SAP-3 Processor Core Design and Implementation

Abderahman Mohamed Khalil	V23010331
Ahmed Kamal	V23010450
Mohaned Tarek	V23010449
Mohamed Gamal	V23010636

Delivered to Dr. Islam Yehia

Contents

Introduction

The project discusses the design of simple as possible (SAP) 3 processor. The instruction set of the processor is similar to 8088-8086 but in a smaller version. Additional hardware is added to the main processor core to communicate with external peripherals. The specification sheet shows the processor specifications, details, instructions with timing diagram, examples of codes to be run by the processor and finally further development that intended to be made for the processor.

Design plan and team plan

The plan is to divide the design into sub-blocks that can be designed independently and then integrate them and decide how each block will be tested. Before each design, the team must meet and agree on the inputs and outputs of each block and how the rhythm will flow.

Here is the team plan:

- Abdelrahman Khalil: the instruction register and the controller, clock, writing the testing assembly code.
- Mohaned: the ALU
- Ahmed Kamal: Register file, memory
- Mohamed Gamal: output controller, clock controller

Blocks details

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

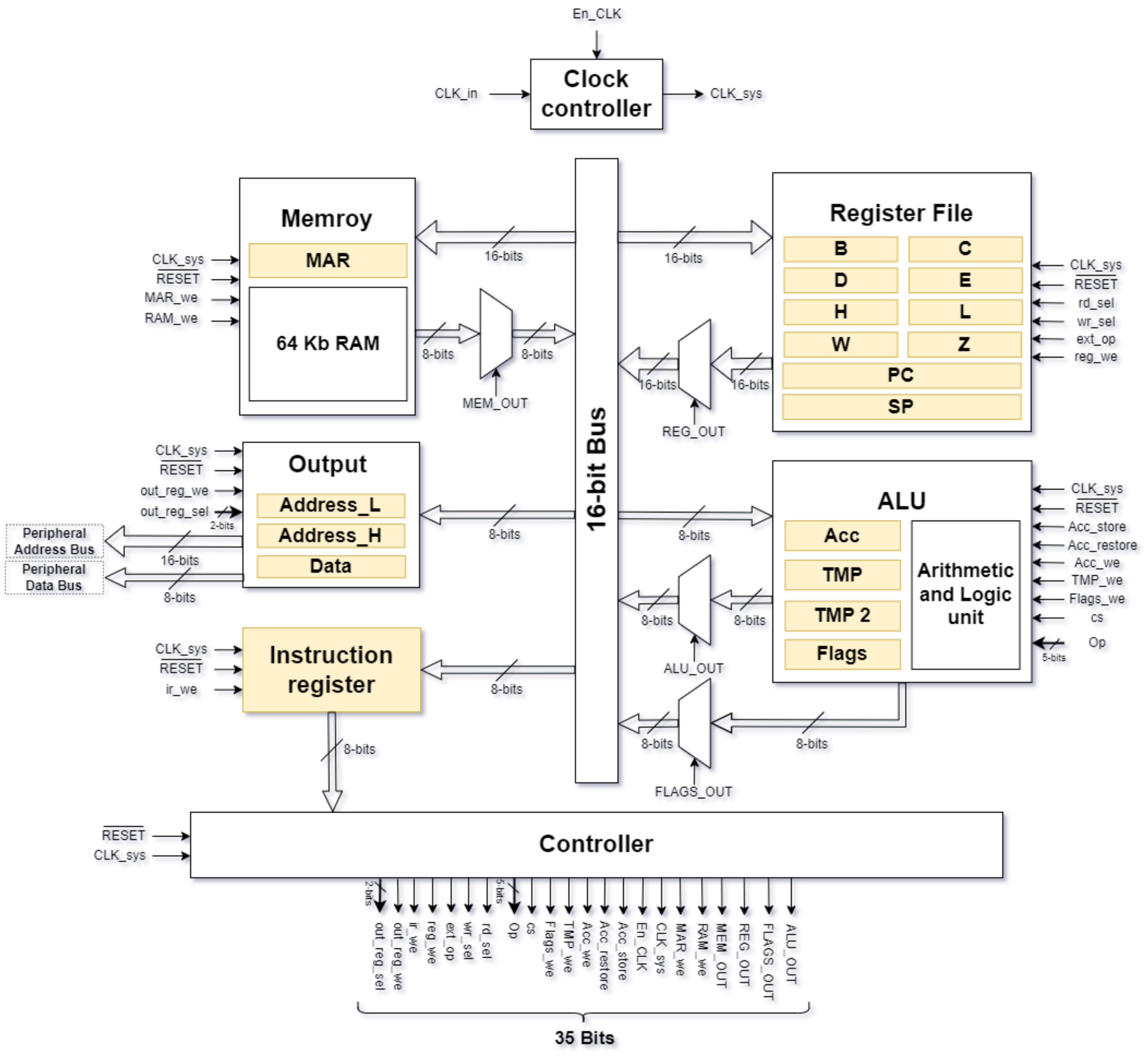


Figure 1 Processor Core Overall Block Diagram

Memory

Function Brief:

The ram memory of the processor is 64Kb size with 16-bit addressing size. The address data that comes from the bus is buffered into the memory address register (MAR). controller controls this block using write enable and external multiplexer to control the flow of the data into the MAR.

ALU

Function Brief:

ALU which is the brain of the processor, can perform several arithmetic and logical operations including:

- 8-bit addition
- Addition with carry
- 8-bit subtraction
- Subtraction with borrow
- AND, OR, Complement, XOR
- Rotate right and left
- Rotate right and left with carry
- Quick increment and decrement by 1

Also provide control to set or reset the carry. The ALU has three main registers which are:

- Accumulator
  - The main register of the ALU and stores the output of the ALU and its state is represented by the flags register
  - All single operand operations like increment and decrement operates on it
  - Can be modified by the ALU or the bus
- Temporary register
  - The second operand of the ALU
  - Can be modified from the bus only
  - Gives flexibility to do more operations (Add other registers to the Accumulator)

- Temporary register 2
    - This register is modified by the controller only and the end user has no control over it.
    - Stores only the value of the Accumulator during some instructions that operate with the value of the Accumulator without affecting the value of the Accumulator.

Register File

Function Brief:

The register file includes all the registers mentioned in the programming model. The register file has an internal increment and decrement circuit to increment and decrement frequently used registers such as PC or SP or operate on other registers with simple control steps and without affecting the flags.

Clock Controller

Function Brief:

This block is used for clock gating after the program finishes. In further development this block won't cut the clock after the program ends because there will be an operating system but will cut it during idle state.

Output Controller

Function Brief:

This block enables the processor to communicate with the other peripherals and with the outside world.

Controller and instruction register

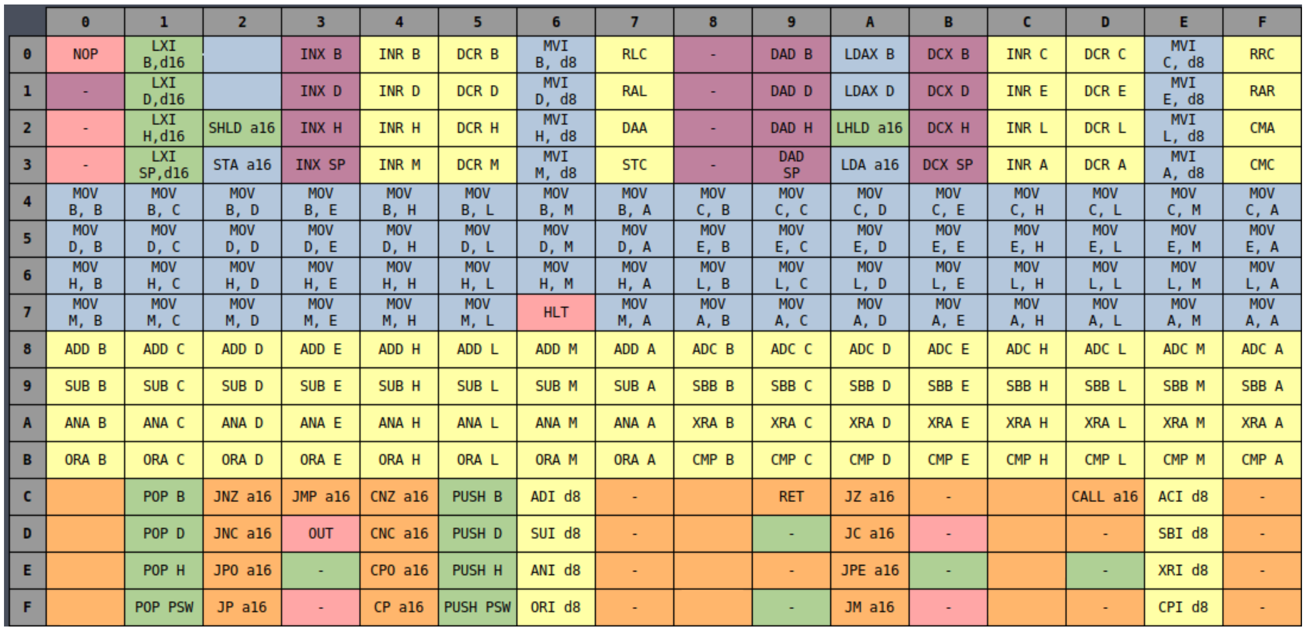
Function Brief:

The Instruction register stored the instruction to be executed. The controller decodes the instruction stored in the instruction register and controls all the other blocks of the processor through the control lines (called controlled word).

Control unit and instruction decoding

The control unit took most of the effort. The use of ”casez” was very powerful since it provided the ability to group multiple of similar instruction in the same case.

We found the idea on the internet and this image which shows a summary for 8085 instructions in compact way.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B,d16		INX B	INR B	DCR B	MVI B, d8	RLC	-	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C, d8	RRC
1	-	LXI D,d16		INX D	INR D	DCR D	MVI D, d8	RAL	-	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E, d8	RAR
2	-	LXI H,d16	SHLD a16	INX H	INR H	DCR H	MVI H, d8	DAA	-	DAD H	LHLD a16	DCX H	INR L	DCR L	MVI L, d8	CMA
3	-	LXI SP,d16	STA a16	INX SP	INR M	DCR M	MVI M, d8	STC	-	DAD SP	LDA a16	DCX SP	INR A	DCR A	MVI A, d8	CMC
4	MOV B, B	MOV B, C	MOV B, D	MOV B, E	MOV B, H	MOV B, L	MOV B, M	MOV B, A	MOV C, B	MOV C, C	MOV C, D	MOV C, E	MOV C, H	MOV C, L	MOV C, M	MOV C, A
5	MOV D, B	MOV D, C	MOV D, D	MOV D, E	MOV D, H	MOV D, L	MOV D, M	MOV D, A	MOV E, B	MOV E, C	MOV E, D	MOV E, E	MOV E, H	MOV E, L	MOV E, M	MOV E, A
6	MOV H, B	MOV H, C	MOV H, D	MOV H, E	MOV H, H	MOV H, L	MOV H, M	MOV H, A	MOV L, B	MOV L, C	MOV L, D	MOV L, E	MOV L, H	MOV L, L	MOV L, M	MOV L, A
7	MOV M, B	MOV M, C	MOV M, D	MOV M, E	MOV M, H	MOV M, L	HLT	MOV M, A	MOV A, B	MOV A, C	MOV A, D	MOV A, E	MOV A, H	MOV A, L	MOV A, M	MOV A, A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C		POP B	JNZ a16	JMP a16	CNZ a16	PUSH B	ADI d8	-		RET	JZ a16	-		CALL a16	ACI d8	-
D		POP D	JNC a16	OUT	CNC a16	PUSH D	SUI d8	-		-	JC a16	-		-	SBI d8	-
E		POP H	JPO a16	-	CPO a16	PUSH H	ANI d8	-		-	JPE a16	-		-	XRI d8	-
F		POP PSW	JP a16	-	CP a16	PUSH PSW	ORI d8	-		-	JM a16	-		-	CPI d8	-

We also found that octal representation is much easier in Verilog so we followed the next table

	0	1	2	3	4	5	6	7
00	NOP	LXI B,d16	STAX B	INX B	INR B	DCR B	MVI B, d8	RLC
01	-	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C, d8	RRC
02	-	LXI D,d16	STAX D	INX D	INR D	DCR D	MVI D, d8	RAL
03	-	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E, d8	RAR
04	-	LXI H,d16	SHLD a16	INX H	INR H	DCR H	MVI H, d8	DAA
05	-	DAD H	LHLD a16	DCX H	INR L	DCR L	MVI L, d8	CMA
06	-	LXI SP,d16	STA a16	INX SP	INR M	DCR M	MVI M, d8	STC
07	-	DAD SP	LDA a16	DCX SP	INR A	DCR A	MVI A, d8	CMC
10	MOV B, B	MOV B, C	MOV B, D	MOV B, E	MOV B, H	MOV B, L	MOV B, M	MOV B, A
11	MOV C, B	MOV C, C	MOV C, D	MOV C, E	MOV C, H	MOV C, L	MOV C, M	MOV C, A
12	MOV D, B	MOV D, C	MOV D, D	MOV D, E	MOV D, H	MOV D, L	MOV D, M	MOV D, A
13	MOV E, B	MOV E, C	MOV E, D	MOV E, E	MOV E, H	MOV E, L	MOV E, M	MOV E, A
14	MOV H, B	MOV H, C	MOV H, D	MOV H, E	MOV H, H	MOV H, L	MOV H, M	MOV H, A
15	MOV L, B	MOV L, C	MOV L, D	MOV L, E	MOV L, H	MOV L, L	MOV L, M	MOV L, A
16	MOV M, B	MOV M, C	MOV M, D	MOV M, E	MOV M, H	MOV M, L	HLT	MOV M, A
17	MOV A, B	MOV A, C	MOV A, D	MOV A, E	MOV A, H	MOV A, L	MOV A, M	MOV A, A
20	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A
21	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
22	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A
23	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
24	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A
25	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
26	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A
27	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
30	RNZ	POP B	JNZ a16	JMP a16	CNZ a16	PUSH B	ADI d8	-
31	RZ	RET	JZ a16	-	CZ a16	CALL a16	ACI d8	-
32	RNC	POP D	JNC a16	OUT	CNC a16	PUSH D	SUI d8	-
33	RC	-	JC a16	-	CC a16	-	SBI d8	-
34	RPO	POP H	JPO a16	-	CPO a16	PUSH H	ANI d8	-
35	RPE	-	JPE a16	-	CPE a16	-	XRI d8	-
36	RP	POP PSW	JP a16	-	CP a16	PUSH PSW	ORI d8	-
37	RM	-	JM a16	-	CM a16	-	CPI d8	-

The reference could be found here:

https://pastraiser.com/cpu/i8085/i8085_opcodes.htm

Results

We tested each block individually to make sure that there is no error with the hardware. All blocks then tested to work properly with each other via writing simple testing assembly programs. The results obtained from the simulation and implemented on the FPGA.

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

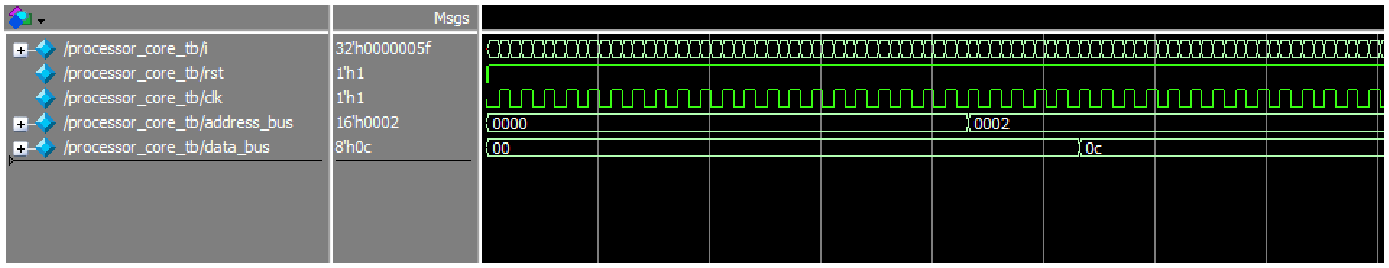
MVI A, 08

MVI B, 04

ADD B

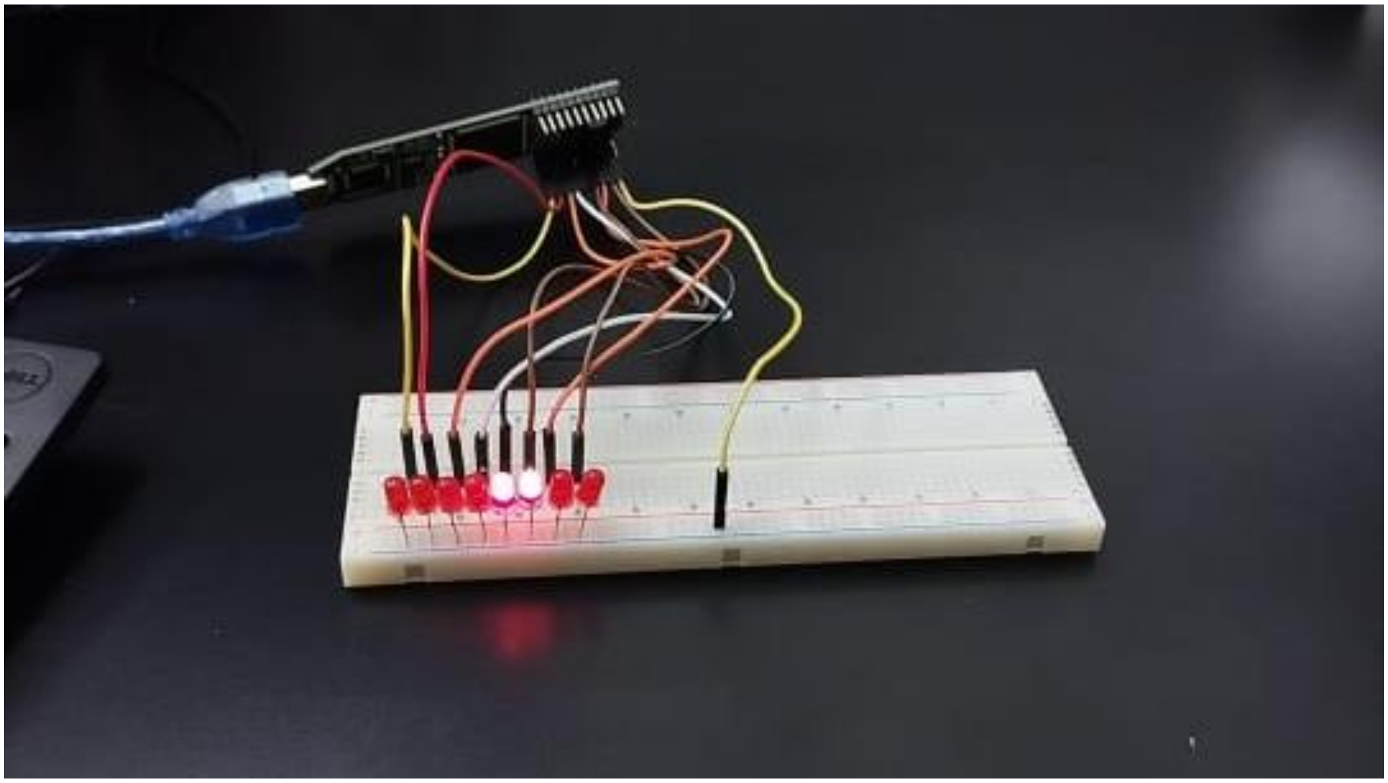
OUT 02

HLT



```
                              Msgs
/processor_core_tb/i          32'h0000005f
/processor_core_tb/rst        1'h1
/processor_core_tb/clk        1'h1
/processor_core_tb/address_bus 16'h0002     0000   0002
/processor_core_tb/data_bus   8'h0c         00     0c
```

Figure 2 addition program results



Stack Testing Code

INX B

INX B

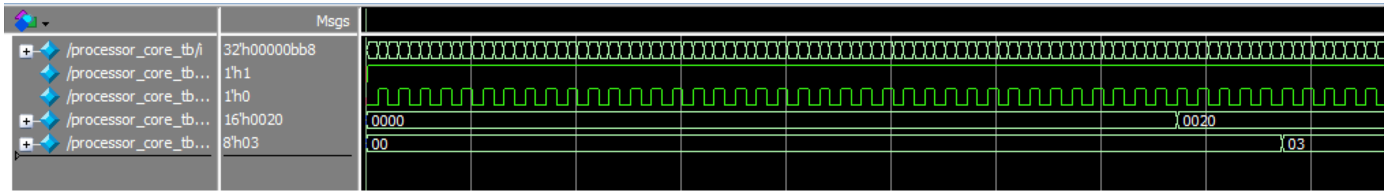
INX B

PUSH B

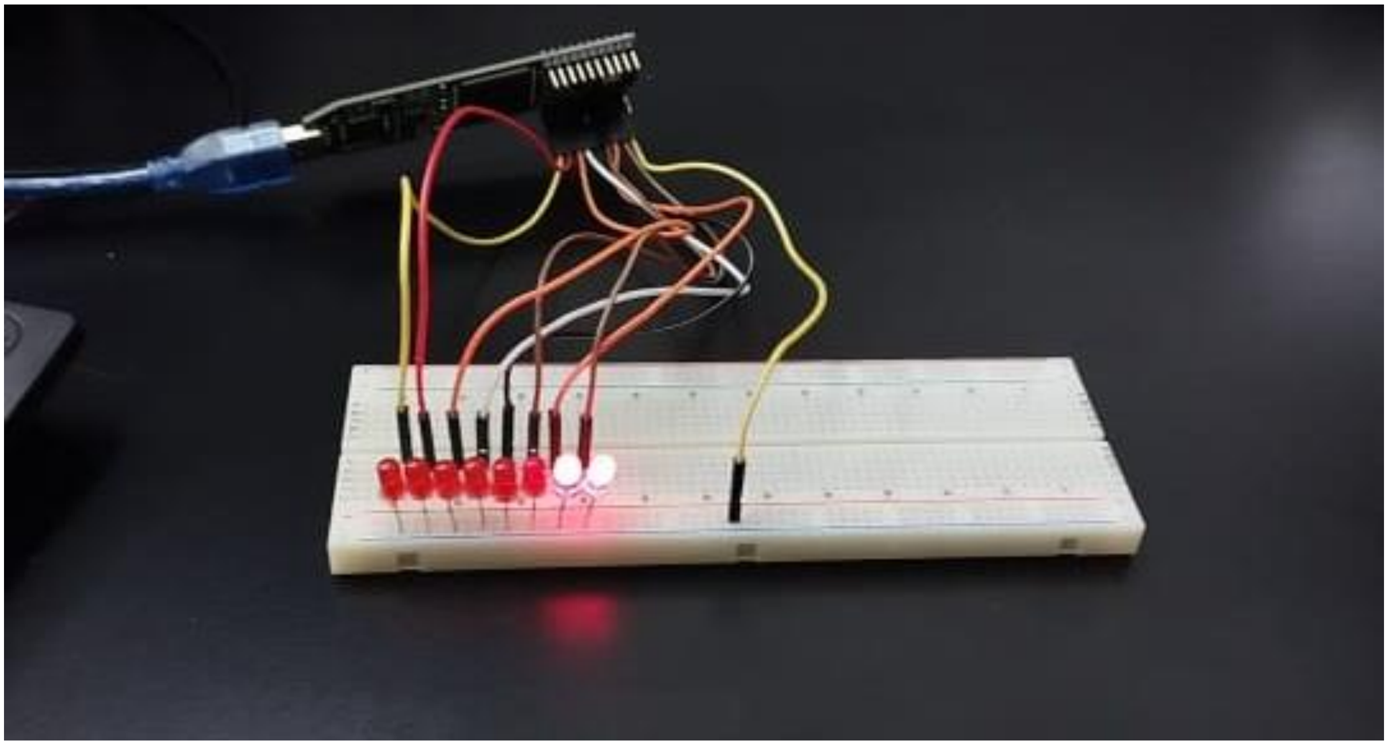
POP D

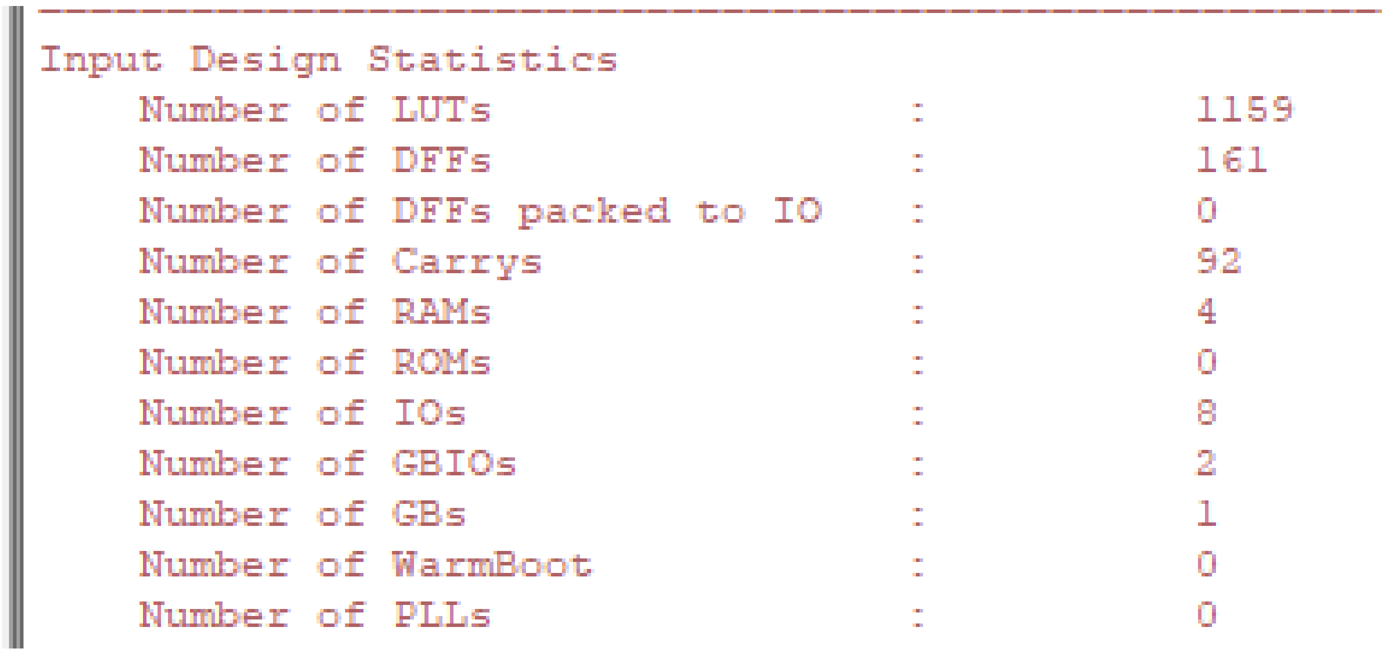
MOV A, E

OUT 20



```
                                Msgs
/processor_core_tb/i      32'h00000bb8
/processor_core_tb...     1'h1
/processor_core_tb...     1'h0
/processor_core_tb...     16'h0020      0000                  0020
/processor_core_tb...     8'h03         00                        03
```





```
--------------------------------------------------------------------------
Input Design Statistics
    Number of LUTs                         :          1159
    Number of DFFs                         :          161
    Number of DFFs packed to IO            :          0
    Number of Carrys                       :          92
    Number of RAMs                         :          4
    Number of ROMs                         :          0
    Number of IOs                          :          8
    Number of GBIOs                        :          2
    Number of GBs                          :          1
    Number of WarmBoot                     :          0
    Number of PLLs                         :          0
```

Further development

Further development could be made as follows:

- Implement dual core processor to investigate this kind of architecture
- Implement approximate instructions targeting high speed and low power