4-BIT COMPUTER

EEE 315 Assignment

Student Name : Mir Sayeed Mohammad
Student ID : 1606003

Submitted to : Dr. Sajid Muhaimin Choudhury

Objectives

Covered in the assignment:

- Implementation of a 4-bit computer using **Verilog** for given instruction set

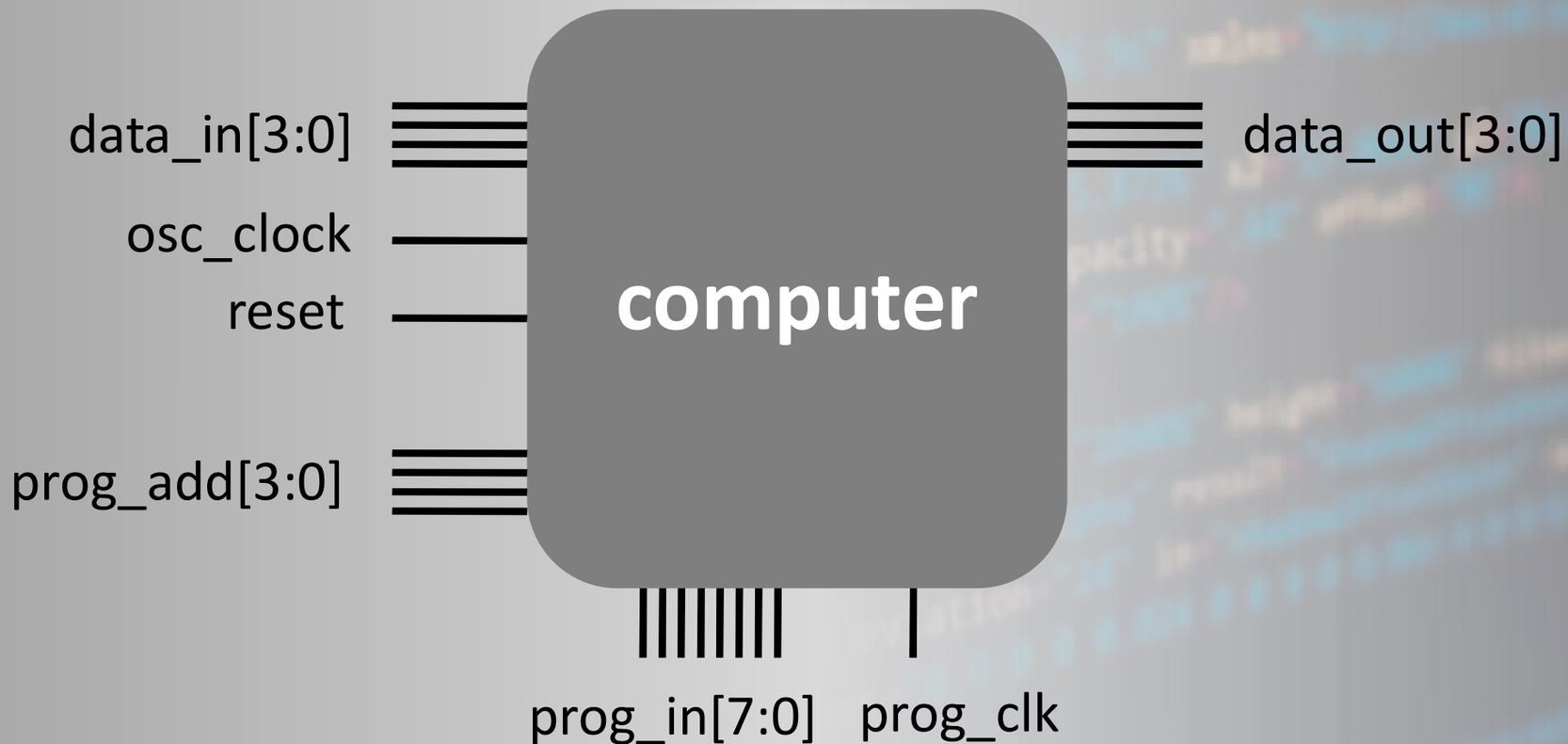
- Programming and testing the computer on **EDA-Playground** with test-bench code and waveform output

- Designing a **python based assembler** for the computer unit

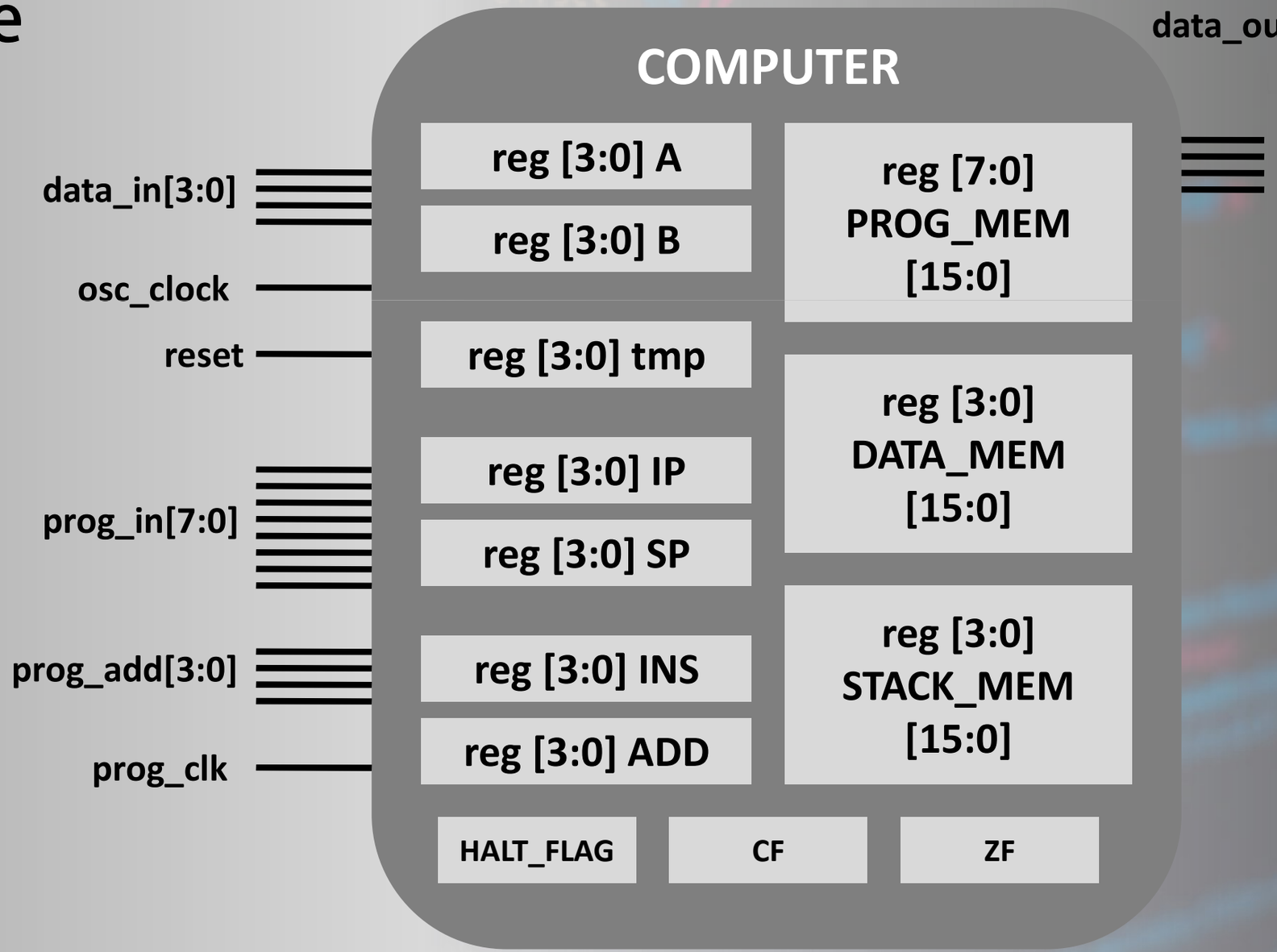
Instruction Set	
0	`ADD A, B`
1	`SUB A, B`
2	`XCHG B, A`
3	`IN A`
4	`OUT A`
5	`INC A`
6	`MOV A, [ADDR]`
7	`MOV A, BYTE`
8	`JZ ADDRESS`
9	`PUSH B`
10	`POP B`
11	`RCL B`
12	`CALL ADDRESS`
13	`RET`
14	`AND A, [ADDR]`
15	`HLT`

Verilog Module

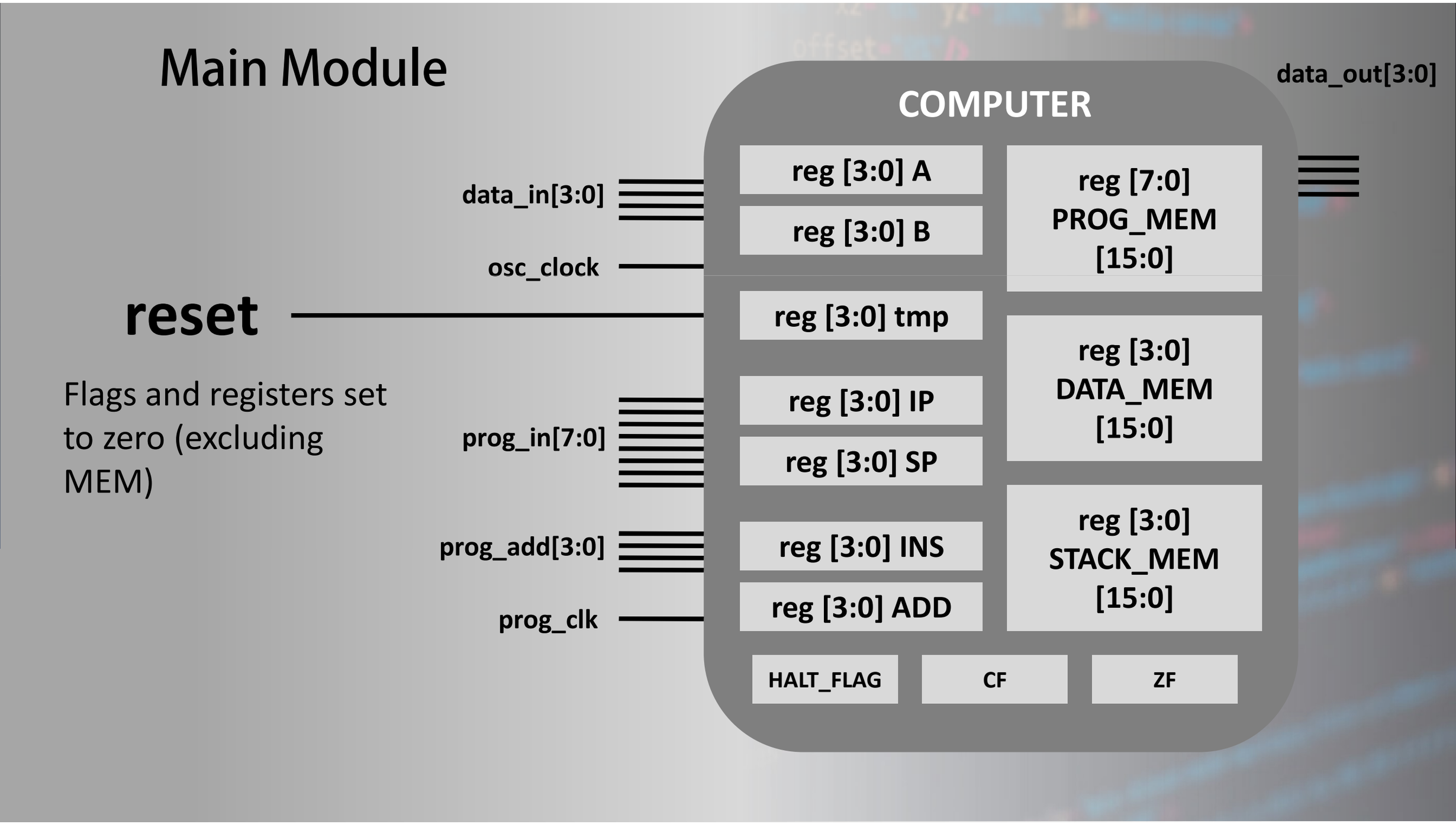
```
module computer (data_in, data_out, osc_clock, reset, prog_in, prog_clk, prog_add);
```



Main Module


COMPUTER
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
data_out[3:0]
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] tmp
reg [3:0] IP
reg [3:0] SP
reg [3:0] DATA_MEM [15:0]
reg [3:0] INS
reg [3:0] ADD
reg [3:0] STACK_MEM [15:0]
HALT_FLAG
CF
ZF

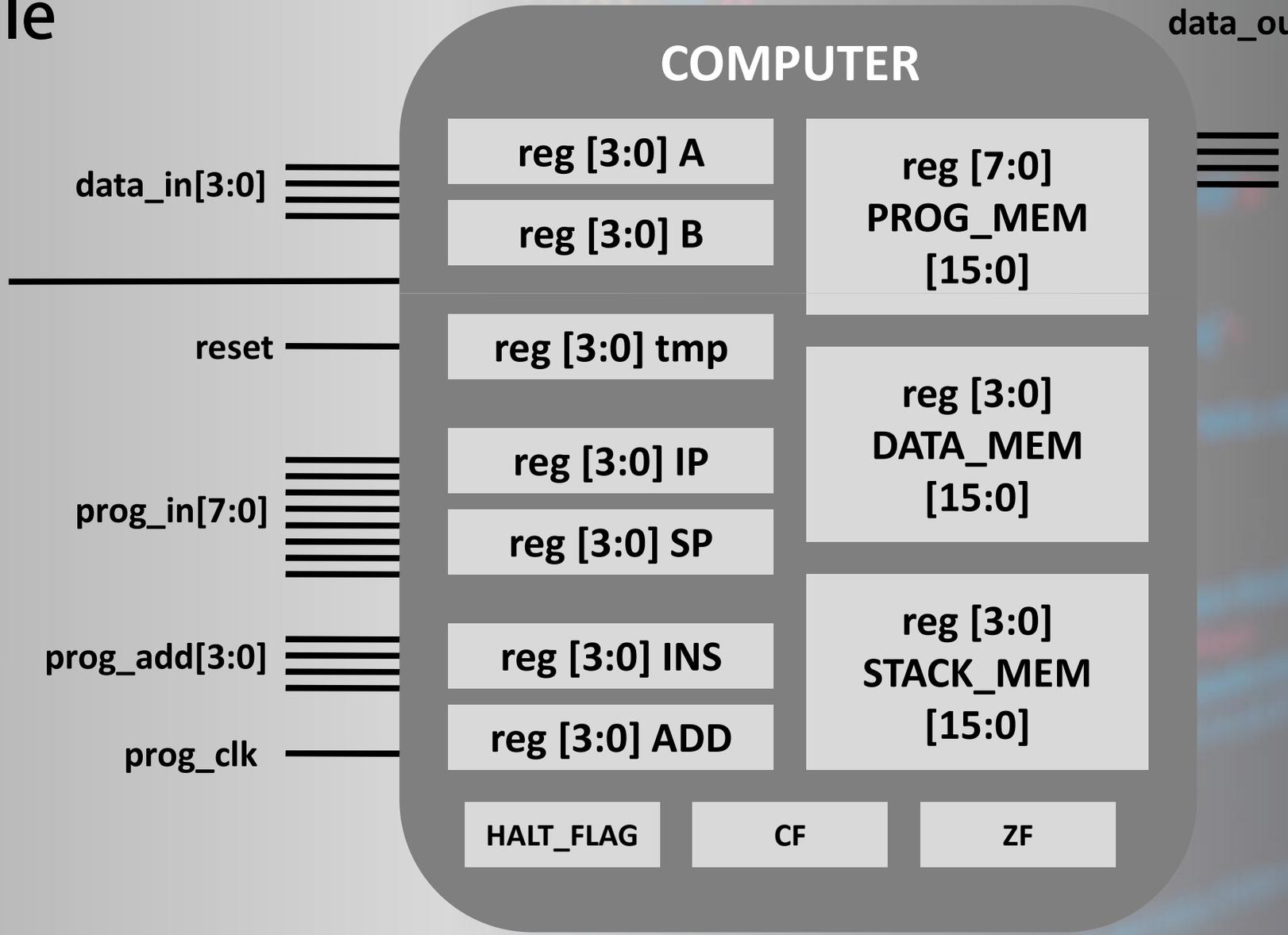
Main Module


COMPUTER
data_out[3:0]
data_in[3:0]
osc_clock
reset
Flags and registers set to zero (excluding MEM)
prog_in[7:0]
prog_add[3:0]
prog_clk
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] tmp
reg [3:0] DATA_MEM [15:0]
reg [3:0] IP
reg [3:0] SP
reg [3:0] INS
reg [3:0] STACK_MEM [15:0]
reg [3:0] ADD
HALT_FLAG
CF
ZF

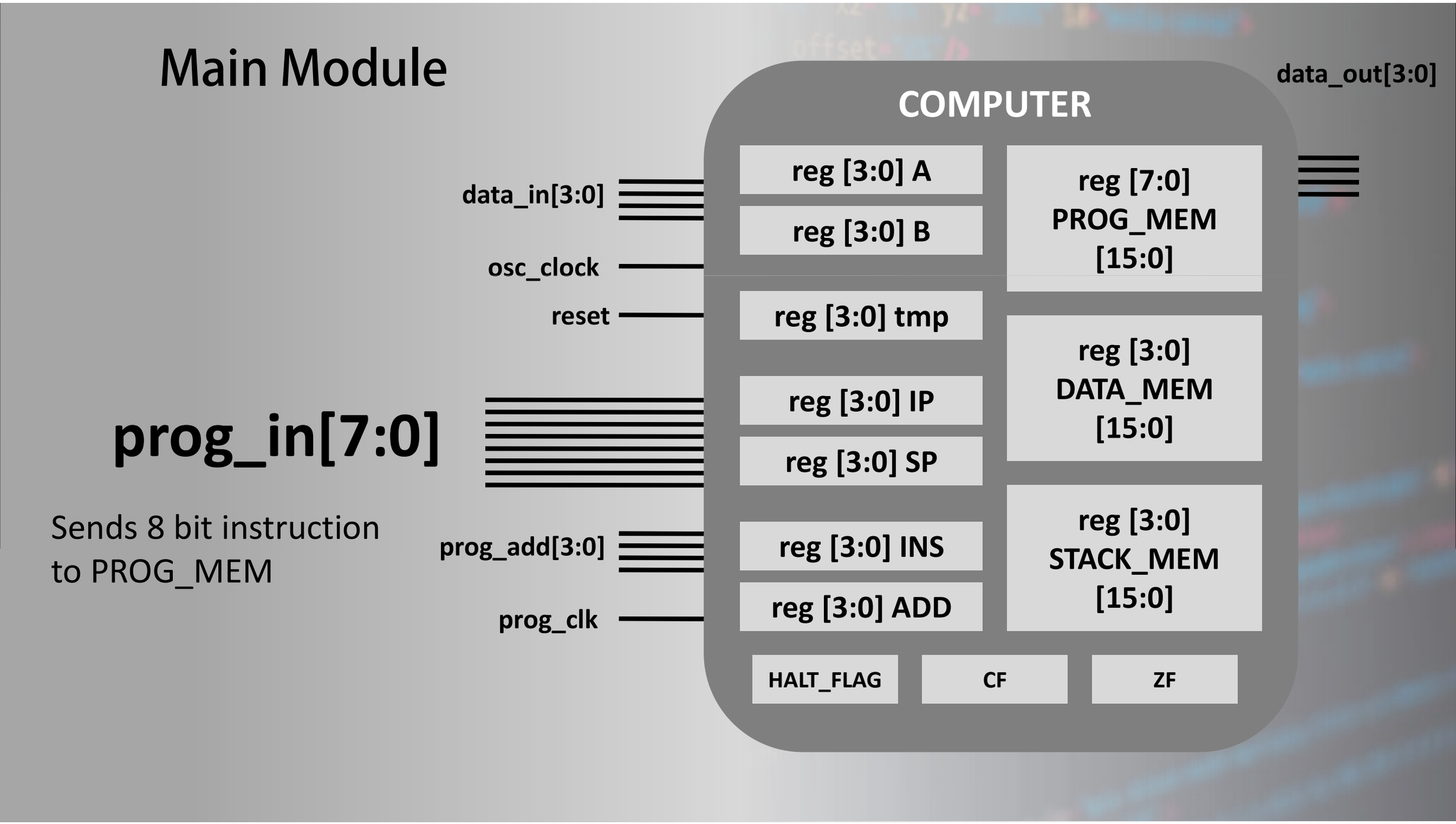
Main Module

osc_clock

Provides clock for all internal operations



Main Module

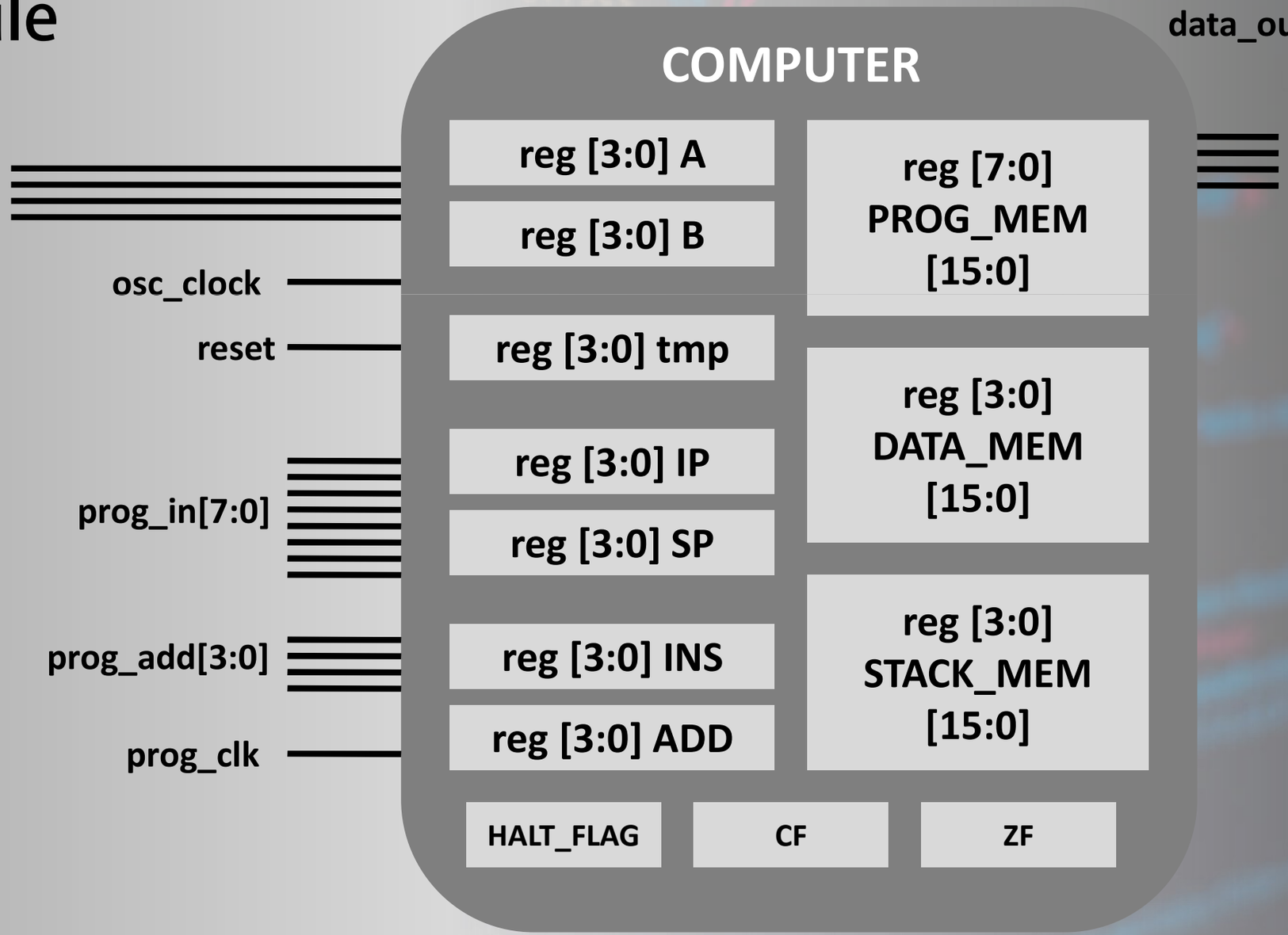


Sends 8 bit instruction to PROG_MEM

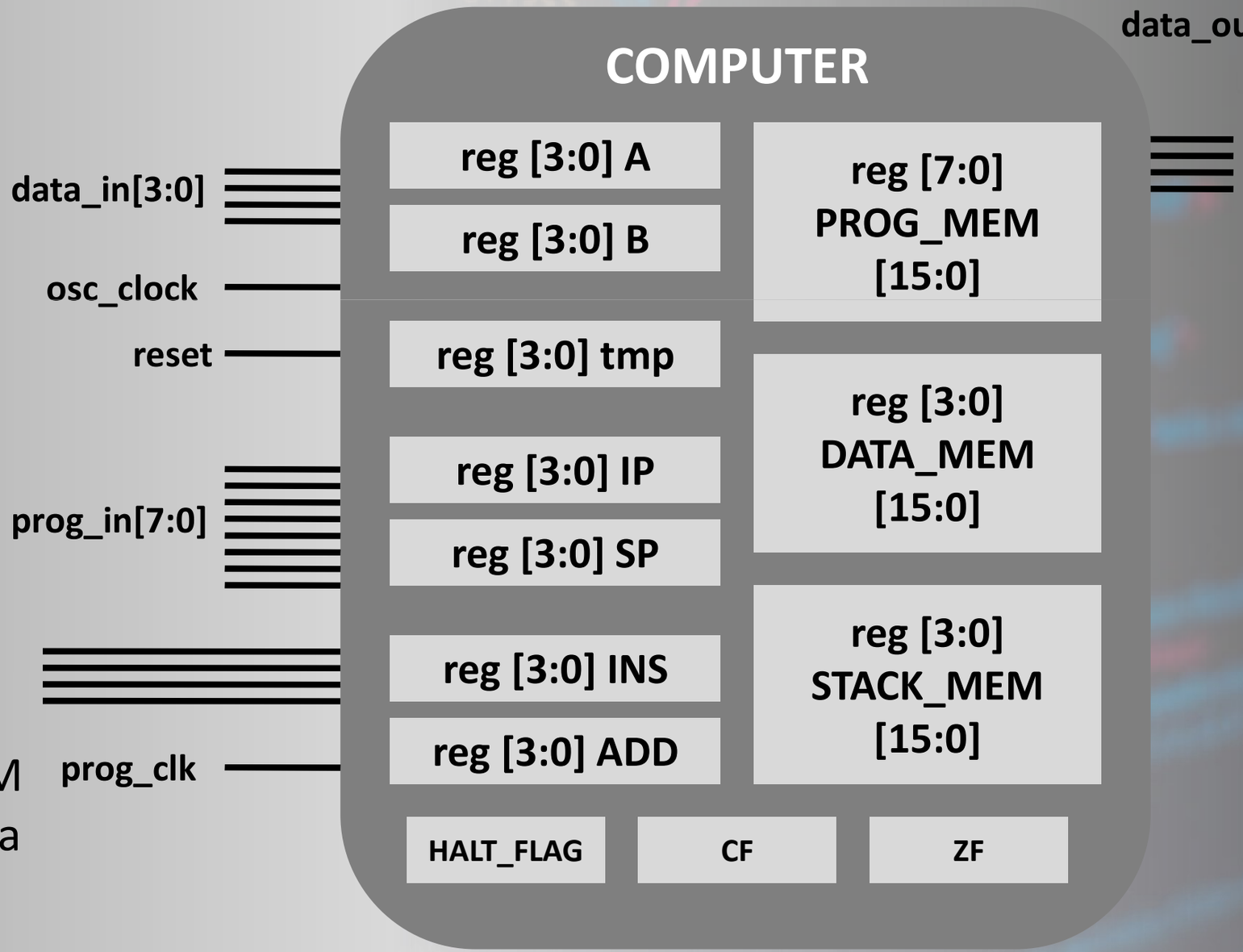
Main Module

data_in[3:0]

1. Acts as computer data input to reg A
2. Loads data in DATA_MEM when programming



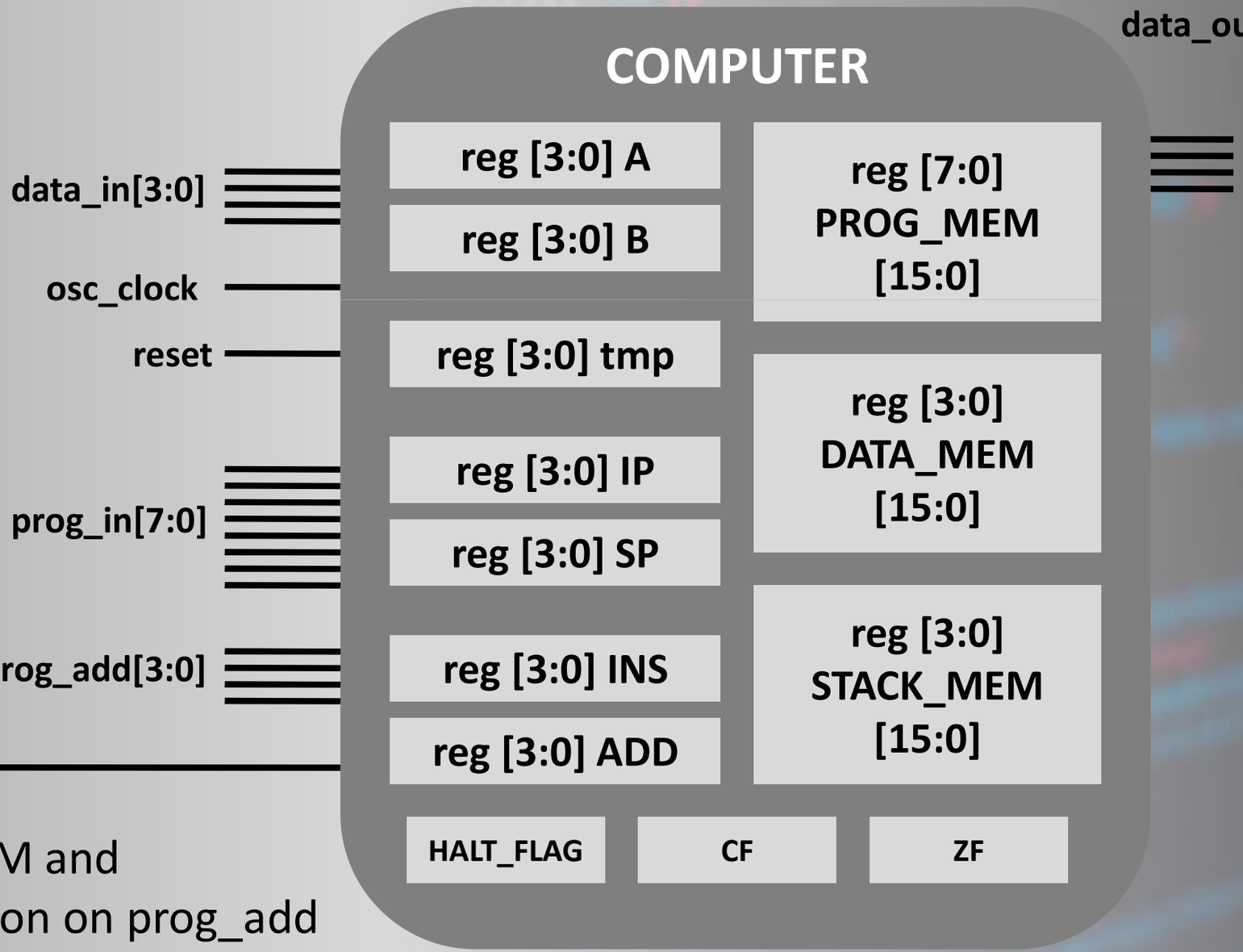
Main Module



prog_add[3:0]

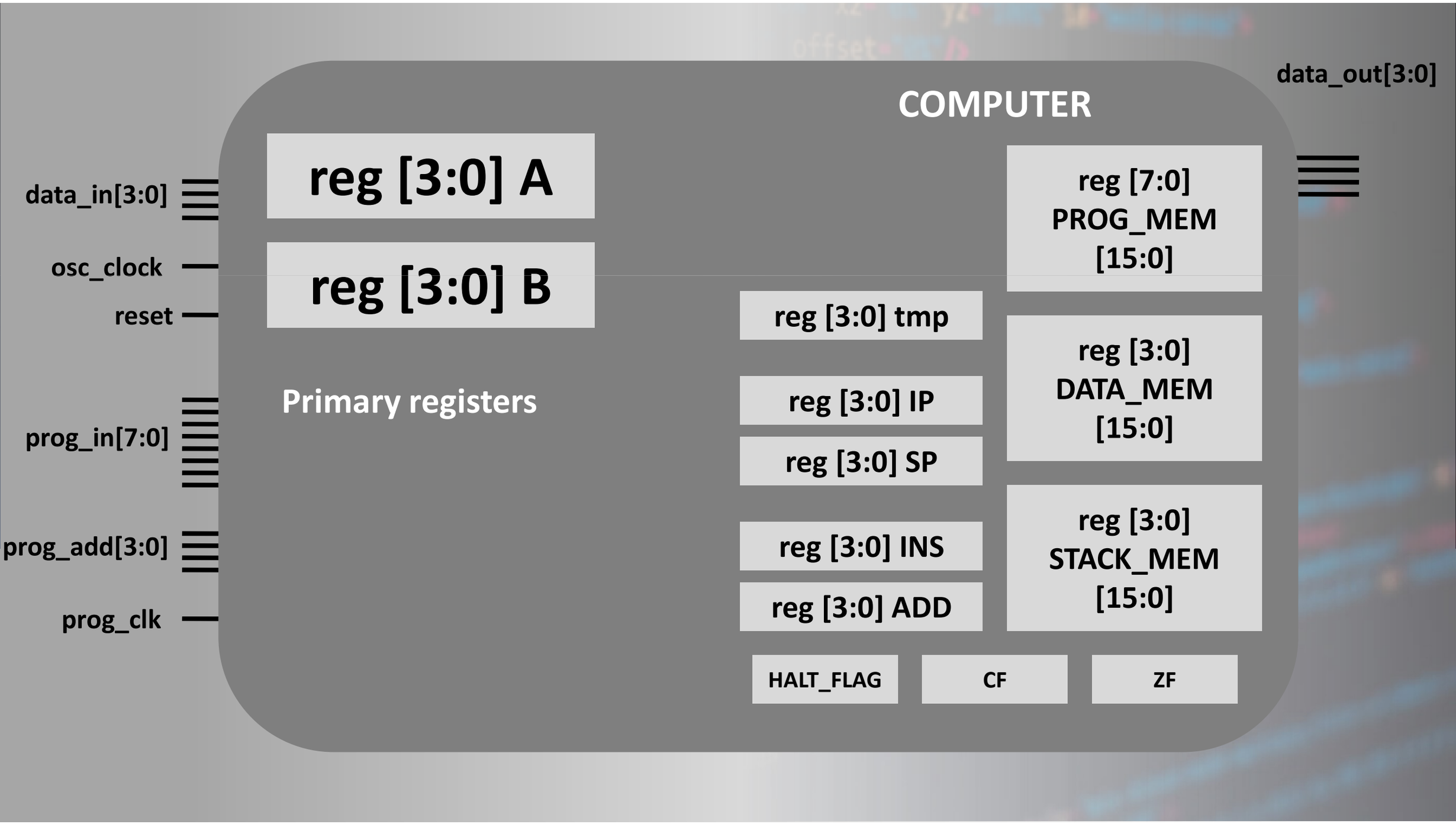
Selects address of PROG_MEM and DATA_MEM on which data will be loaded

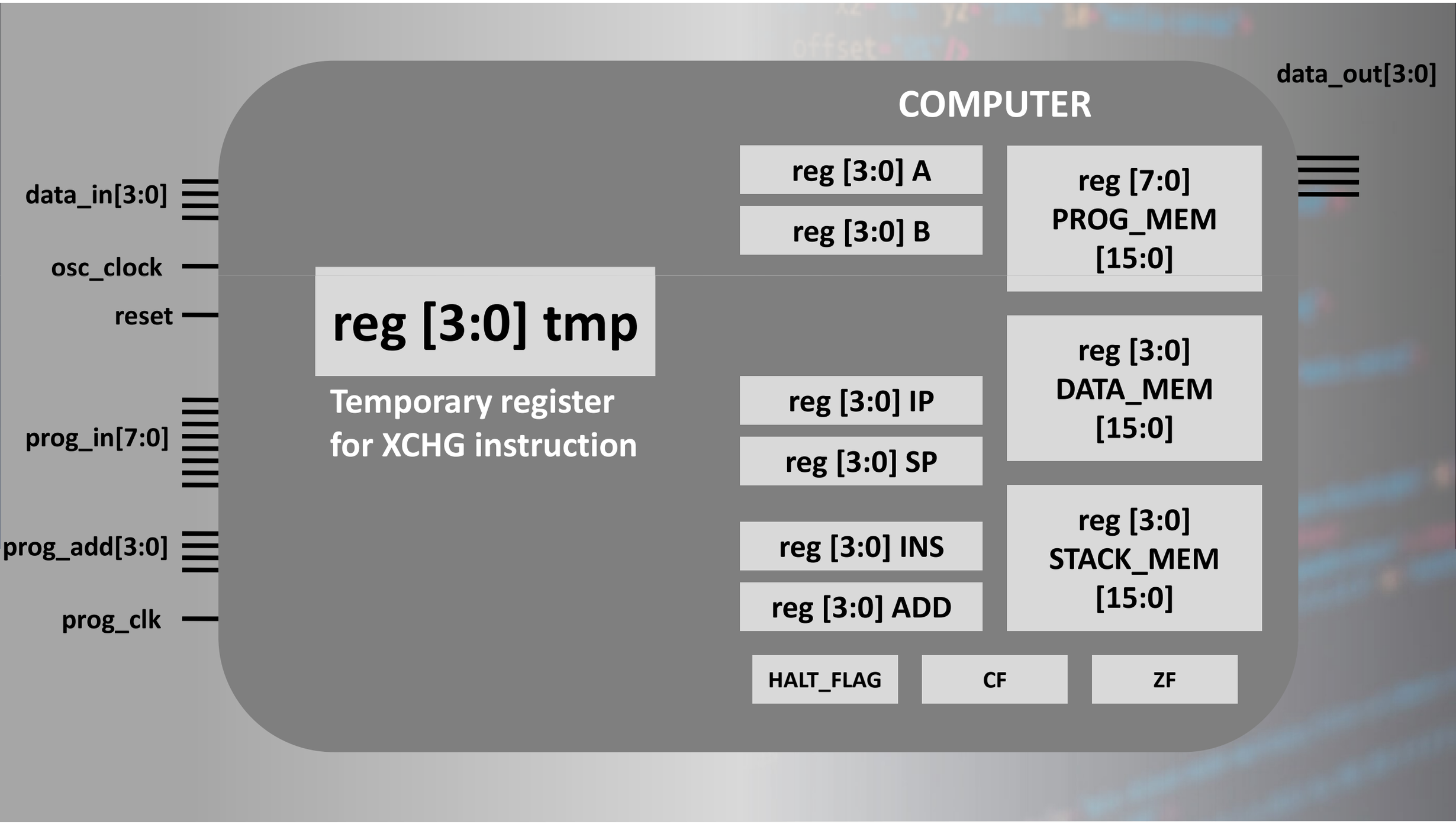
Main Module

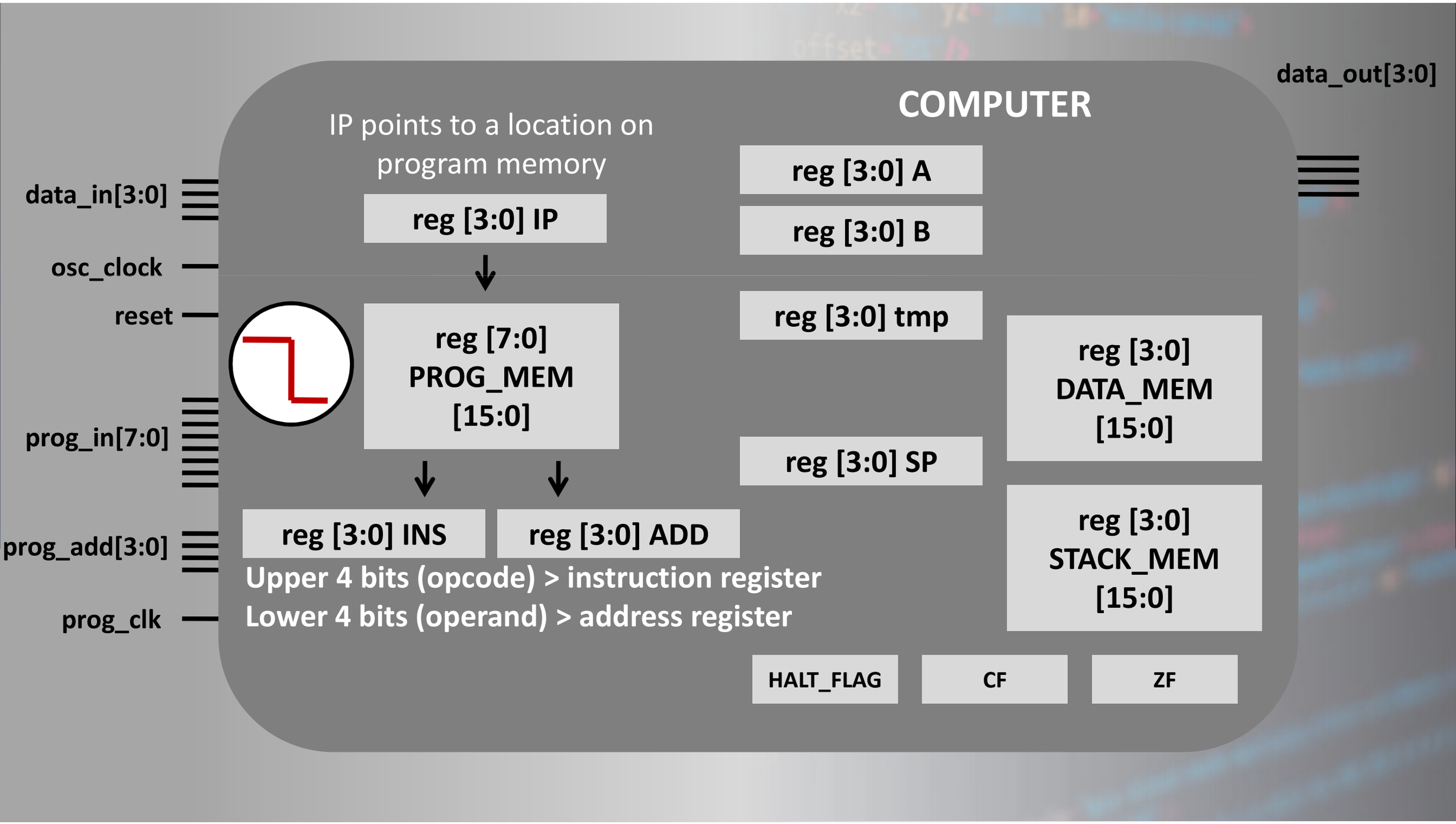


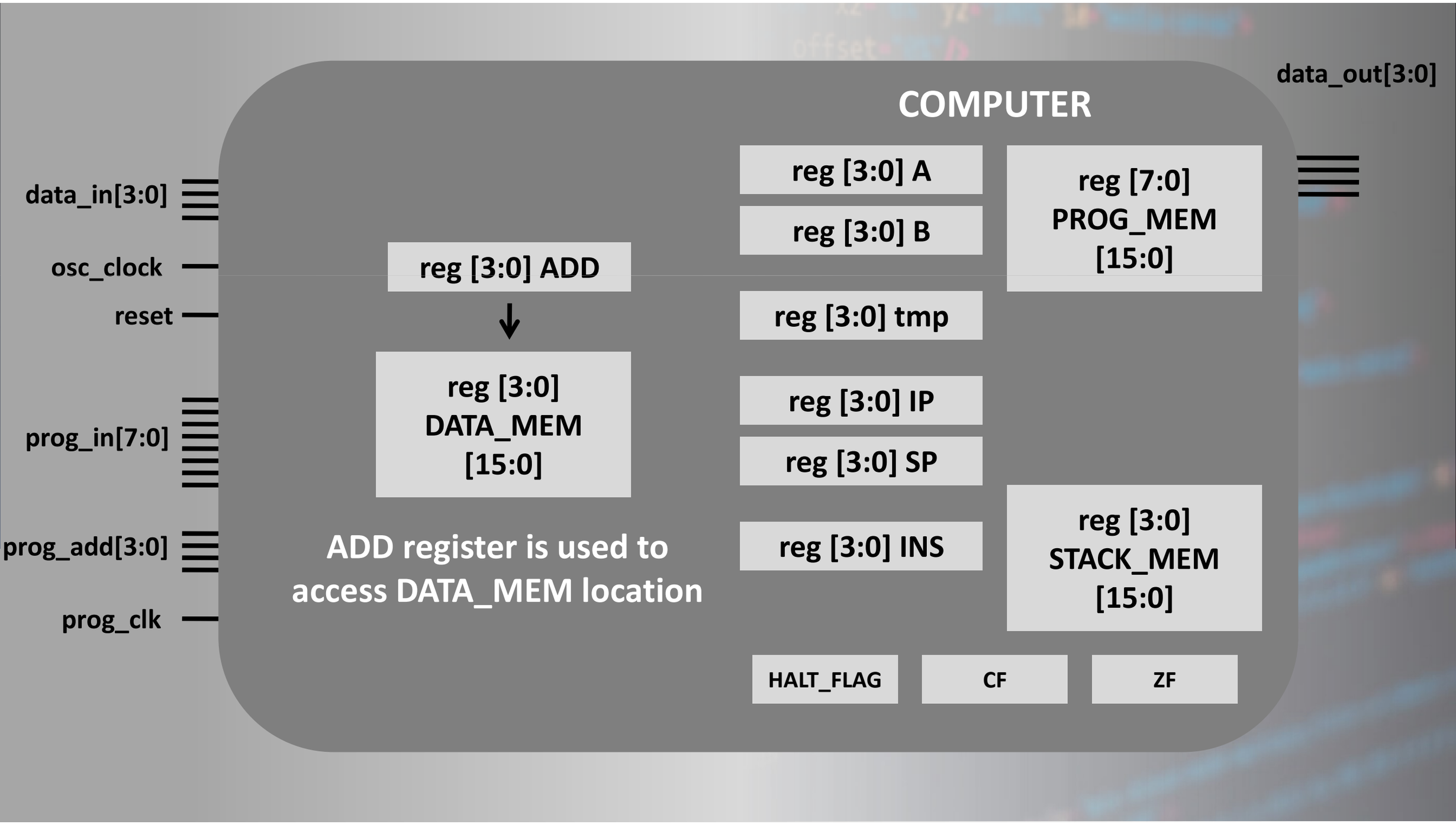
prog_clk

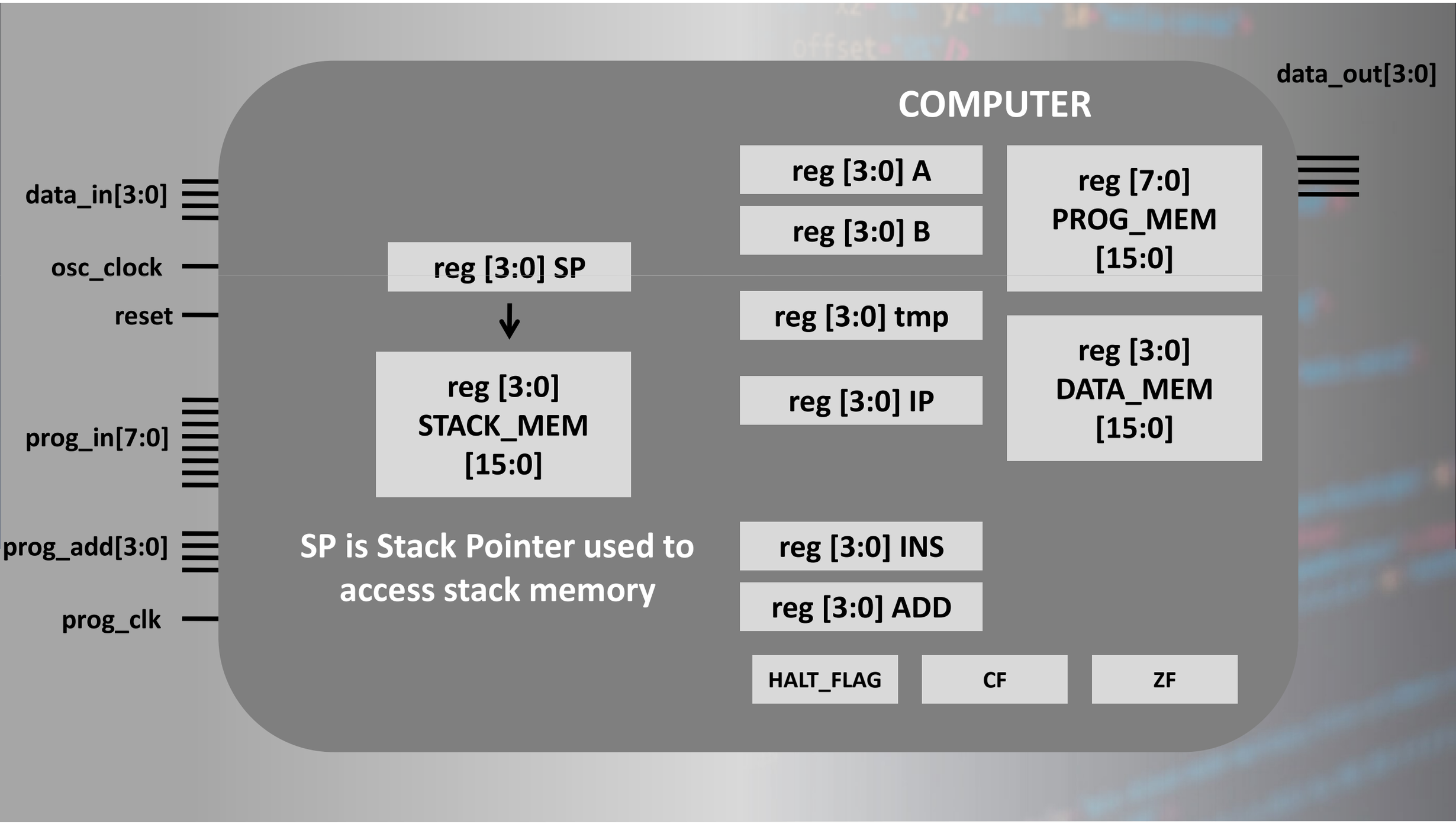
On positive edge, DATA_MEM and PROG_MEM loads information on prog_add address from data_in and prog_in

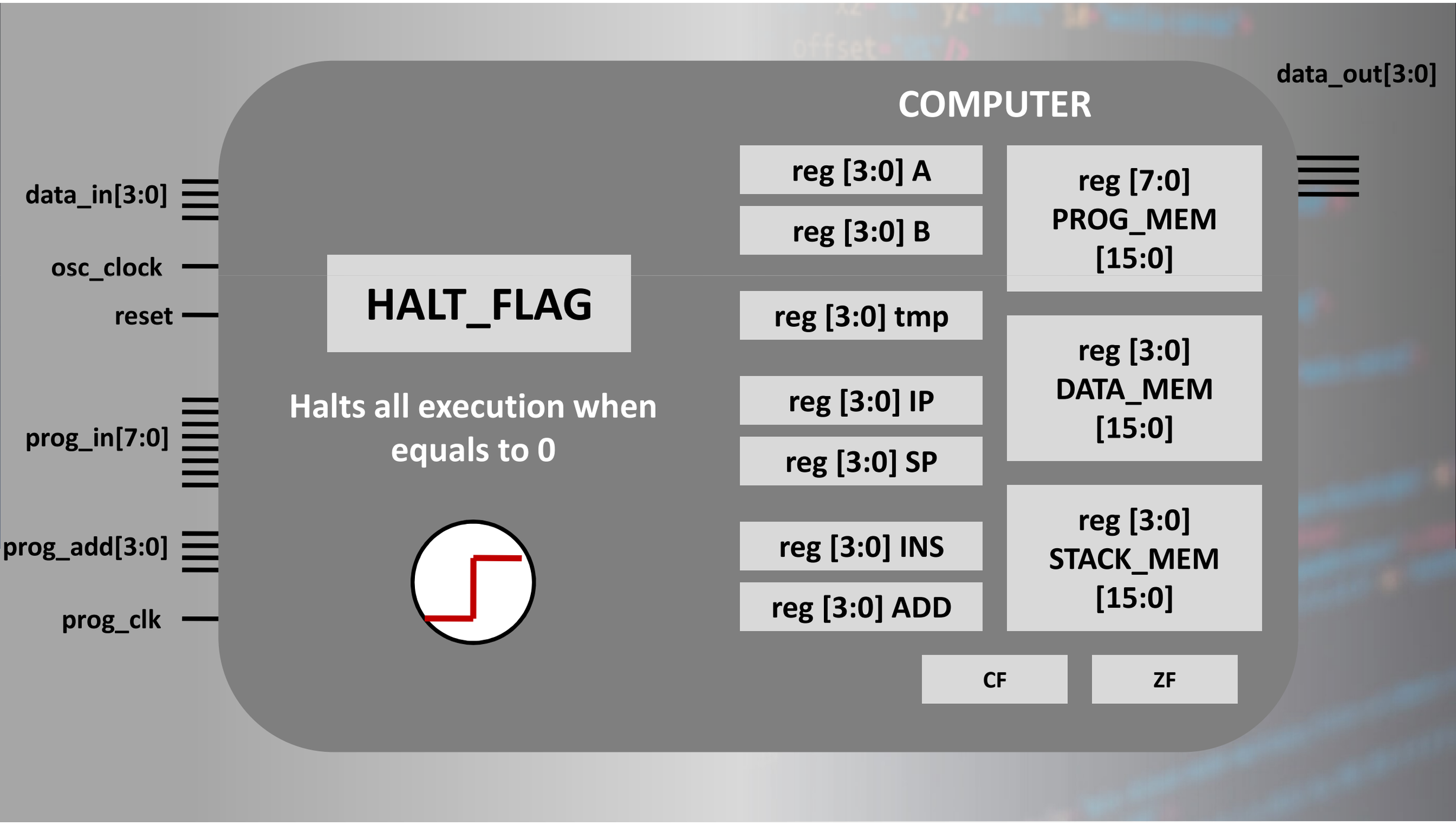

COMPUTER
data_out[3:0]
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
reg [3:0] A
reg [3:0] B
Primary registers
reg [7:0] PROG_MEM [15:0]
reg [3:0] tmp
reg [3:0] DATA_MEM [15:0]
reg [3:0] IP
reg [3:0] SP
reg [3:0] INS
reg [3:0] STACK_MEM [15:0]
reg [3:0] ADD
HALT_FLAG
CF
ZF

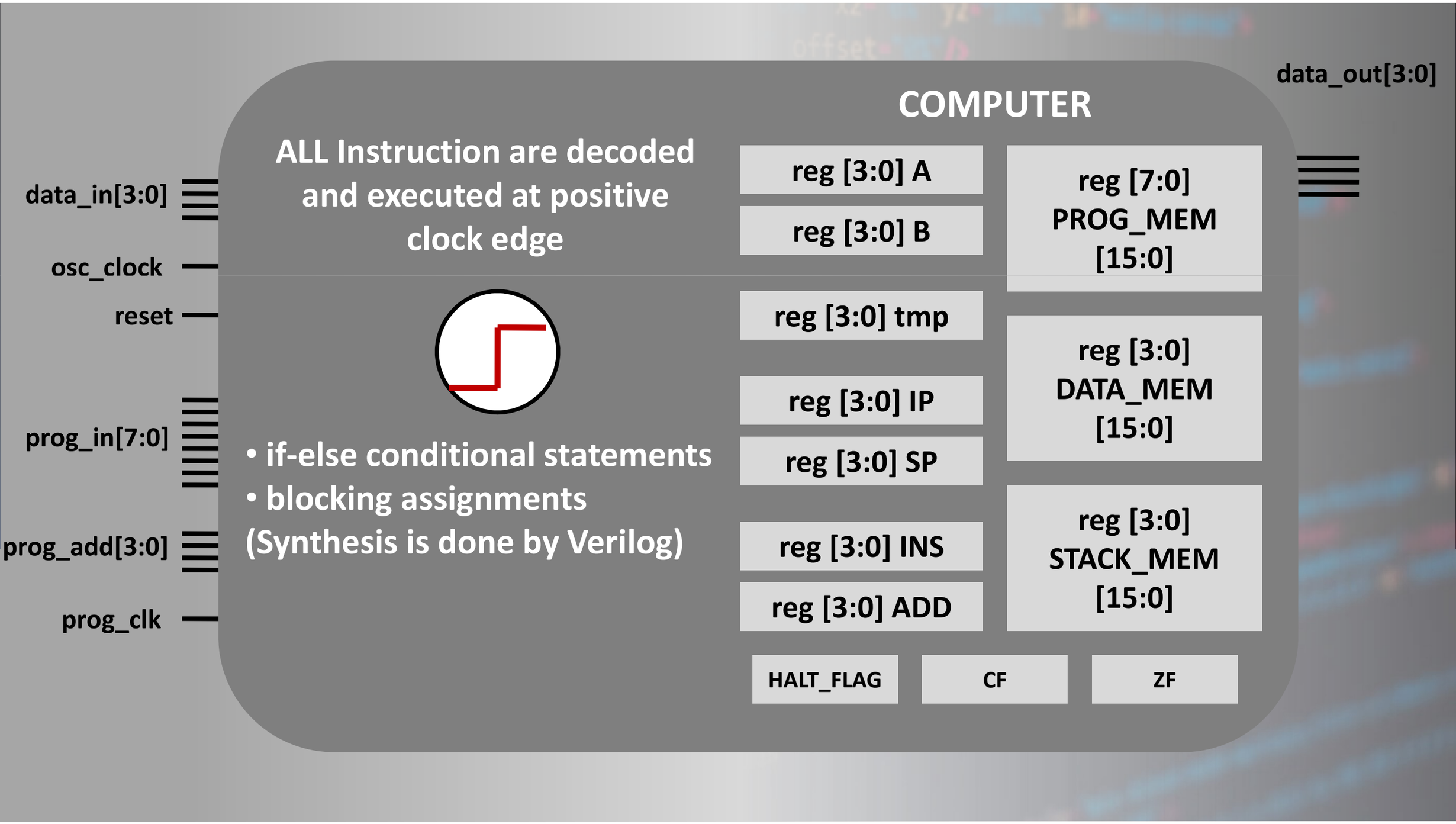

data_out[3:0]
COMPUTER
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] tmp
Temporary register for XCHG instruction
reg [3:0] IP
reg [3:0] SP
reg [3:0] DATA_MEM [15:0]
reg [3:0] INS
reg [3:0] ADD
reg [3:0] STACK_MEM [15:0]
HALT_FLAG
CF
ZF

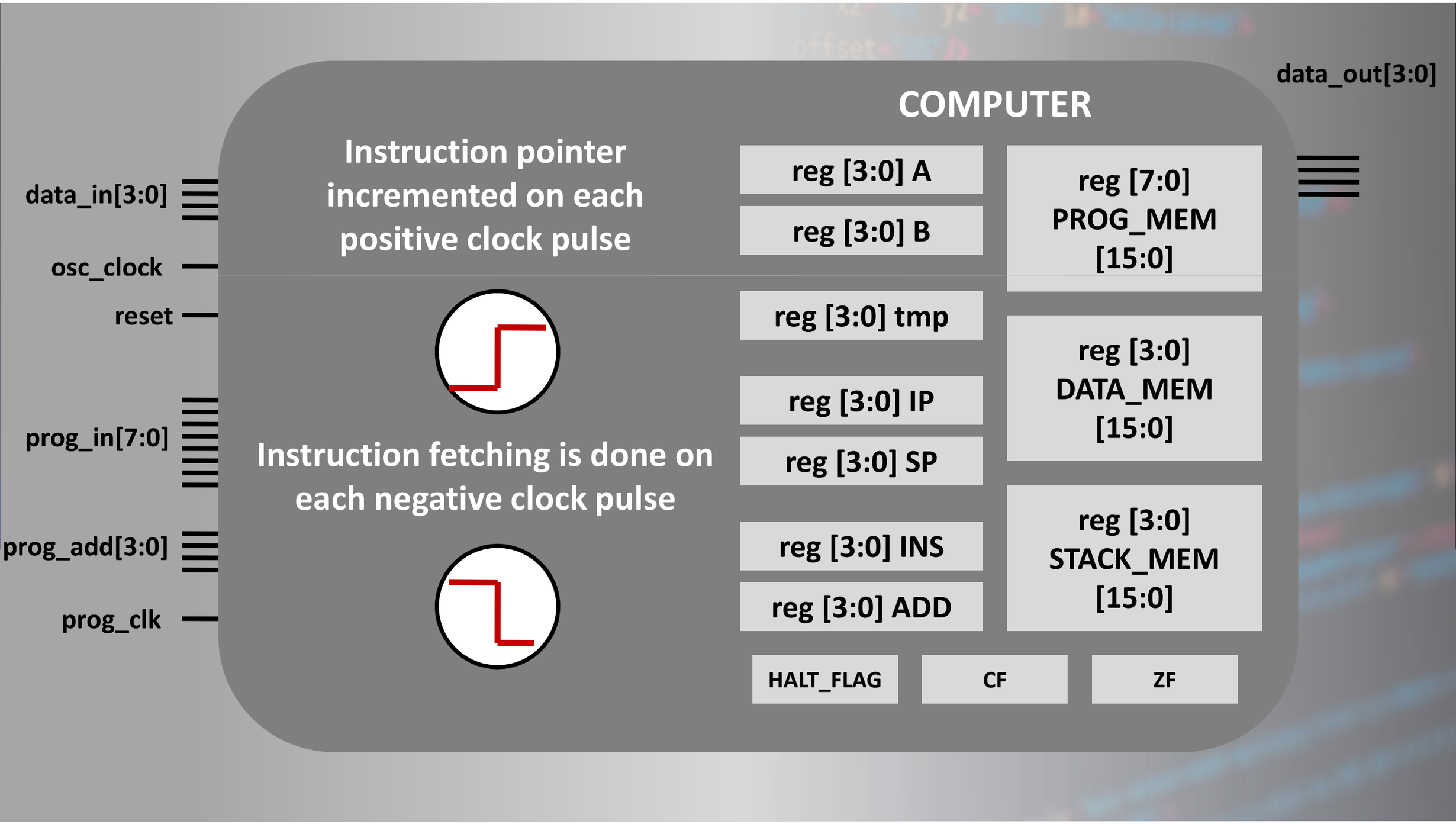

COMPUTER
data_out[3:0]
IP points to a location on program memory
reg [3:0] IP
reg [3:0] A
reg [3:0] B
reg [3:0] tmp
reg [7:0] PROG_MEM [15:0]
reg [3:0] DATA_MEM [15:0]
reg [3:0] SP
reg [3:0] INS
reg [3:0] ADD
reg [3:0] STACK_MEM [15:0]
Upper 4 bits (opcode) > instruction register
Lower 4 bits (operand) > address register
HALT_FLAG
CF
ZF
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk


data_out[3:0]
COMPUTER
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] ADD
reg [3:0] tmp
reg [3:0] DATA_MEM [15:0]
reg [3:0] IP
reg [3:0] SP
reg [3:0] INS
reg [3:0] STACK_MEM [15:0]
ADD register is used to access DATA_MEM location
HALT_FLAG
CF
ZF


data_out[3:0]
COMPUTER
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] SP
reg [3:0] tmp
reg [3:0] DATA_MEM [15:0]
reg [3:0] IP
reg [3:0] STACK_MEM [15:0]
SP is Stack Pointer used to access stack memory
reg [3:0] INS
reg [3:0] ADD
HALT_FLAG
CF
ZF


COMPUTER
data_out[3:0]
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
HALT_FLAG
Halts all execution when equals to 0
reg [3:0] A
reg [3:0] B
reg [3:0] tmp
reg [3:0] IP
reg [3:0] SP
reg [3:0] INS
reg [3:0] ADD
reg [7:0] PROG_MEM [15:0]
reg [3:0] DATA_MEM [15:0]
reg [3:0] STACK_MEM [15:0]
CF
ZF


data_out[3:0]
COMPUTER
ALL Instruction are decoded and executed at positive clock edge
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
• if-else conditional statements
• blocking assignments
(Synthesis is done by Verilog)
reg [3:0] A
reg [3:0] B
reg [7:0] PROG_MEM [15:0]
reg [3:0] tmp
reg [3:0] IP
reg [3:0] SP
reg [3:0] DATA_MEM [15:0]
reg [3:0] INS
reg [3:0] ADD
reg [3:0] STACK_MEM [15:0]
HALT_FLAG
CF
ZF


COMPUTER
data_out[3:0]
data_in[3:0]
osc_clock
reset
prog_in[7:0]
prog_add[3:0]
prog_clk
Instruction pointer incremented on each positive clock pulse
Instruction fetching is done on each negative clock pulse
reg [3:0] A
reg [3:0] B
reg [3:0] tmp
reg [3:0] IP
reg [3:0] SP
reg [3:0] INS
reg [3:0] ADD
reg [7:0] PROG_MEM [15:0]
reg [3:0] DATA_MEM [15:0]
reg [3:0] STACK_MEM [15:0]
HALT_FLAG
CF
ZF

Instruction Execution

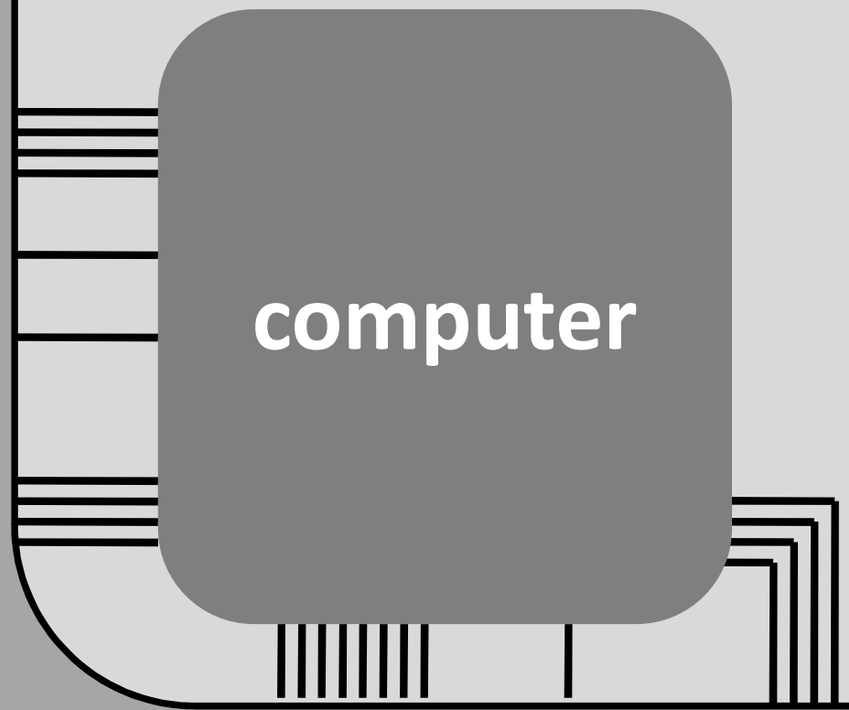
ADD A, B	SUB A, B	XCHG B, A	IN A
{CF, A} = A + B; if (A == 0) ZF = 1;	{CF, A} = A - B; if (A == 0) ZF = 1;	TMP = A; A = B; B = TMP; if (B == 0) ZF = 1;	A = data_in; if (A == 0) ZF = 1;
OUT A	**INC A**	**MOV A, [ADDRESS]**	**MOV A, BYTE**

data_out = A;
if (A == 0)
ZF = 1;
	{CF, A} = A + 1;
	if (A == 0)
	ZF = 1;
		A = DATA_MEM[ADD];
		if (A == 0)
		ZF = 1;
			A = ADD;
			if (A == 0)
			ZF = 1;

Instruction Execution

JZ ADDRESS	PUSH B	POP B	RCL B
if (ZF == 1) IP = ADD;	STAK_MEM[SP] = B; SP = SP - 1; if (B == 0) ZF = 1;	SP = SP + 1; B = STAK_MEM[SP]; if (B == 0) ZF = 1;	{CF, B} = {B[3:0], CF}; if (B == 0) ZF = 1;
CALL ADDRESS	**RET**	**AND A, [ADDRESS]**	**HLT**
STAK_MEM[SP] = IP; IP = ADD; SP = SP - 1;	SP = SP + 1; IP = STAK_MEM[SP];	A = A & DATA_MEM[ADD]; if (A == 0) ZF = 1;	HALT_FLAG = 1

Testbench Module



Register contains ***Machine Code***

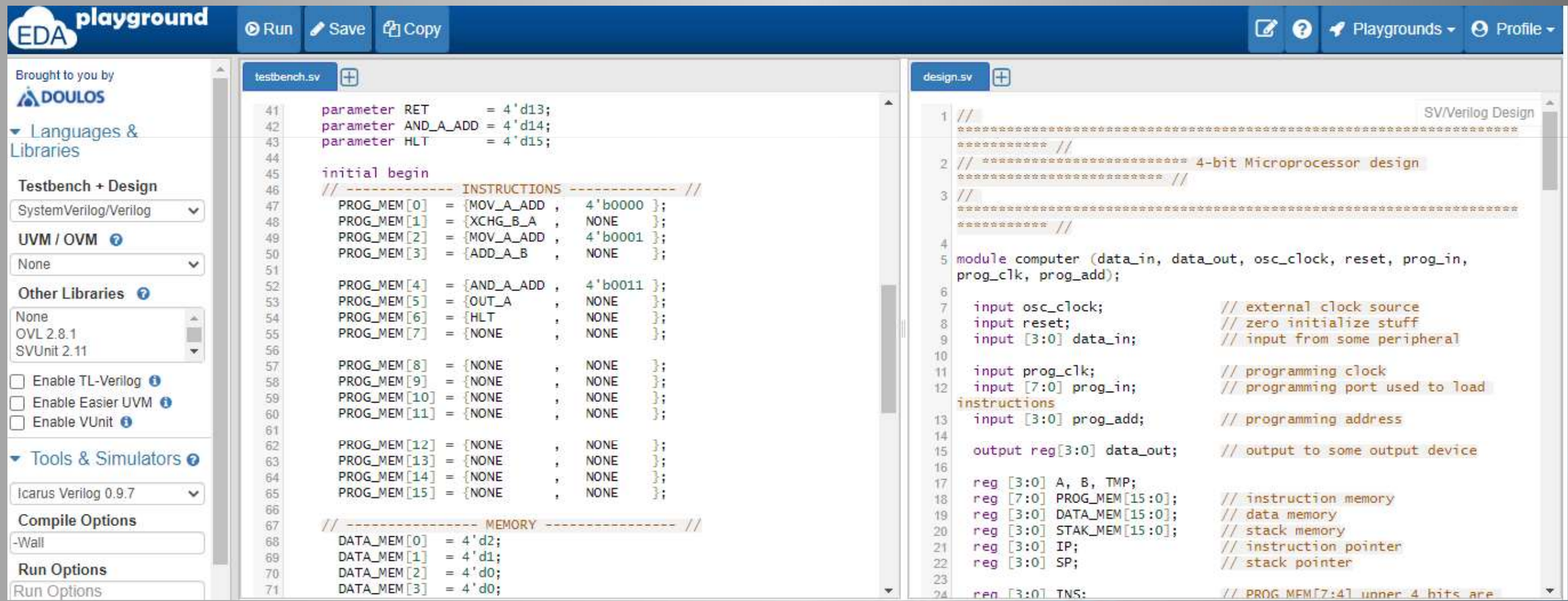
Provides ***clock*** and input to computer

Reset pin to HIGH (Computer not operating)

Loop iterates 16 times to ***upload instruction*** and data to PROG_MEM and DATA_MEM

Reset pin to LOW (Computer operating)

Sample Run-1 on EDA Playground



Sample Run-1 on EDA Playground

Testbench Code

PROG_MEM

```
initial begin
// ------------- INSTRUCTIONS ------------- //
  PROG_MEM[0]  = {MOV_A_ADD ,   4'b0000 };
  PROG_MEM[1]  = {XCHG_B_A  ,   NONE    };
  PROG_MEM[2]  = {MOV_A_ADD ,   4'b0001 };
  PROG_MEM[3]  = {ADD_A_B   ,   NONE    };

  PROG_MEM[4]  = {AND_A_ADD ,   4'b0011 };
  PROG_MEM[5]  = {OUT_A     ,   NONE    };
  PROG_MEM[6]  = {HLT       ,   NONE    };
  PROG_MEM[7]  = {NONE      ,   NONE    };

  PROG_MEM[8]  = {NONE      ,   NONE    };
  PROG_MEM[9]  = {NONE      ,   NONE    };
  PROG_MEM[10] = {NONE      ,   NONE    };
  PROG_MEM[11] = {NONE      ,   NONE    };

  PROG_MEM[12] = {NONE      ,   NONE    };
  PROG_MEM[13] = {NONE      ,   NONE    };
  PROG_MEM[14] = {NONE      ,   NONE    };
  PROG_MEM[15] = {NONE      ,   NONE    };
```

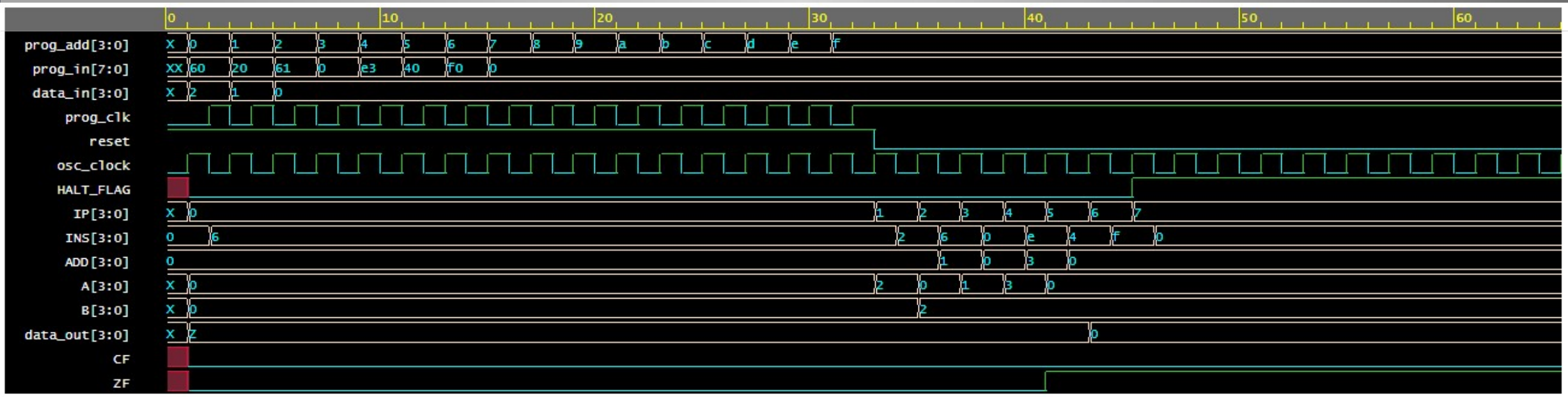
DATA_MEM

```
// ---------------- MEMORY
  DATA_MEM[0]  = 4'd2;
  DATA_MEM[1]  = 4'd1;
  DATA_MEM[2]  = 4'd0;
  DATA_MEM[3]  = 4'd0;

  DATA_MEM[4]  = 4'd0;
  DATA_MEM[5]  = 4'd0;
  DATA_MEM[6]  = 4'd0;
  DATA_MEM[7]  = 4'd0;

  DATA_MEM[8]  = 4'd0;
  DATA_MEM[9]  = 4'd0;
  DATA_MEM[10] = 4'd0;
  DATA_MEM[11] = 4'd0;

  DATA_MEM[12] = 4'd0;
  DATA_MEM[13] = 4'd0;
  DATA_MEM[14] = 4'd0;
  DATA_MEM[15] = 4'd0;
```

Address	Instruction Memory	Data Memory
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

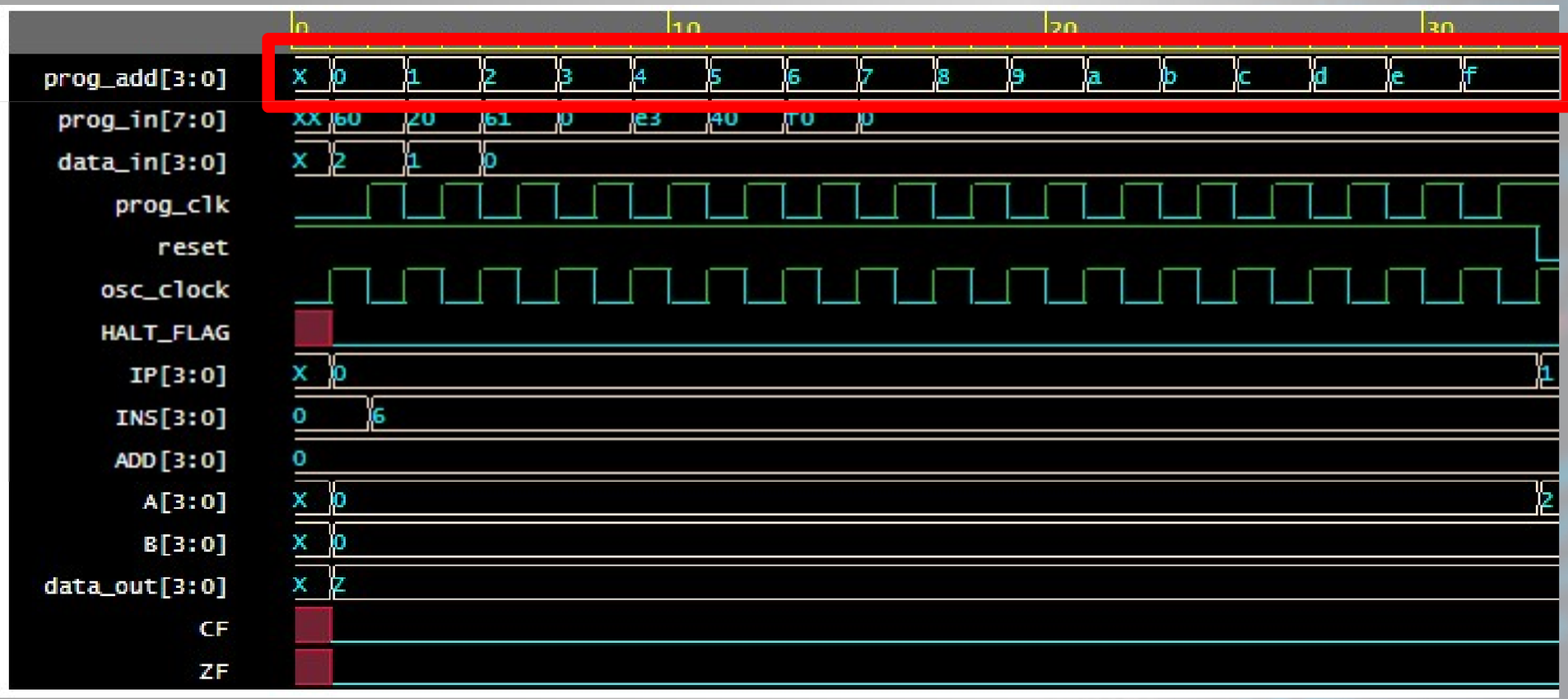
Sample Run-1 on EDA Playground

Output Waveshape



Sample Run-1 on EDA Playground

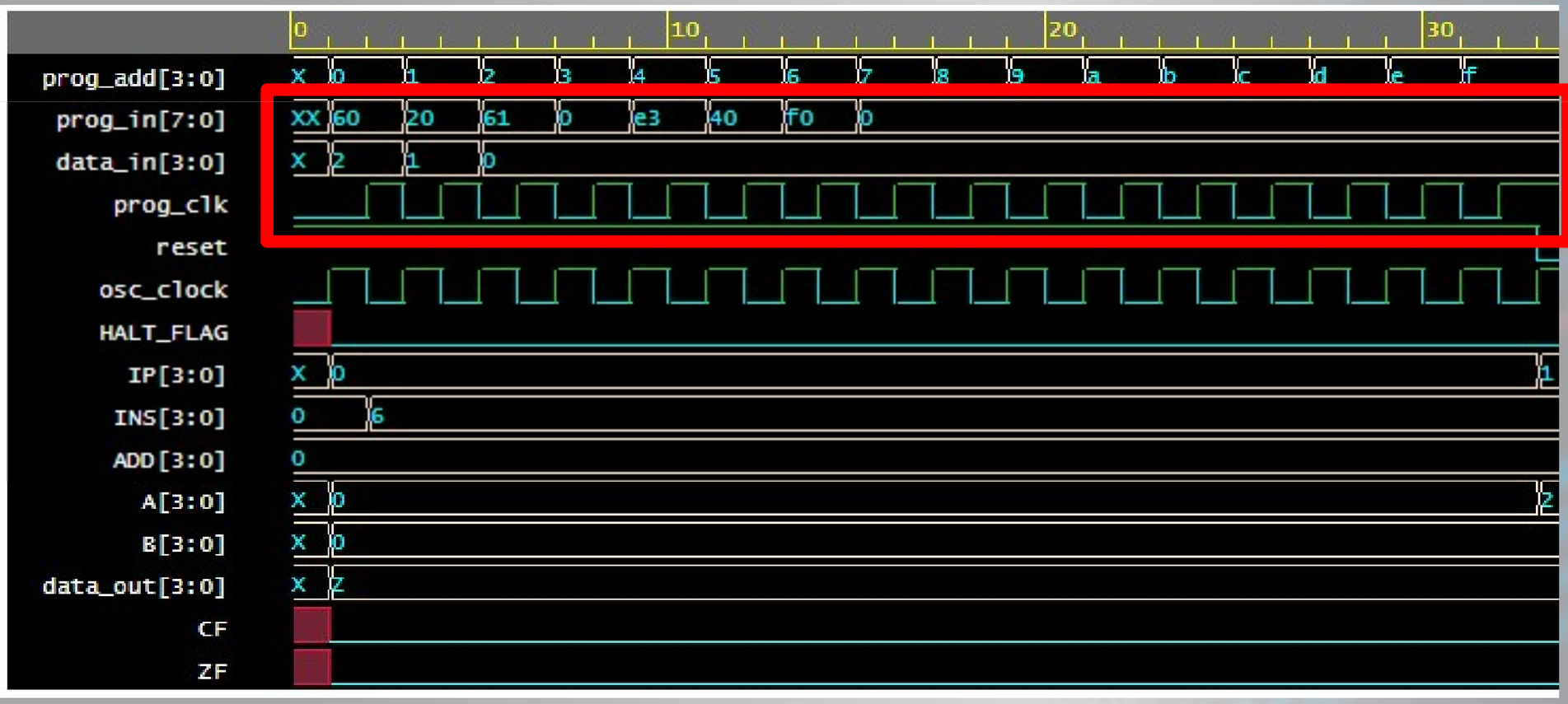
Program and Data Loading



Program address increasing from 0 to 15

Sample Run-1 on EDA Playground

Program and Data Loading



On each posedge of prog_clk, program and data is loaded into the computer address by address

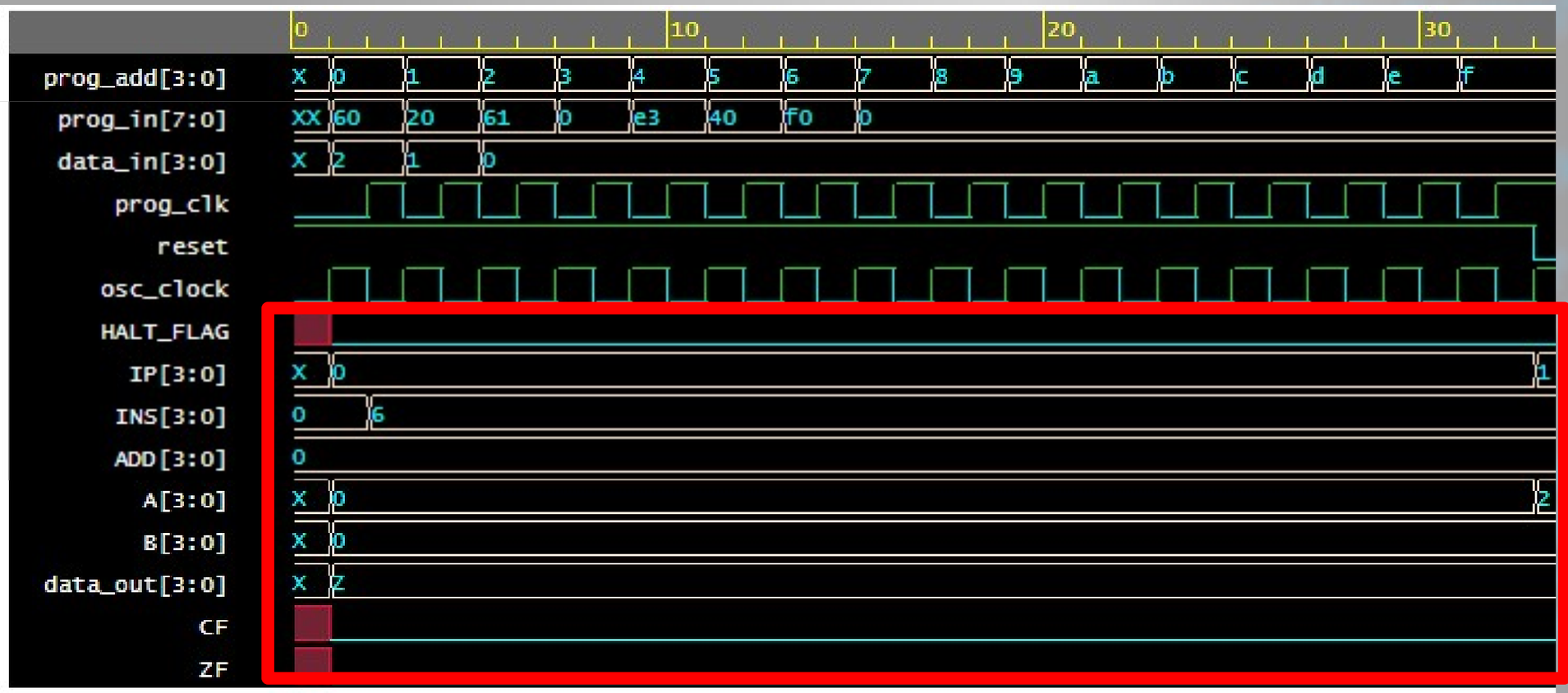
Sample Run-1 on EDA Playground

Program and Data Loading

After loading all 16 addresses, reset pin is set to 0 and computer starts executing

Sample Run-1 on EDA Playground

Program and Data Loading

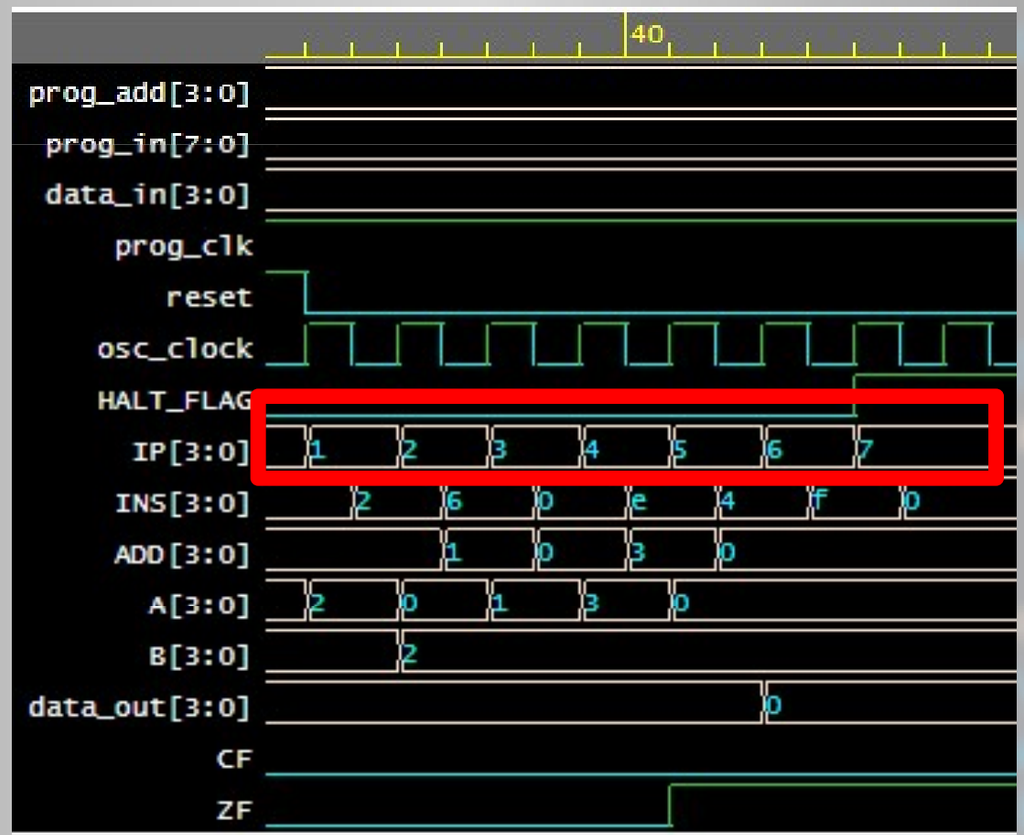


During loading, the internal registers have no/garbage data

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

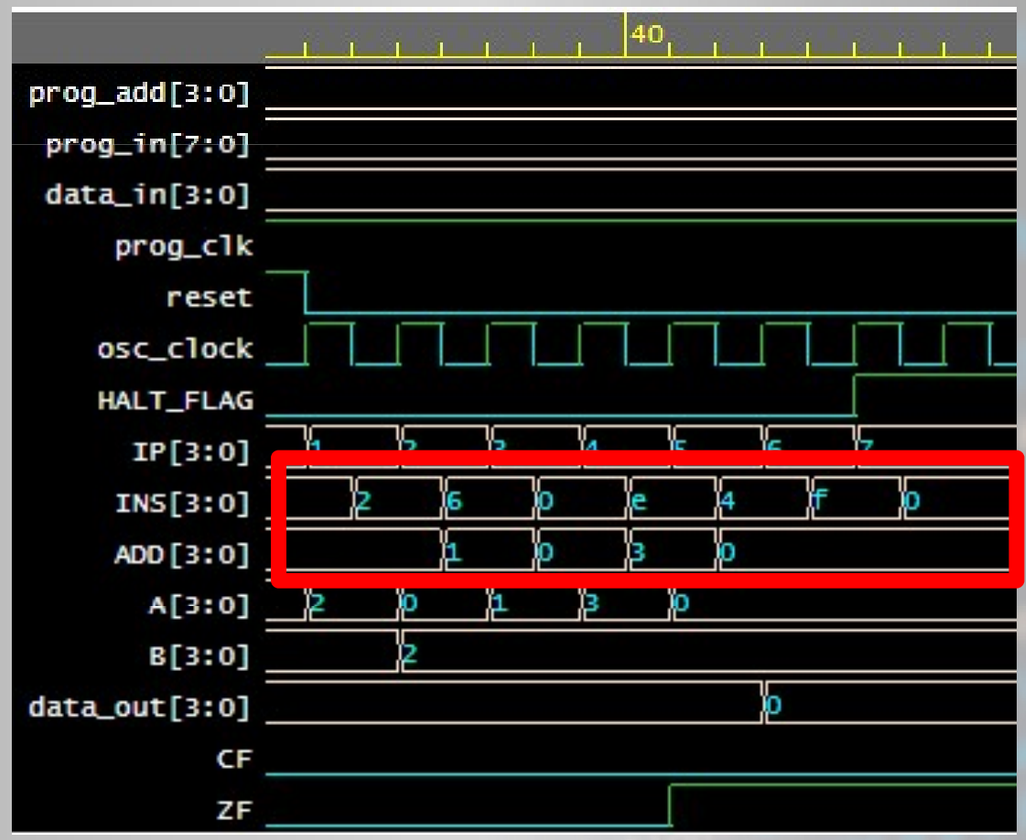
Execution



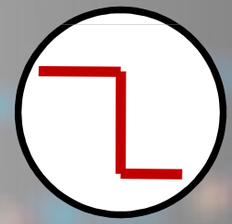
No jump/call operation, instruction pointer increases steadily

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-



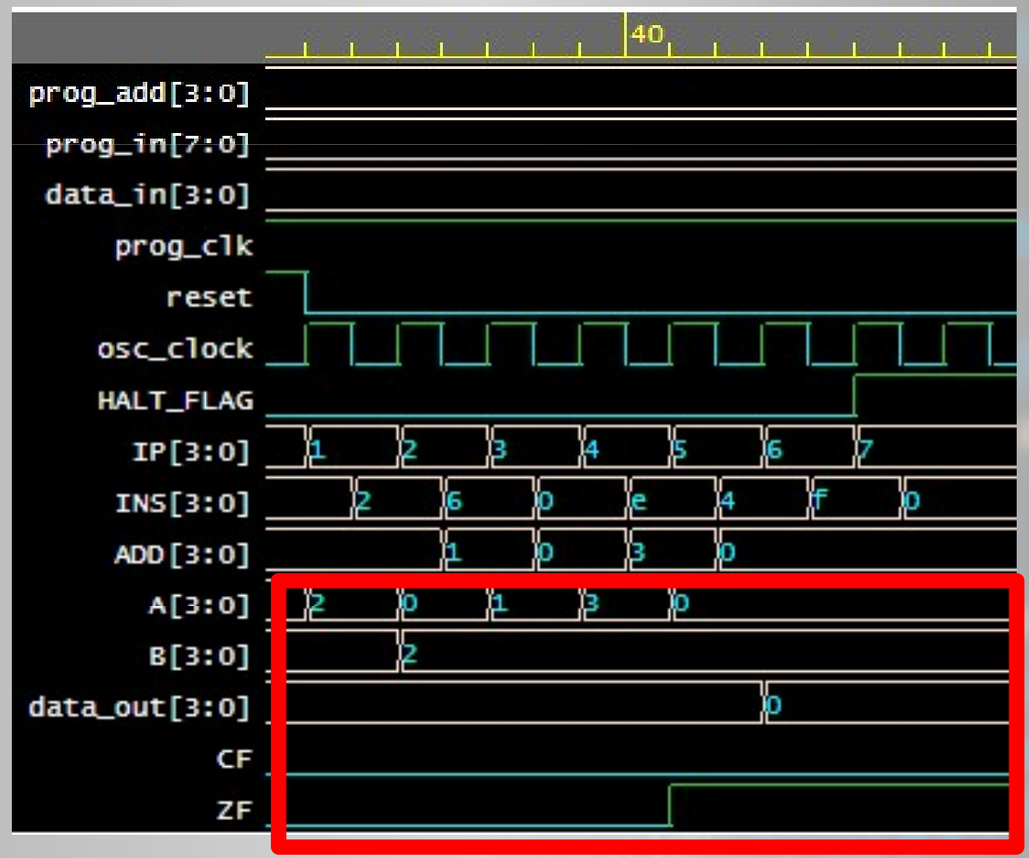
Execution



(Fetch cycle)
Instruction register and address register loads information on negative clock edge

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-



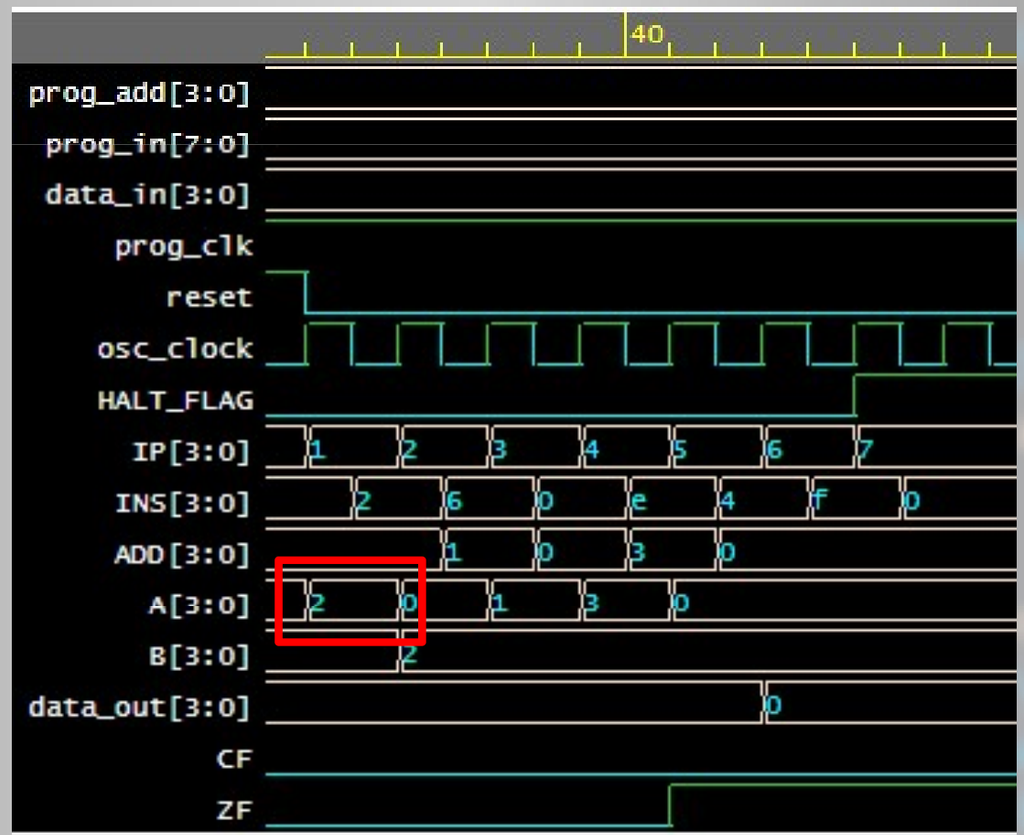
Execution

Execution of instructions are on positive edge of clock

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

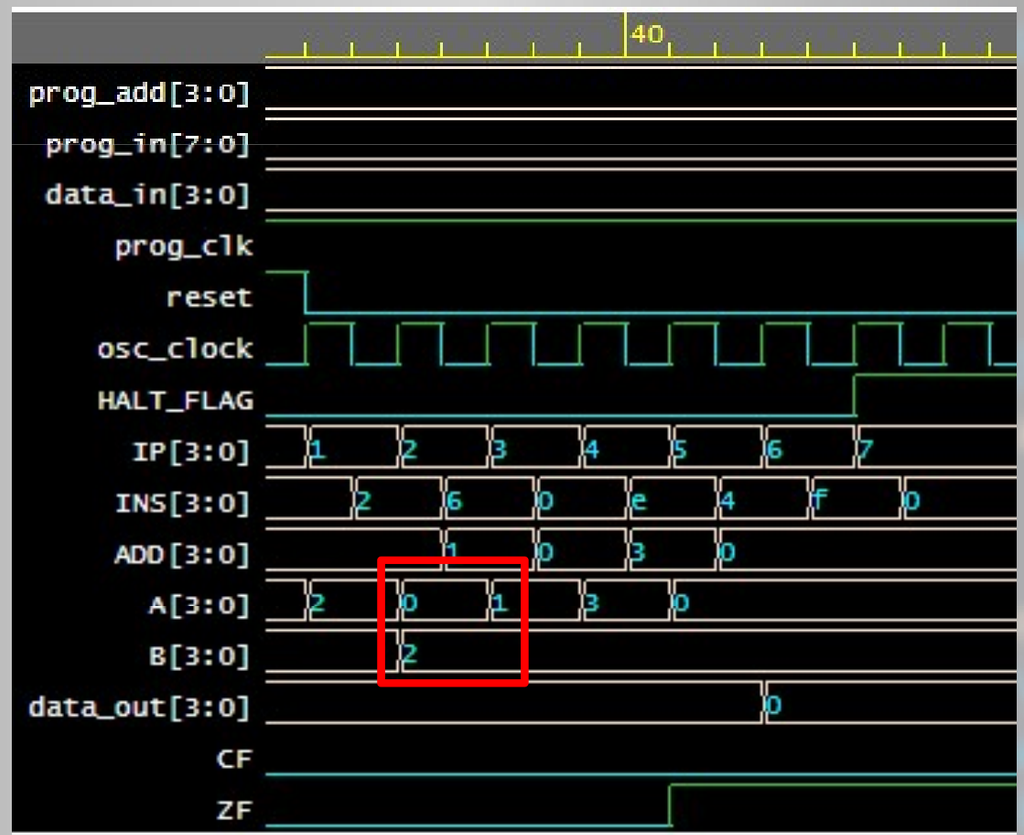


Cycle 1:
Data=2 is loaded from memory 0 into reg A

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

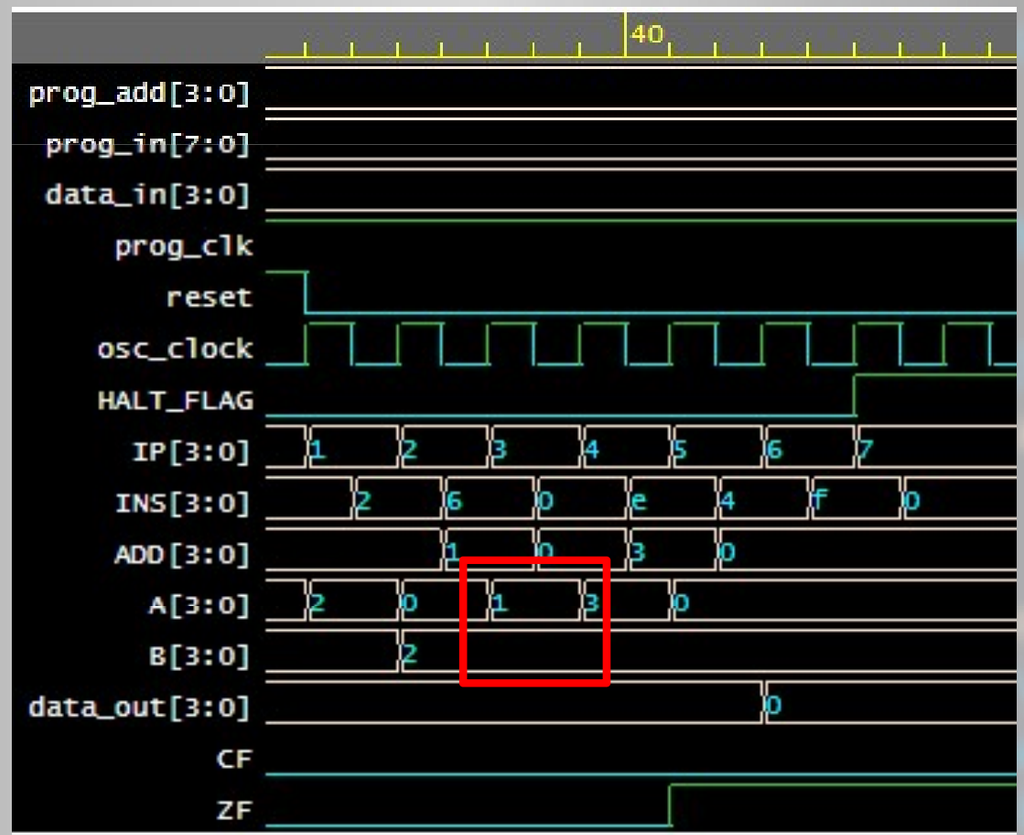


Cycle 2:
reg B, reg A swap data

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

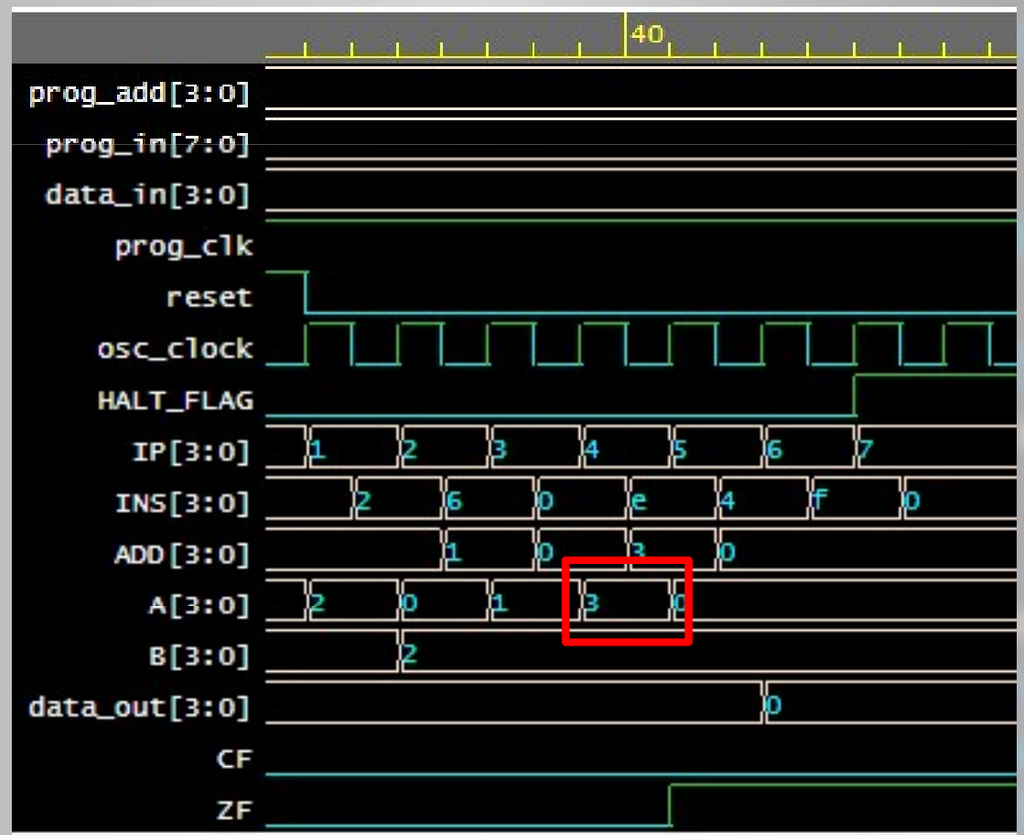


Cycle 3:
Data=1 is loaded from memory 1 into reg A

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

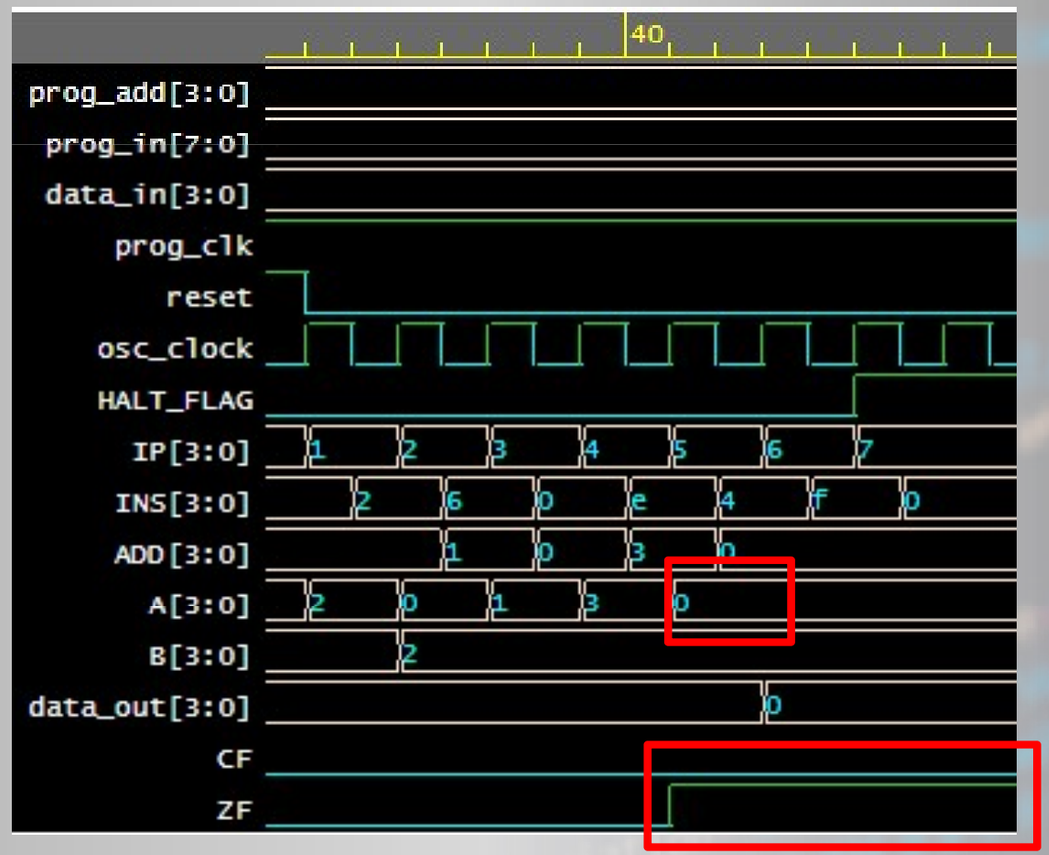


Cycle 4:
ADD A, B saves the sum on reg A

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

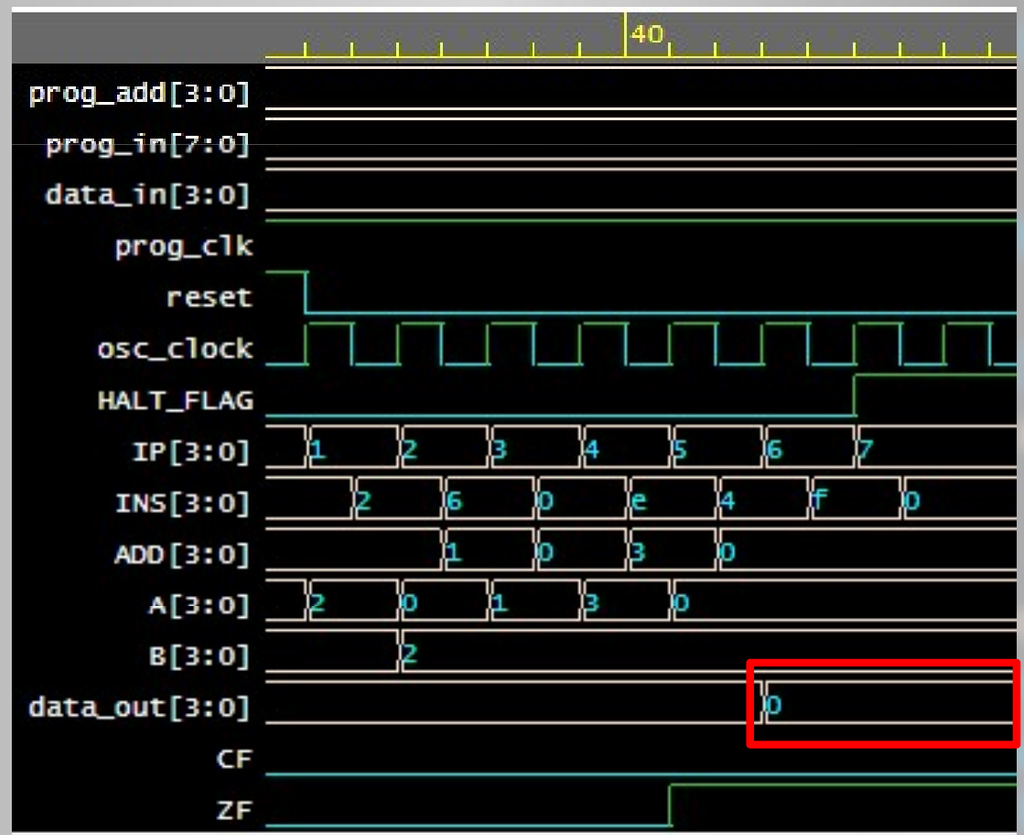


Cycle 5:
reg A AND operation with data = 0 at memory location 3, zero flag goes to 1 for 0 output

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution

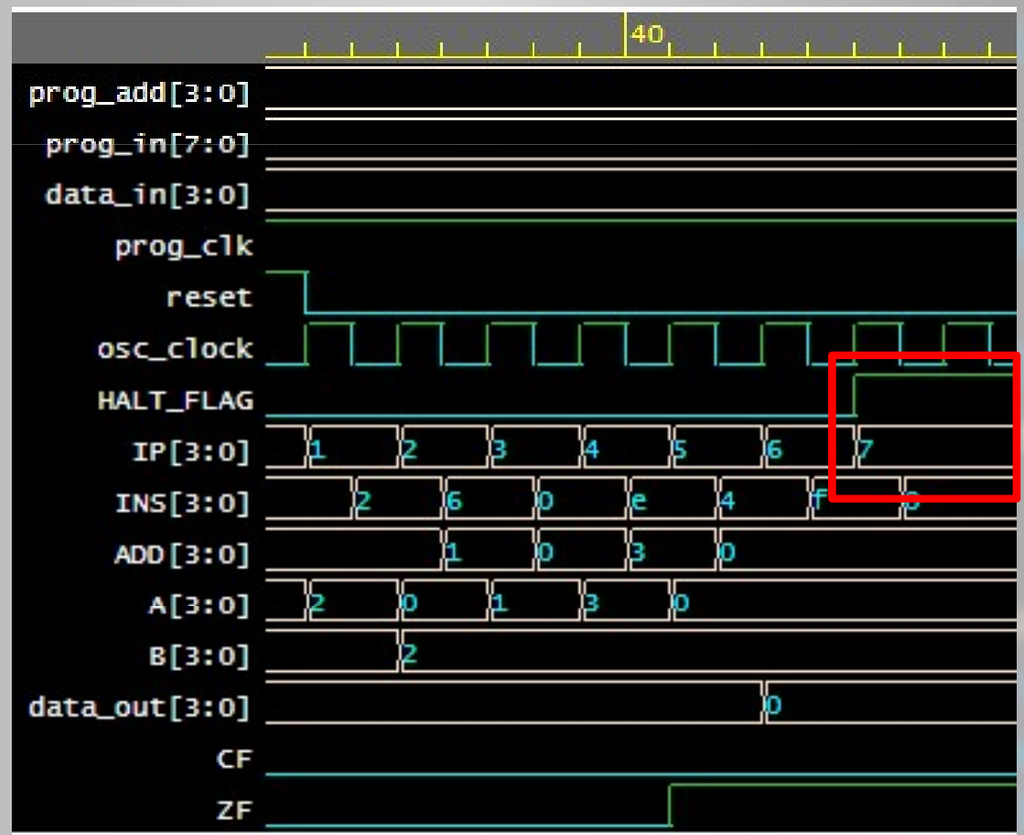


Cycle 6:
Contents of reg A are output

Sample Run-1 on EDA Playground

	Instruction Memory	Data
0	MOV A, [0000]	D2
1	XCHG B, A	D1
2	MOV A, [0001]	D0
3	ADD A, B	D0
4	AND A, [0011]	-
5	OUT A	-
6	HLT	-
7	-	-
8	-	-
9	-	-
10	-	-
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-

Execution



Cycle 7:
Computer reaches halt, instruction pointer no longer increases

Sample Run-2 on EDA Playground

Testbench Code

PROG_MEM

```
// ------------- INSTRUCTIONS ------------- //
PROG_MEM[0]  = {MOV_A_BYT   ,  4'b0010 };
PROG_MEM[1]  = {XCHG_B_A    ,  NONE    };
PROG_MEM[2]  = {MOV_A_ADD   ,  4'b0000 };
PROG_MEM[3]  = {ADD_A_B     ,  NONE    };

PROG_MEM[4]  = {OUT_A       ,  NONE    };
PROG_MEM[5]  = {CALL_ADD    ,  4'd8    };
PROG_MEM[6]  = {HLT         ,  NONE    };
PROG_MEM[7]  = {NONE        ,  NONE    };

PROG_MEM[8]  = {PUSH_B      ,  NONE    };
PROG_MEM[9]  = {XCHG_B_A    ,  NONE    };
PROG_MEM[10] = {POP_B       ,  NONE    };
PROG_MEM[11] = {INC_A       ,  NONE    };

PROG_MEM[12] = {RET         ,  NONE    };
PROG_MEM[13] = {NONE        ,  NONE    };
PROG_MEM[14] = {NONE        ,  NONE    };
PROG_MEM[15] = {NONE        ,  NONE    };
```

DATA_MEM

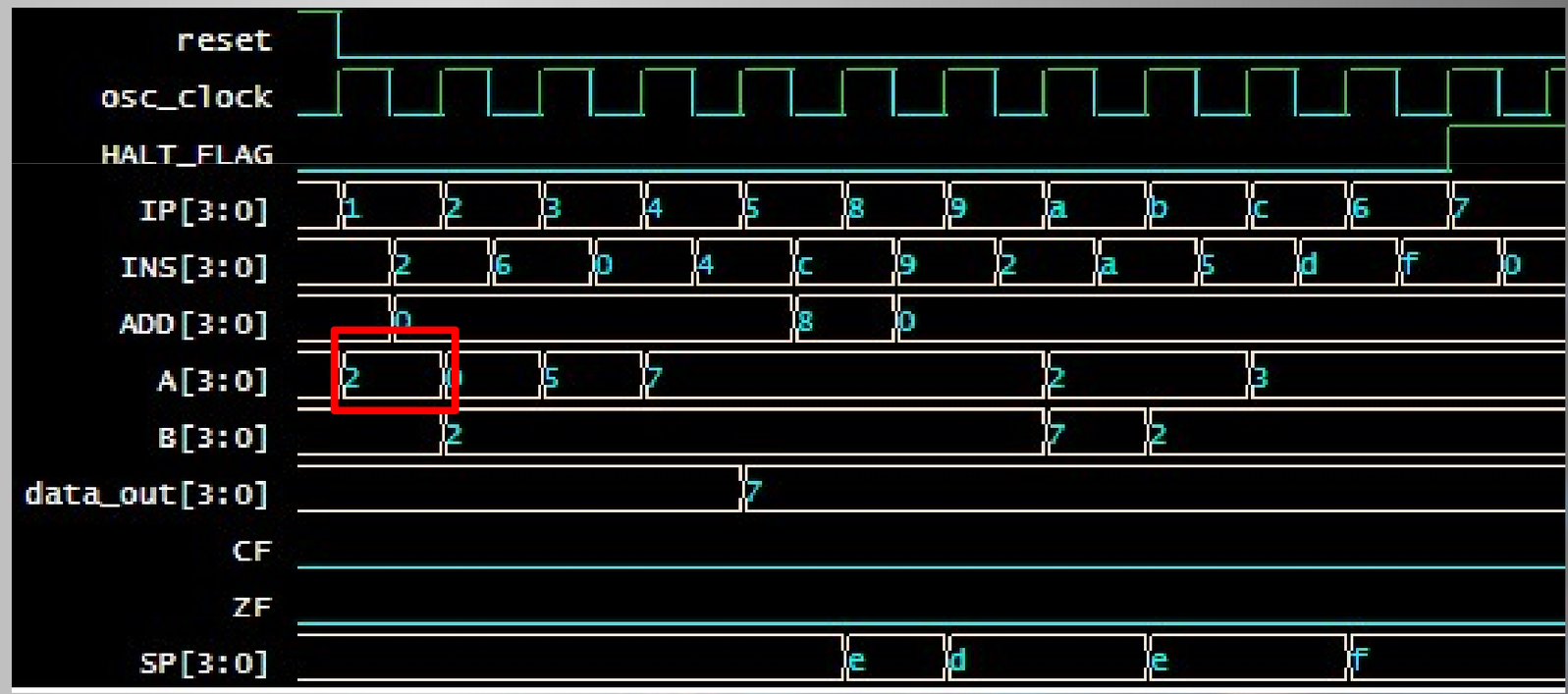
```
// --------------- MEMORY
DATA_MEM[0]  = 4'd5;
DATA_MEM[1]  = 4'd1;
DATA_MEM[2]  = 4'd0;
DATA_MEM[3]  = 4'd0;

DATA_MEM[4]  = 4'd0;
DATA_MEM[5]  = 4'd0;
DATA_MEM[6]  = 4'd0;
DATA_MEM[7]  = 4'd0;

DATA_MEM[8]  = 4'd0;
DATA_MEM[9]  = 4'd0;
DATA_MEM[10] = 4'd0;
DATA_MEM[11] = 4'd0;

DATA_MEM[12] = 4'd0;
DATA_MEM[13] = 4'd0;
DATA_MEM[14] = 4'd0;
DATA_MEM[15] = 4'd0;
```

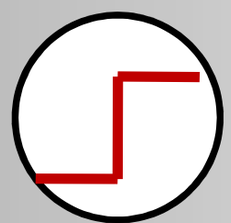
	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



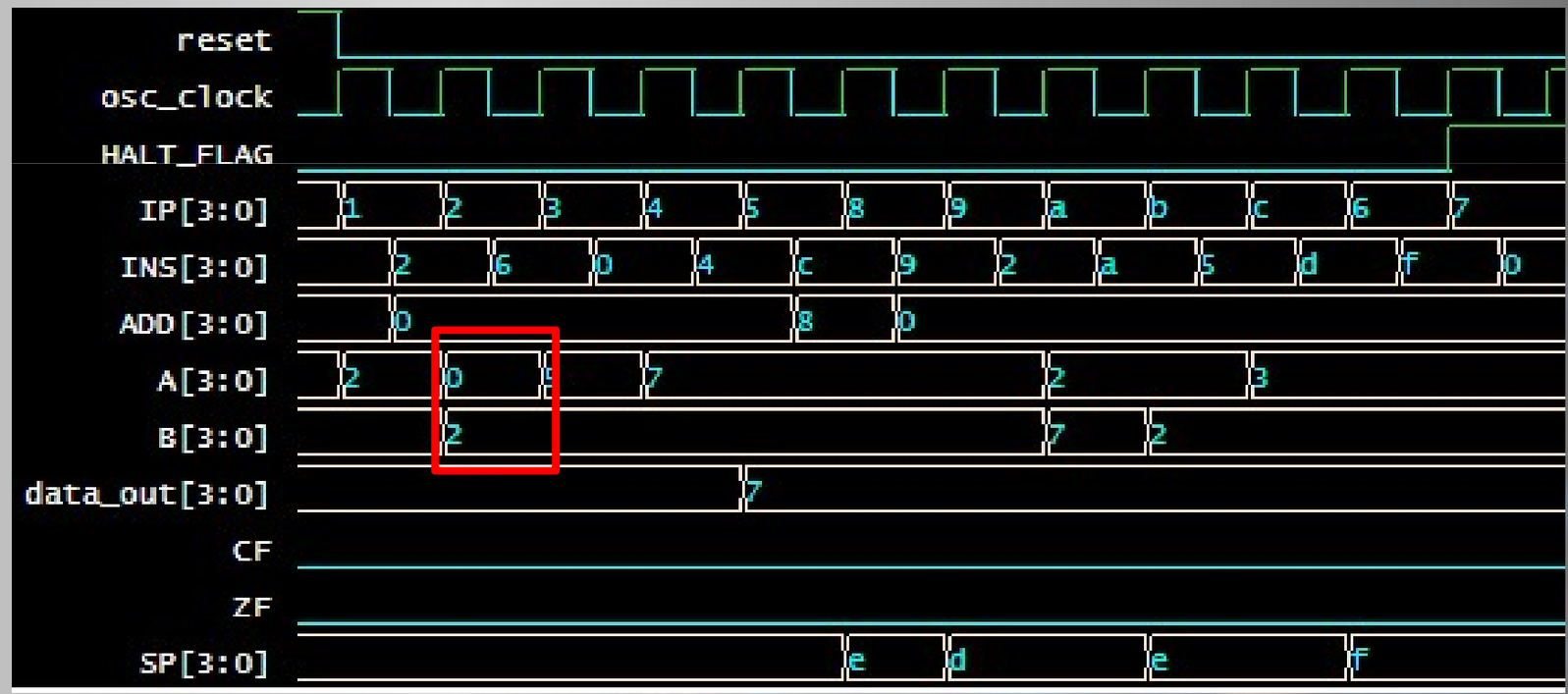


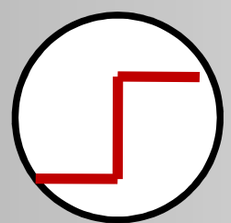
Cycle 1:
MOV A, 0010 moves value 2 into A

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



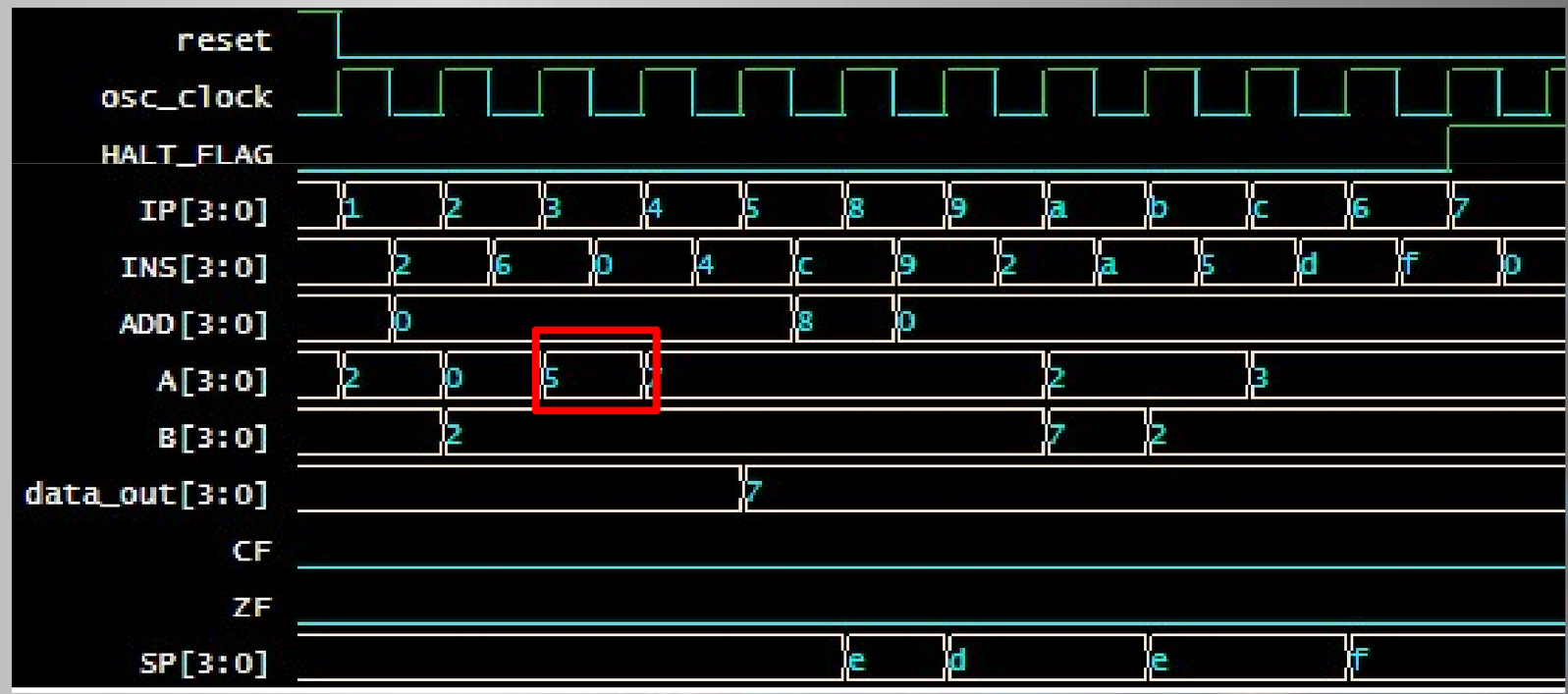


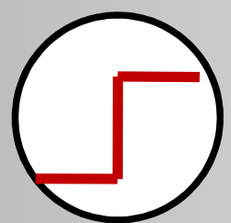
Cycle 2:
B and A registers swap values

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



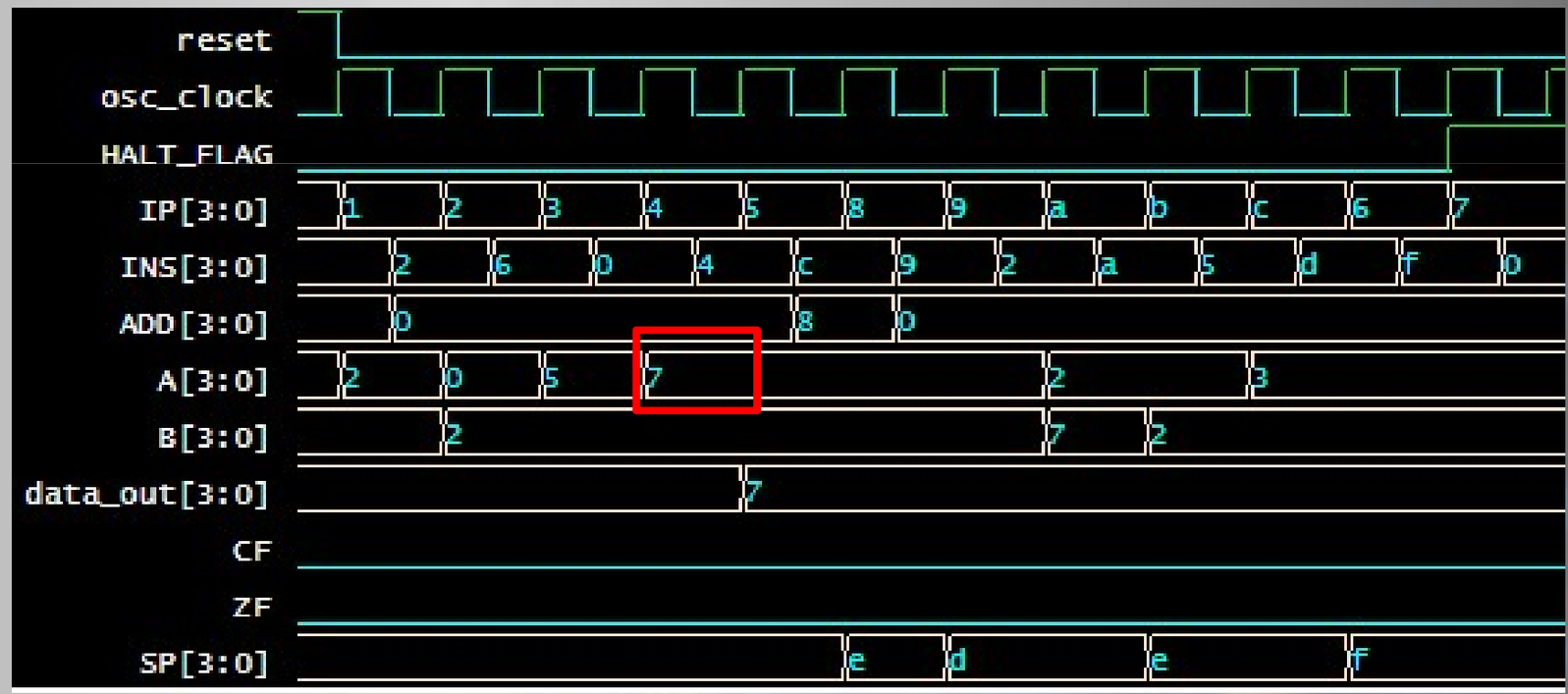


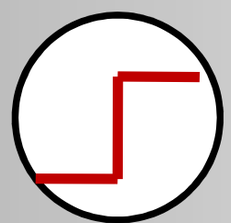
Cycle 3:
MOV A, [0000] moves value 5 on memory 0 into A

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



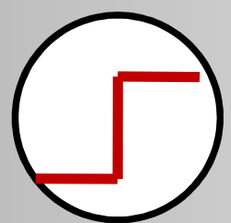


Cycle 4:
ADD A, B stores sum 2+5=7 in A

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution

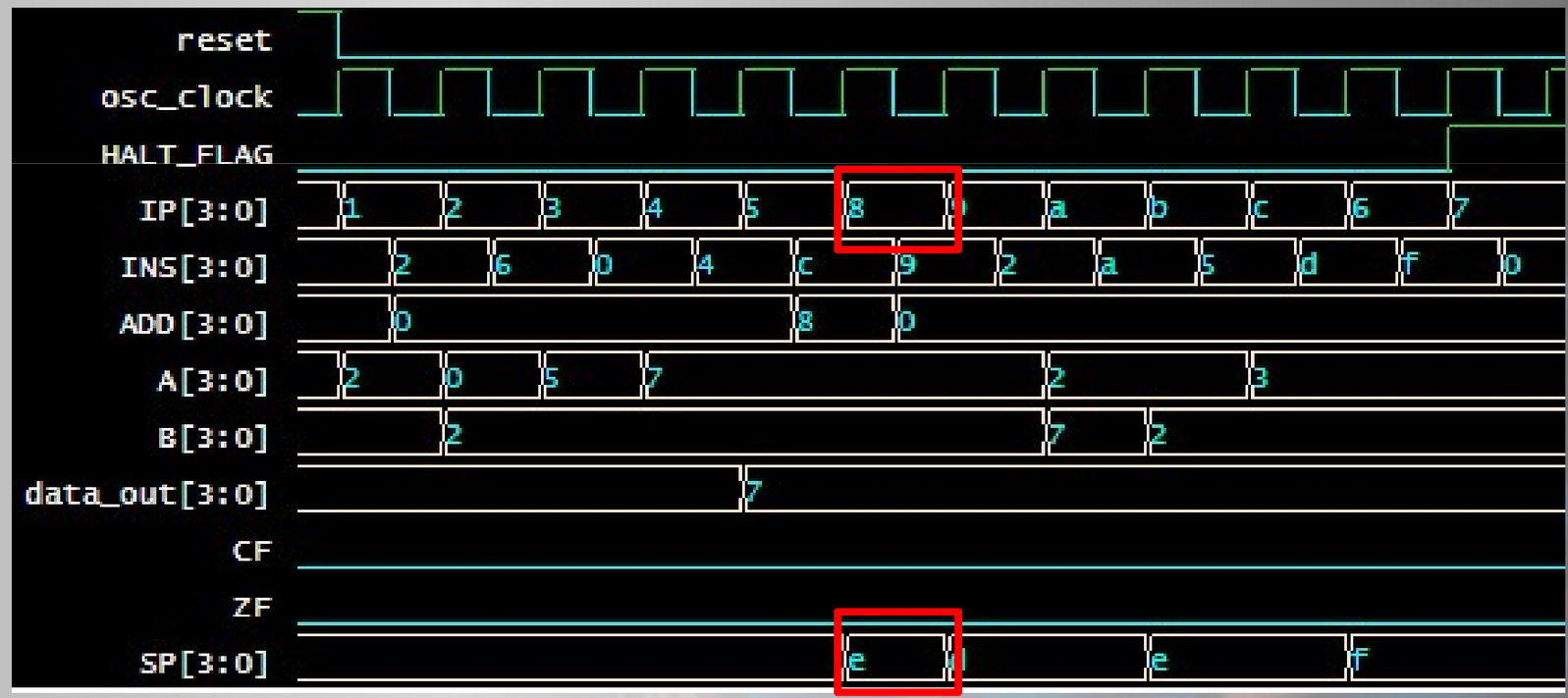


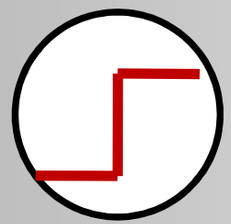
Cycle 5:
Value of A=5 output

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



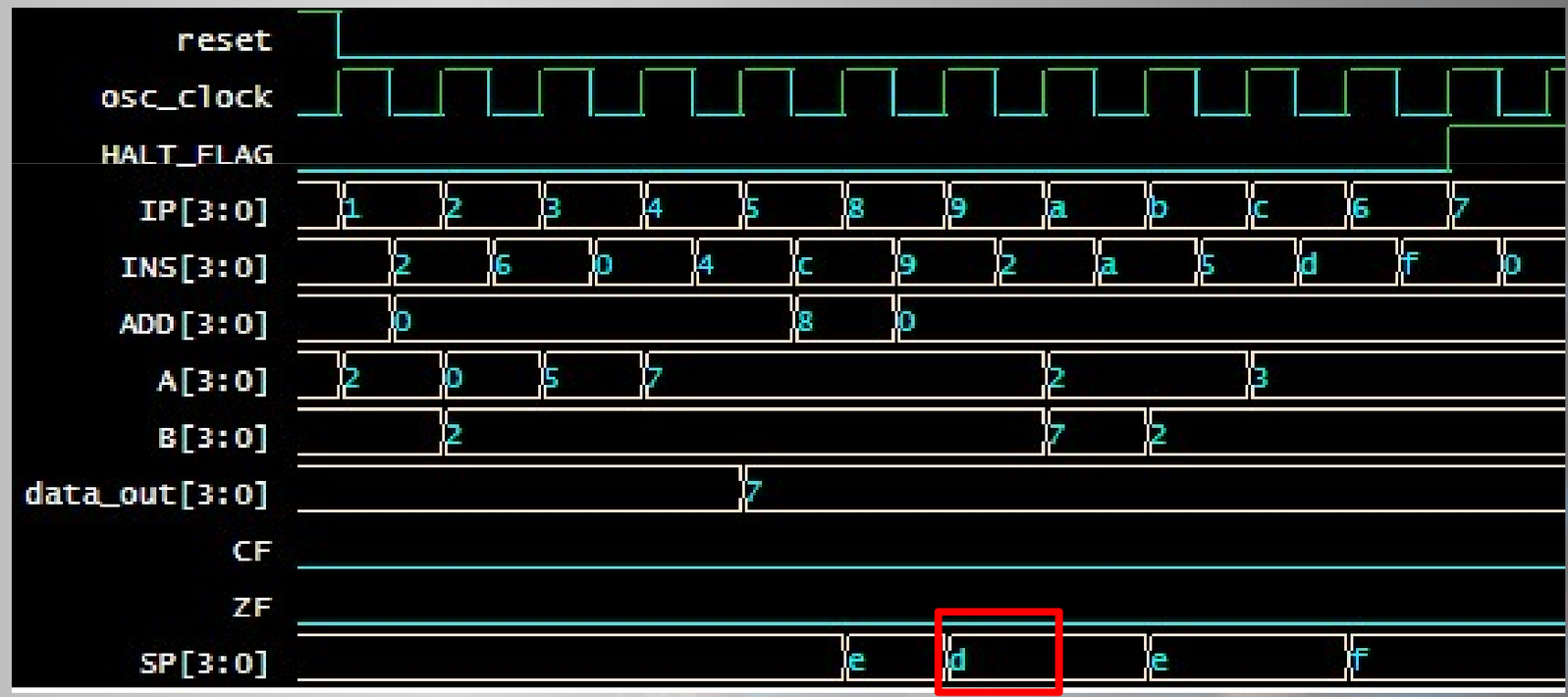


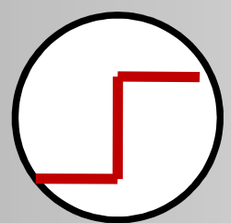
Cycle 6:
Calling instruction at memory location 8 sets instruction pointer to 8, pushes the current location to stack, stack pointer decreases from 15(f) to 14 (e)

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



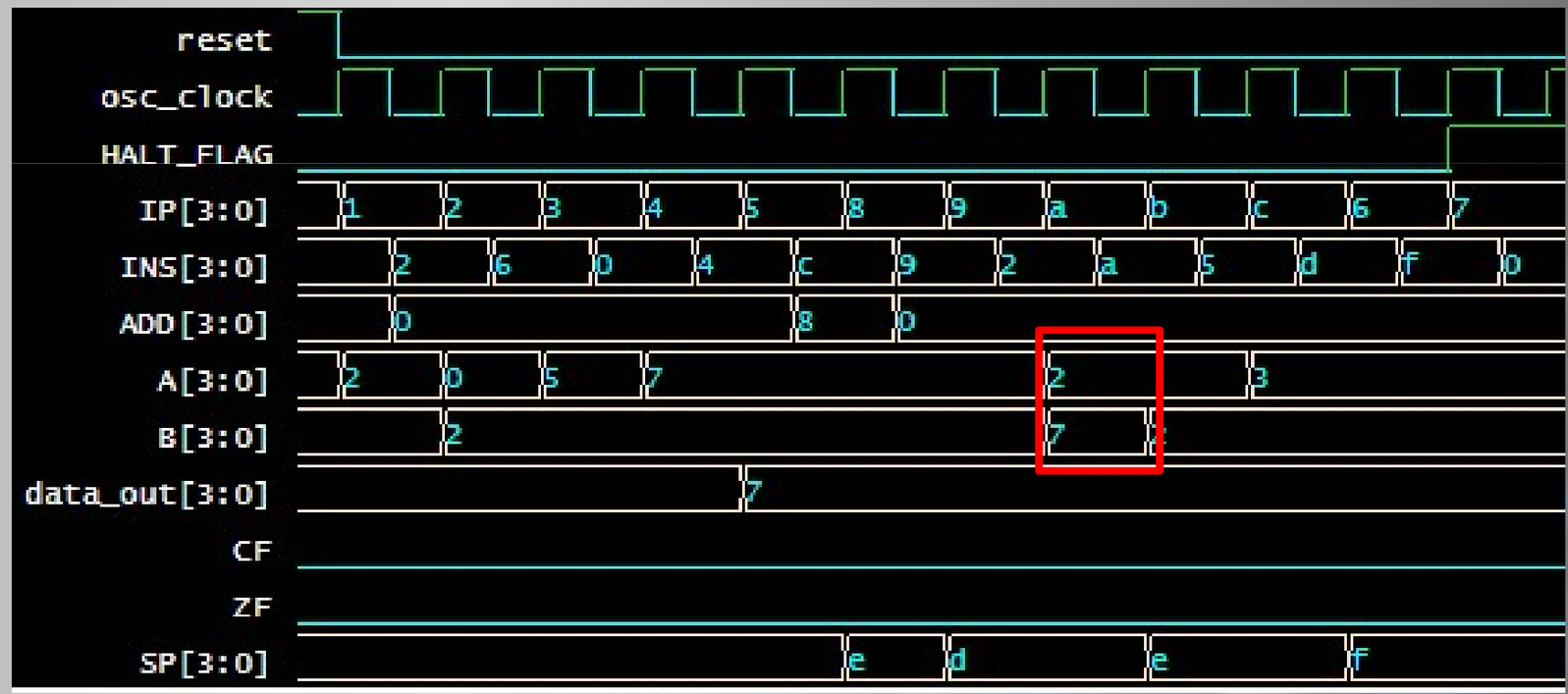


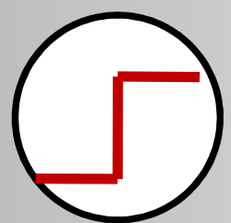
Cycle 7:
Value of B=2 is pushed to the stack, stack pointer reduced by another step

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



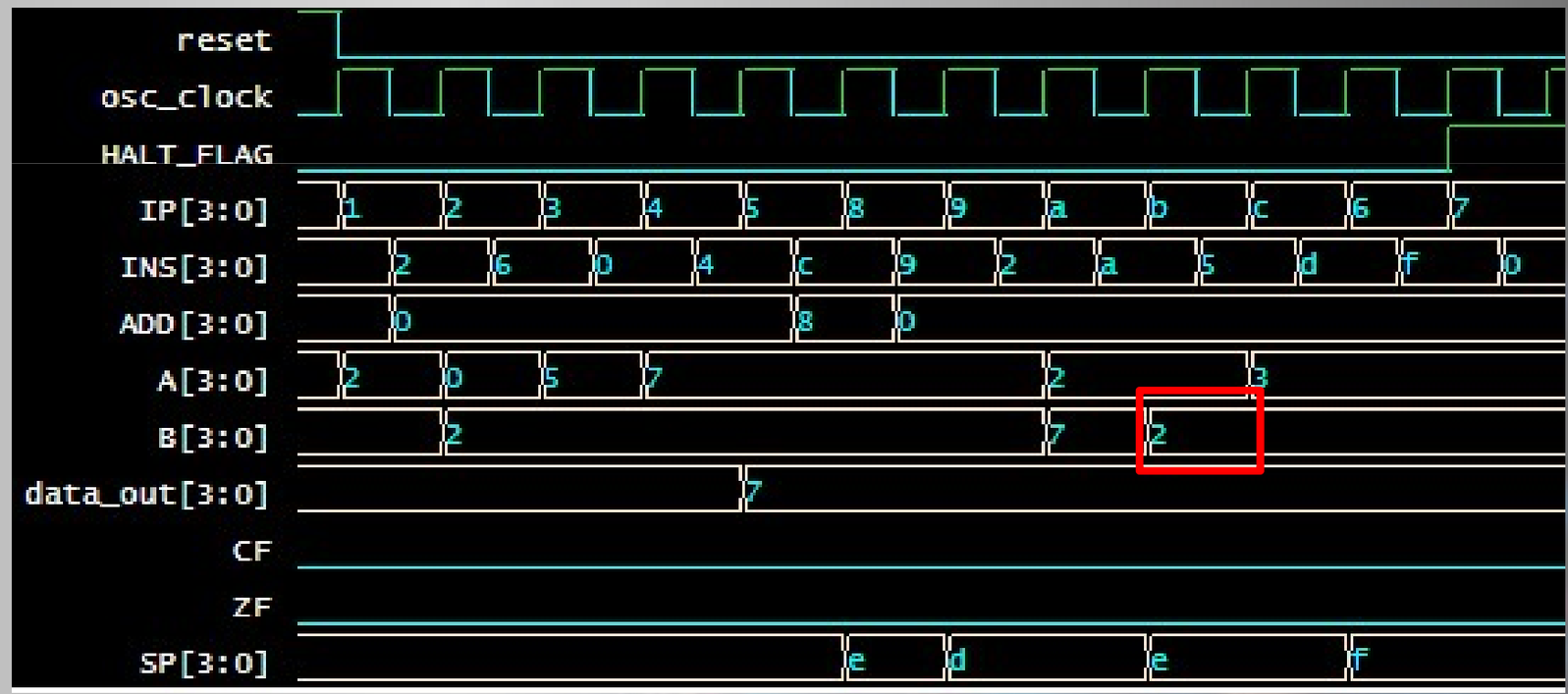


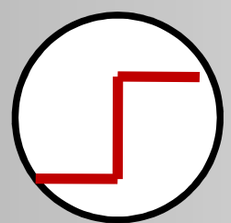
Cycle 8:
A and B swapped, A = 2 and B = 7

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



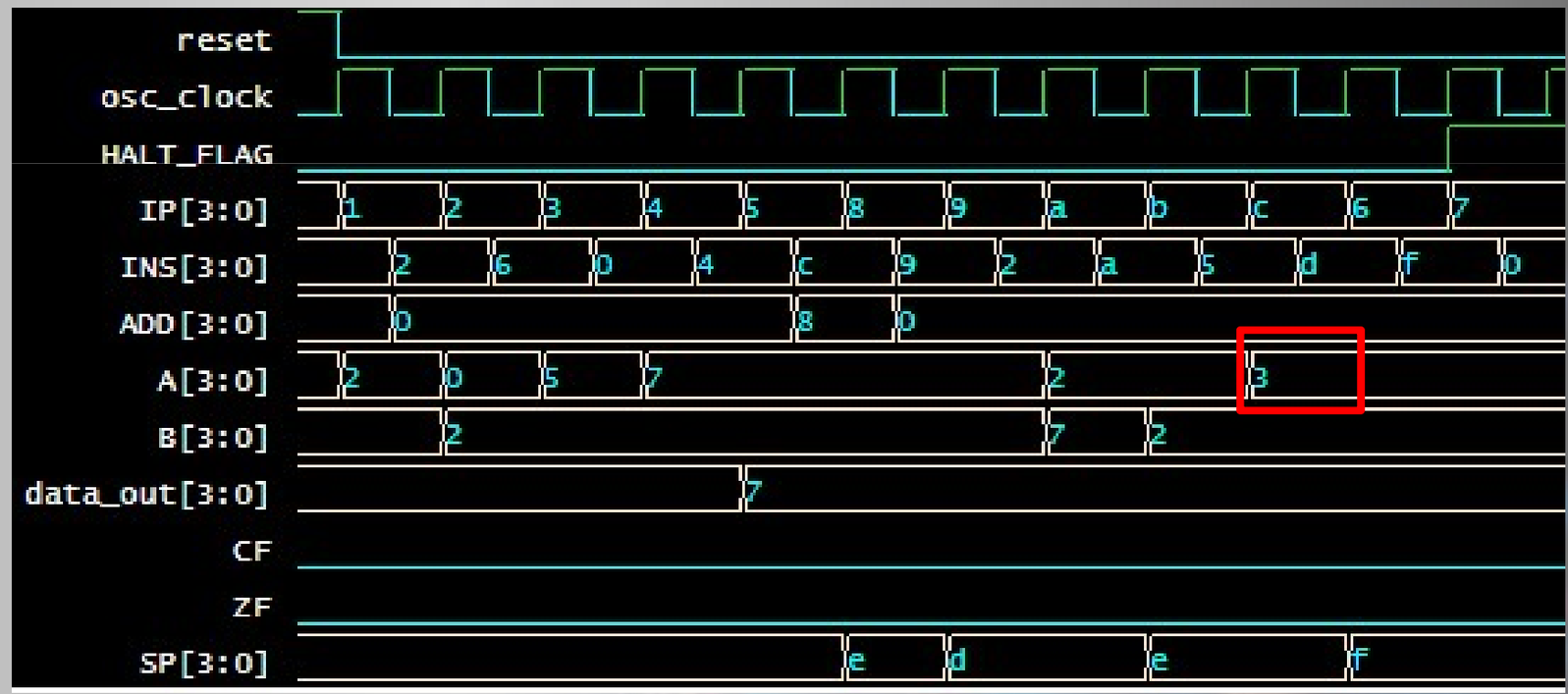


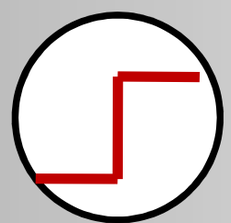
Cycle 9:
POP B increases stack pointer, pops the value 2 from stack, saves in B

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



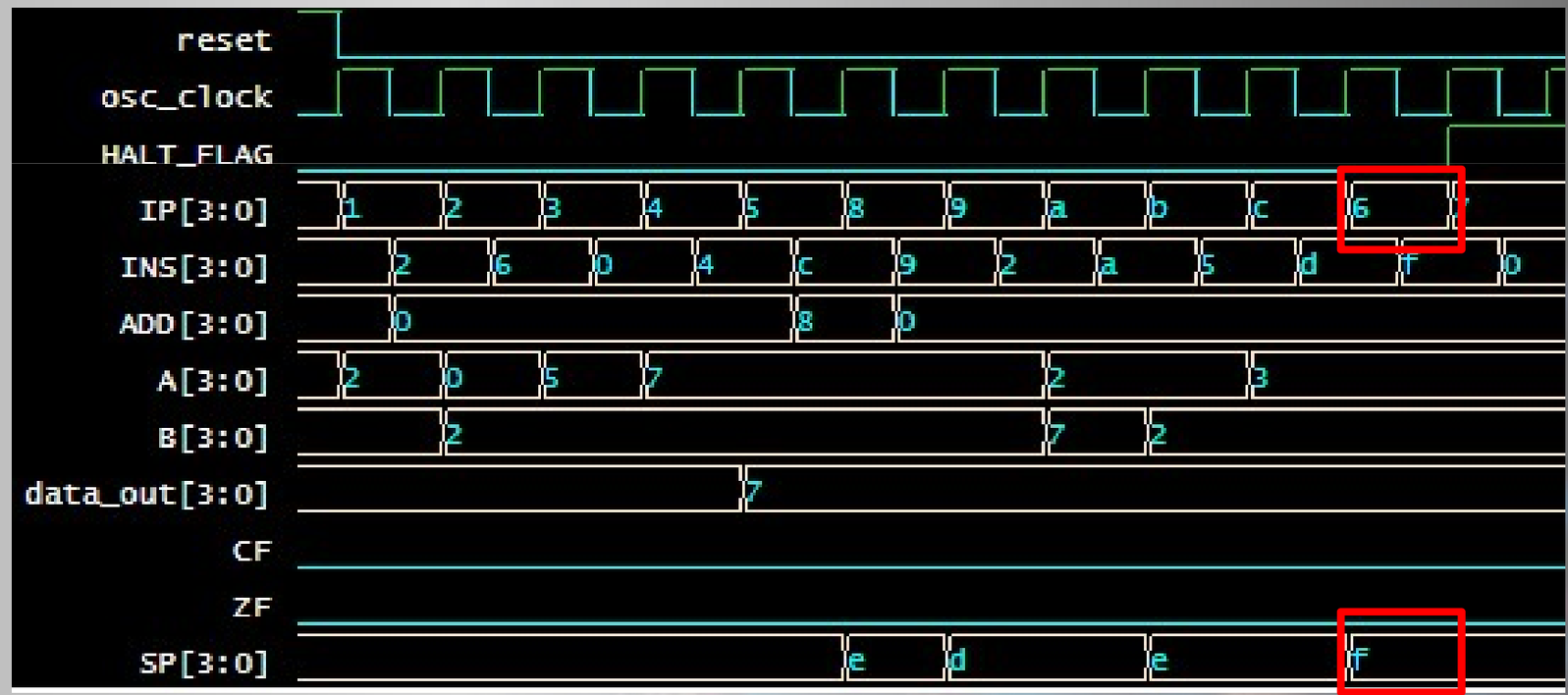


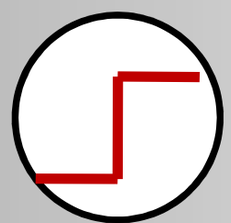
Cycle 10:
INC A increases A from 2 to 3

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution



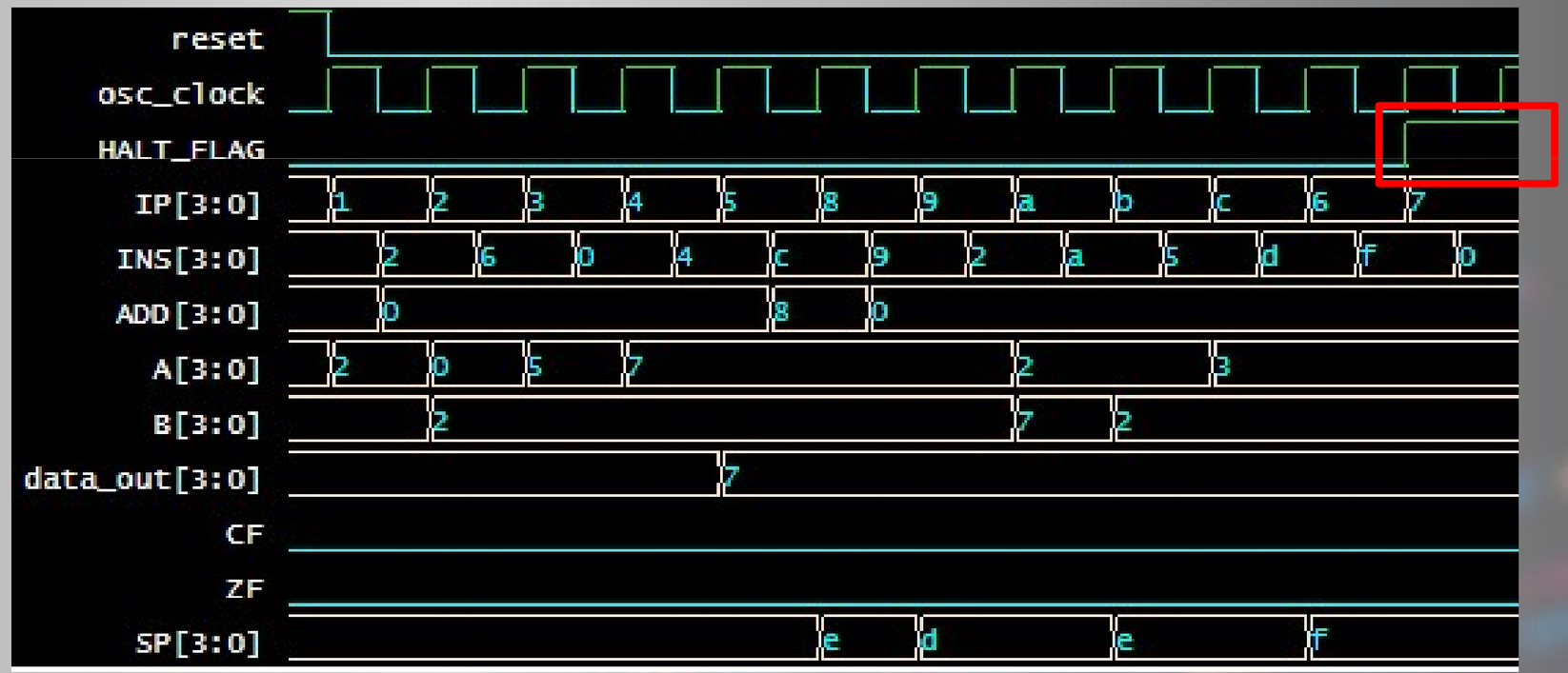


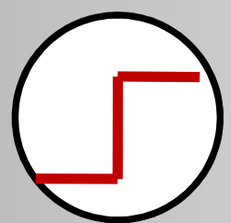
Cycle 11:
Increases stack pointer, retrieves memory location = 5, jumps to that instruction and IP increased to 6

Sample Run-2 on EDA Playground

	Instruction Memory	Data
0	MOV A, 0010	D5
1	XCHG B, A	D1
2	MOV A, [0000]	-
3	ADD A, B	-
4	OUT A	-
5	CALL [d8]	-
6	HLT	-
7	-	-
8	PUSH B	-
9	XCHG B, A	-
10	POP B	-
11	INC A	-
12	RET	-
13	-	-
14	-	-
15	-	-

Execution





Cycle 12:
Program halts

Python Assembler

```
assembler.py
233
234  def myAssembler(assembly_file):
235
236      f = open(assembly_file, "r")                # load the assembly code
237      text = f.read()                             # read the code data
238      f.close()
239
240      code, proc, data = extract_segments(text)
241      code, code_jumps, code_ins_count = pre_process_code(code)
242      if proc != []:
243          proc, proc_jumps, proc_ins_count = pre_process_code(proc)
244      else:
245          proc_jumps = {}
246          proc_ins_count = 0
247
248      print_segments(code=code, data=data, proc=proc)
249      print("Main instructions: " + str(code_ins_count))
250      print("Main jumps: ")
251      print(code_jumps)
252      print("Procedure instructions: " + str(proc_ins_count))
253      print("Procedure jumps: ")
254      print(proc_jumps)
255      machine_code, machine_data = generate_machine_code_data(data, code, co
extract_segments()
```

FEATURES:

- **Data segment** and **code segment**
- Data in **variable** format (x, y, z etc var name)
- Multiple **Jump** instructions supported
- 0-1 **procedure** supported
- Machine code generation in binary/Verilog assignment statement format from **.asm** files

USAGE:

>> python assembler.py addition_example.asm

Python Assembler

Addition_example.asm

```
.DATA
x 1
y 4
z 2
.END_DATA

.CODE
MOV A, x
XCHG B, A
MOV A, y
HERE:
CALL DECREASE
XCHG B, A
MOV A, z
SUB A, B
JZ HERE
OUT A
HLT

.PROC
DECREASE
SUB A, B
SUB A, B
RET
.END_PROC

.END_CODE
```

Command Line Output

```
_______CODE_______
['MOV A, x', 'XCHG B, A', 'MOV A, y', 'CALL DECREASE',
'XCHG B, A', 'MOV A, z', 'SUB A, B', 'JZ HERE', 'OUT
A', 'HLT']
_______PROC_______
['SUB A, B', 'SUB A, B', 'RET']
_______VARS_______
{'x': [0, 1], 'y': [1, 4], 'z': [2, 2]}
_______STOP_______
Main instructions: 10
Main jumps:
{'HERE': 3}
Procedure instructions: 3
Procedure jumps:
{}
Machine Code:
['01100000', '00100000', '01100001', '11001010',
'00100000', '01100010', '00010000', '10000011',
'01000000', '11110000', '00010000', '00010000',
'11010000']
Machine Data:
['0001', '0100', '0010']
```

Python Assembler

Addition_example.asm

```
.DATA
x 1
y 4
z 2
.END_DATA

.CODE
MOV A, x
XCHG B, A
MOV A, y
HERE:
CALL DECREASE
XCHG B, A
MOV A, z
SUB A, B
JZ HERE
OUT A
HLT

.PROC
DECREASE
SUB A, B
SUB A, B
RET
.END_PROC

.END_CODE
```

Addition_example.v

```
PROG_MEM[0] = 8'b01100000;
PROG_MEM[1] = 8'b00100000;
PROG_MEM[2] = 8'b01100001;
PROG_MEM[3] = 8'b11001010;
PROG_MEM[4] = 8'b00100000;
PROG_MEM[5] = 8'b01100010;
PROG_MEM[6] = 8'b00010000;
PROG_MEM[7] = 8'b10000011;
PROG_MEM[8] = 8'b01000000;
PROG_MEM[9] = 8'b11110000;
PROG_MEM[10] = 8'b00010000;
PROG_MEM[11] = 8'b00010000;
PROG_MEM[12] = 8'b11010000;
PROG_MEM[13] = 8'b00000000;
PROG_MEM[14] = 8'b00000000;
PROG_MEM[15] = 8'b00000000;
```

```
DATA_MEM[0] = 4'b0001;
DATA_MEM[1] = 4'b0100;
DATA_MEM[2] = 4'b0010;
DATA_MEM[3] = 4'b0000;
DATA_MEM[4] = 4'b0000;
DATA_MEM[5] = 4'b0000;
DATA_MEM[6] = 4'b0000;
DATA_MEM[7] = 4'b0000;
DATA_MEM[8] = 4'b0000;
DATA_MEM[9] = 4'b0000;
DATA_MEM[10] = 4'b0000;
DATA_MEM[11] = 4'b0000;
DATA_MEM[12] = 4'b0000;
DATA_MEM[13] = 4'b0000;
DATA_MEM[14] = 4'b0000;
DATA_MEM[15] = 4'b0000;
```

This data has to be pasted onto testbench code for proof of operation


</THANK YOU>