The American University in Cairo

CENTER OF NANOELECTRONICS & DEVICES

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

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.

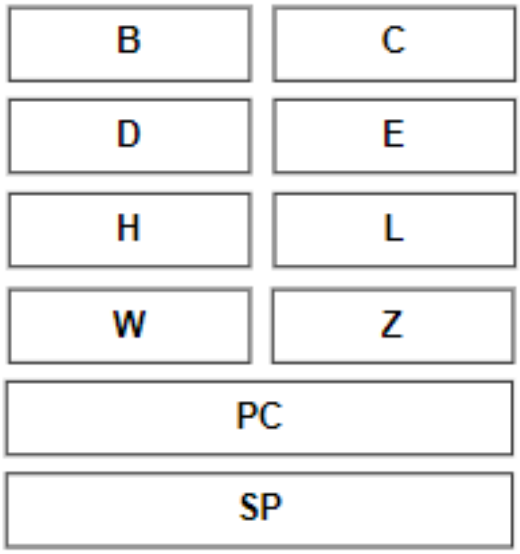


Figure 1 Programming model of the designed processor

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.

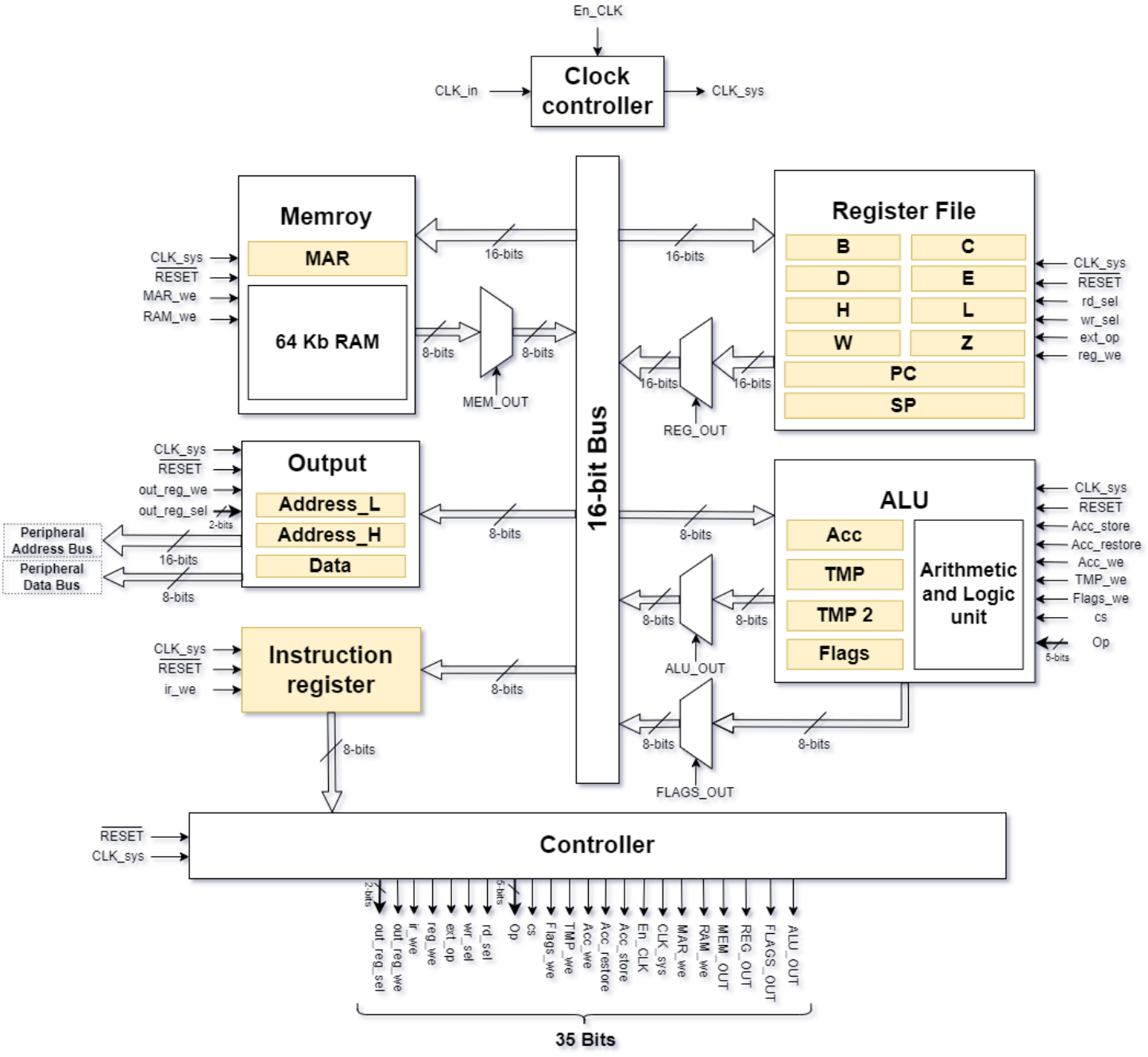


Figure 2 Processor Core Overall Block Diagram

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

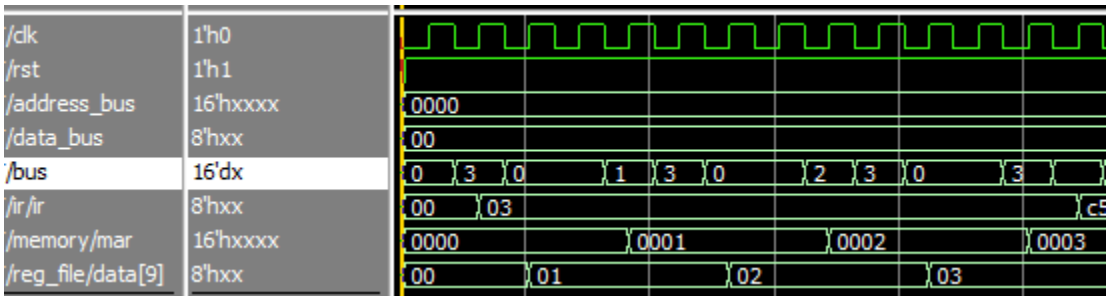


Figure 3 Fetching cycle

Area Analysis

```
------------------------------------------------------------
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
```

Figure 4 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

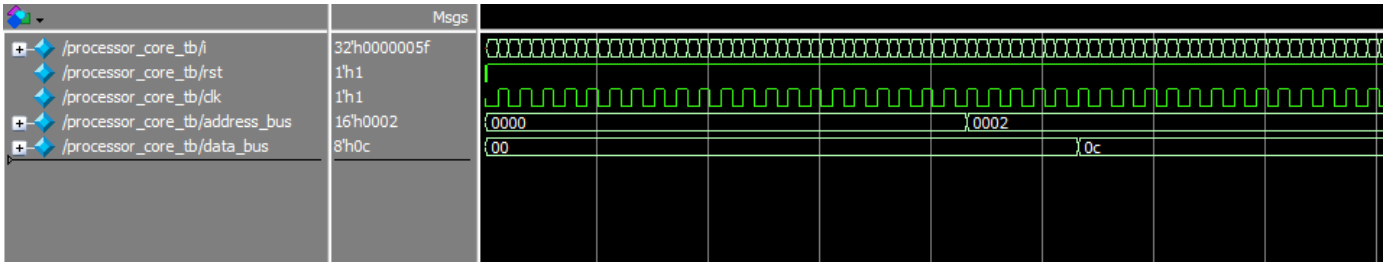


Figure 5 addition program results

Stack Testing Code

INX B

INX B

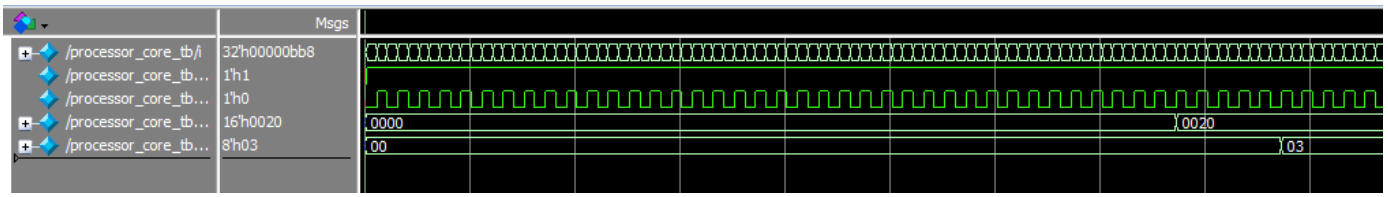
INX B

PUSH B

POP D

MOV A, E

OUT 20



The American University in Cairo

CENTER OF NANOELECTRONICS & DEVICES

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.

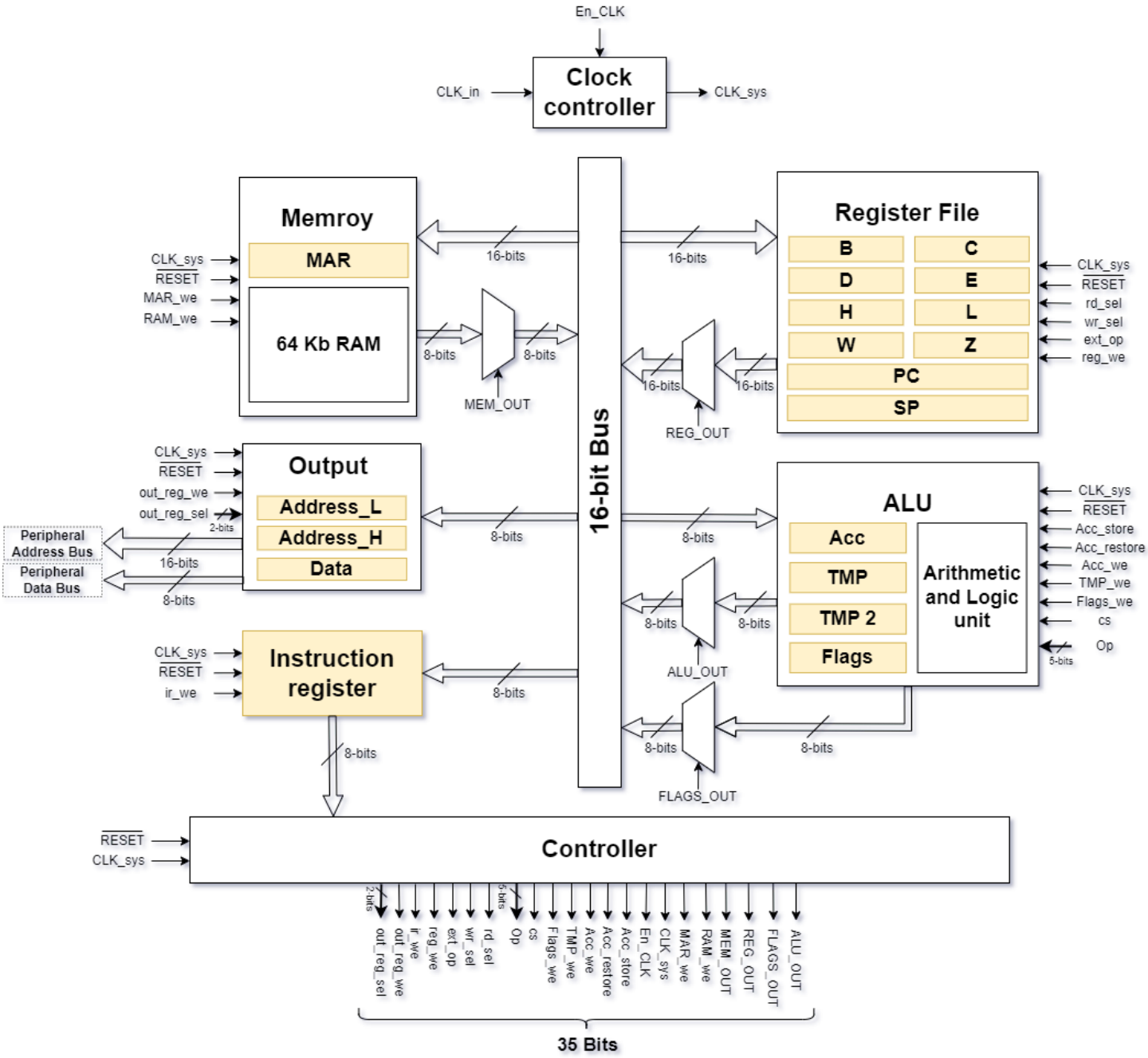


Figure 1 Processor Core Overall Block Diagram

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

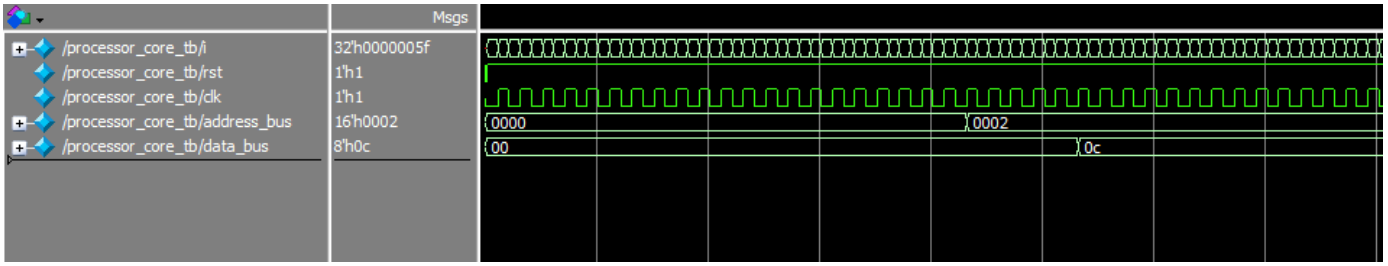
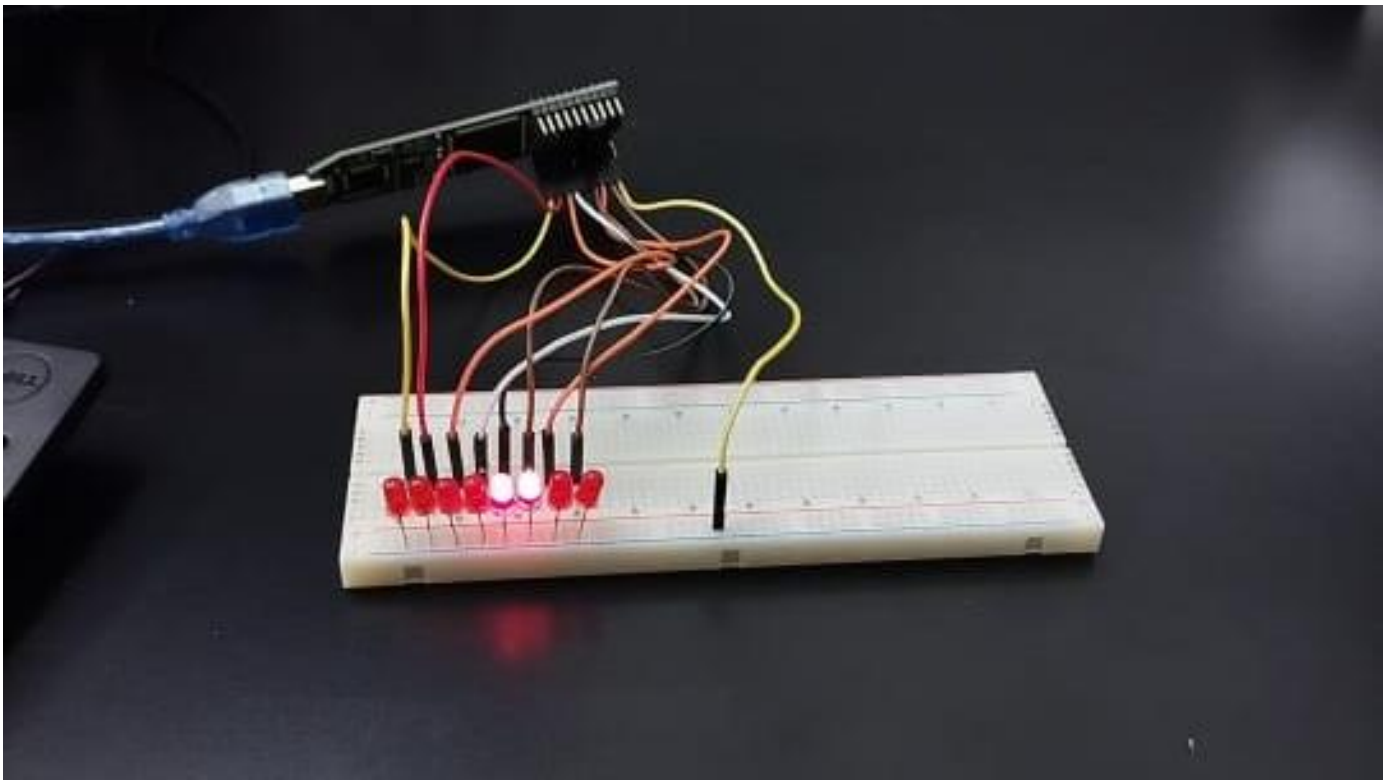


Figure 2 addition program results



Stack Testing Code

INX B

INX B

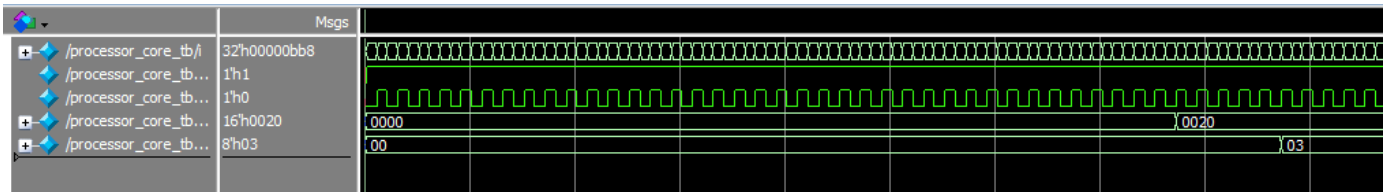
INX B

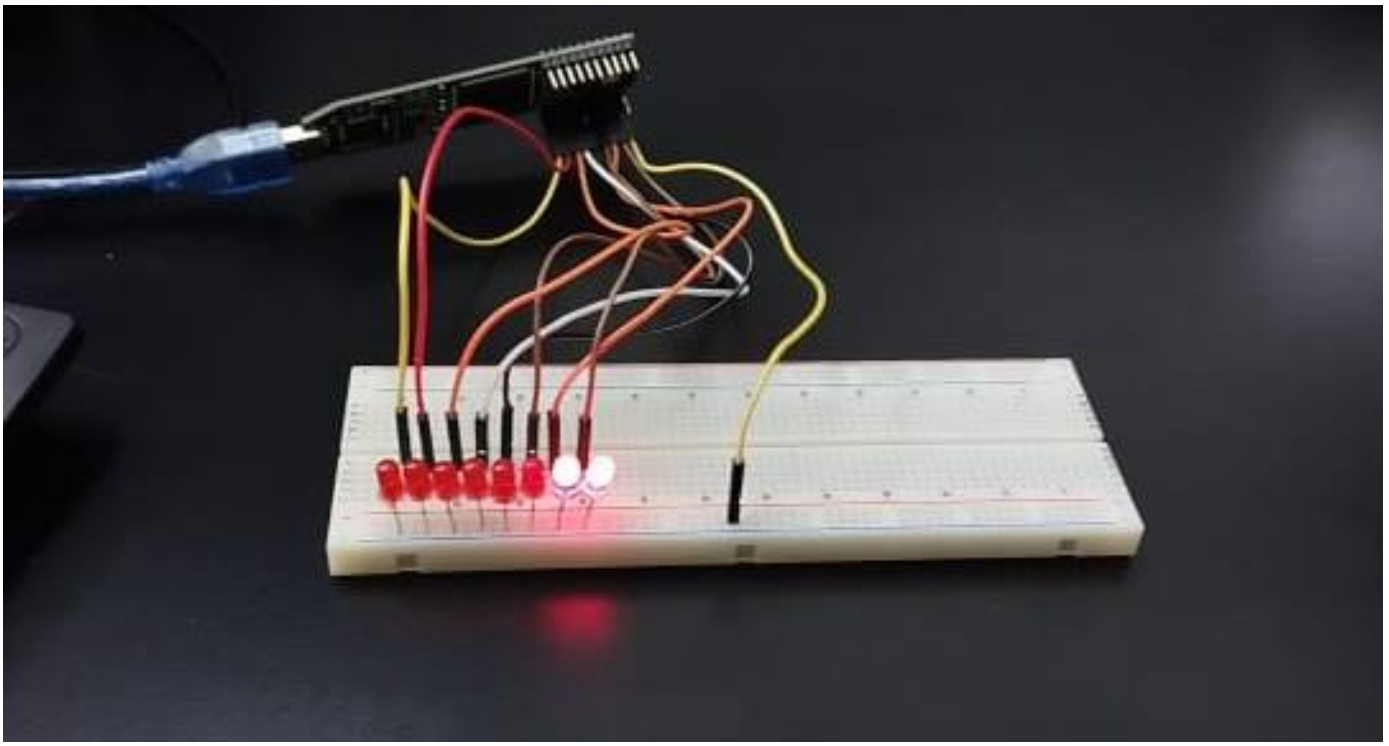
PUSH B

POP D

MOV A, E

OUT 20





```
------------------------------------------------------------
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