, byte | 0E | 2 | ---- | 6 | C = byte | Immediate |
| MVI D, byte | 16 | 2 | ---- | 6 | D = byte | Immediate |
| MVI E, byte | 1E | 2 | ---- | 6 | E = byte | Immediate |
| MVI H, byte | 26 | 2 | ---- | 6 | H = byte | Immediate |
| MVI L, byte | 2E | 2 | ---- | 6 | L = byte | Immediate |
| MVI M, byte | 36 | 2 | ---- | 8 | [HL] = byte | Immediate |
| Stack instructions | | | | | | |
| PUSH | | | | | | |
| PUSH B | C5 | 1 | ---- | 9 | Push value in BC onto the stack | - |
| PUSH D | D5 | 1 | ---- | 9 | Push value in DE onto the stack | - |
| PUSH H | E5 | 1 | ---- | 9 | Push value in HL onto the stack | - |
| PUSH PSW | F5 | 1 | ---- | 9 | Push value in AF onto the stack | - |
| POP | | | | | | |
| POP B | C1 | 1 | ---- | 9 | Pop value on stack into BC | - |
| POP D | D1 | 1 | ---- | 9 | Pop value on stack into DE | - |
| POP H | E1 | 1 | ---- | 9 | Pop value on stack into HL | - |
| POP PSW | F1 | 1 | ---- | 9 | Pop value on stack into AF | - |
| Routines instructions | | | | | | |
| CALL | | | | | | |
| CALL addr | CD | 3 | ---- | 16 | Call function at addr | Direct |
| Return | | | | | | |
| RET | C9 | 1 | ---- | 4 | Return from function | - |
| Jump | | | | | | |
| JMP addr | C3 | 3 | ---- | 4 | Jump to addr | Direct |
| JP addr | F2 | 3 | ---- | 4/9 | Jump to addr if FlagS == 0 | Direct |
| JM addr | FA | 3 | ---- | 4/9 | Jump to addr if FlagS == 1 | Direct |
| JNZ addr | C2 | 3 | ---- | 4/9 | Jump to addr if FlagZ == 0 | Direct |
| JZ addr | CA | 3 | ---- | 4/9 | Jump to addr if FlagZ == 1 | Direct |
| JPO addr | E2 | 3 | ---- | 4/9 | Jump to addr if FlagP == 0 | Direct |
| JPE addr | EA | 3 | ---- | 4/9 | Jump to addr if FlagP == 1 | Direct |
| JNC addr | D2 | 3 | ---- | 4/9 | Jump to addr if FlagC == 0 | Direct |
| JC addr | DA | 3 | ---- | 4/9 | Jump to addr if FlagC == 1 | Direct |
| Control instruction | | | | | | |
| NOP | 00 | 1 | ---- | 4 | Do nothing | - |
| HLT | 76 | 1 | ---- | 4 | Halt execution | - |
| OUT *addr* | D3 | 3 | ---- | 10 | Outputs A to *addr* peripheral | - |

## **Timing Diagram**

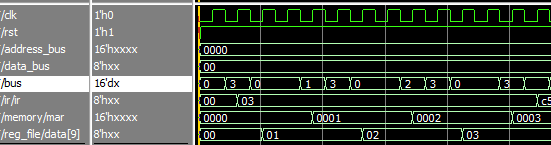


Figure Fetching cycle

# **Area Analysis**

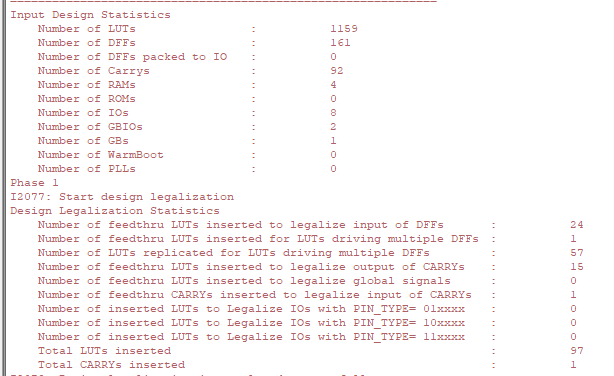


Figure Area analysis of the processor

# **Testing Plan**

To test our processor, three main tests were conducted to make sure it works properly:

## Testing Each Module

Each module is tested through simulation in order to make sure that its hardware is functioning properly, and the error is not from the hardware.

## Testing the main instructions required

Main instruction required from the proposal is tested that it functions properly such as arithmetic, logic, stack and branching operations. The rest is tested through programs.

## Programming Examples

To make sure that all the processor works in harmony, 2 simple programs (Addition program and stack testing program) were written to test some of the rest of the instructions.

### Addition code

**MVI** A, 08

**MVI** B, 04

**ADD** B

**OUT** 02

**HLT**

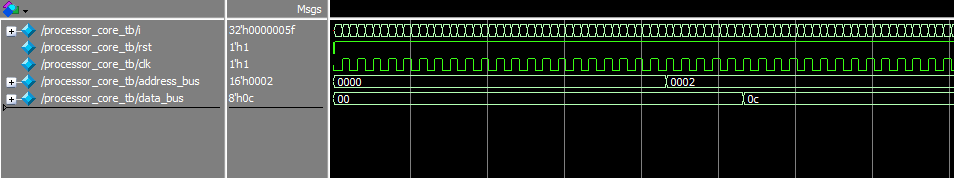


Figure 5 addition program results

### Stack Testing Code

**INX** B

**INX** B

**INX** B

**PUSH** B

**POP** D

**MOV** A, E

**OUT** 20

