Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Organization and Architecture
Course Number:	COE628
Semester/Year (e.g.F2016)	W2024

Instructor:	Khalid Abdel Hafeez

Assignment/Lab Number:	4b
Assignment/Lab Title:	Data Memory Module

Submission Date:	2024/03/15
Due Date:	2024/03/15

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Sarim	Shahwar	501109286	12	SS

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.

Lab Objective:

A data path is essentially the network of hardware parts inside devices like computers or digital processors that helps data move and get processed. It's made up of various components and the links between them that transport data around the system and perform tasks on the data. Some of the components in a data path include Registers, an Arithmetic Logic Unit (ALU), Multiplexers (MUX), a Bus, a Control Unit, and a Memory interface.

Component	Description
LDAI	Load the Accumulator A immediately. Loads value directly into the accumulator register.
LDBI	Load the Accumulator B immediately. Loads value directly into the accumulator register.
STA	Store the value of the register in the memory for A.
STB	Store the value of the register in the memory for B.
LDA	Load Accumulator A. Loads value into memory.
LDB	Load Accumulator B. Loads value into memory.
LUI	Load upper immediately. Loads value into the upper half of the register.
JMP	Alter the flow of execution by setting the program counter to the address specified by the operand.
ANDI	“AND” immediate period operation between “AND” and immediate value
ADDI	adds an immediate value to the register by using a register file adder and another separate register file to store the immediate value
ORI	“OR” immediate. “OR” operation between register and immediate value
ADD	The function to add contents to registers usually registers “A” and “B”
SUB	The function to subtract contents of two registers usually registers “A” and “B”
DECA	Decrement accumulator. decrease the value in the accumulator by 1
INCA	Increments a register value by using a registered file and an adder
ROL	Rotate left. Shifts all bits in a register/memory to the Left
ROR	Rotate right, Shifts all bits in a register/ memory to the Right
CLRA	Clears accumulator “A”, sets value to 0
CLRB	Clears accumulator “B”, sets value to 0
UZE	Unsigned Zero Extension.Denoted as unsigned
LZE	Leading zero extension. add zeros to convert 8 Bits to 32 bits
RED	gets 8-bits from 32-bits

Experiment Details:

Data-Path VHDL:

DataPath.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity DataPath is
7     port(
8        --Clock Signal
9        Clk, mClk    : in std_logic;
10
11       --Memory Signals
12       WEN, EN      : in std_logic;
13
14       --Register Control Signals (CIR and LD)
15       Clr_A, Ld_A     : in std_logic;
16       Clr_B, Ld_B     : in std_logic;
17       Clr_C, Ld_C     : in std_logic;
18       Clr_Z, Ld_Z     : in std_logic;
19       ClrPC, Ld_PC    : in std_logic;
20       ClrIR, Ld_IR    : in std_logic;
21
22       --Register Outputs
23       Out_A     : out std_logic_vector(31 downto 0);
24       Out_B     : out std_logic_vector(31 downto 0);
25       Out_C     : out std_logic;
26       Out_Z     : out std_logic;
27       Out_PC    : out std_logic_vector(31 downto 0);
28       Out_IR    : out std_logic_vector(31 downto 0);
29
30       --Special Inputs to PC
31       Inc_PC    : in std_logic;
32
33       --Address and Data Bus signals for debugging
34       ADDR_OUT     : out std_logic_vector(31 downto 0);
35       DATA_IN      : in std_logic_vector(31 downto 0);
36       DATA_BUS,
```

```
37          MEM_OUT,
38          MEM_IN      : out std_logic_vector(31 downto 0);
39          MEM_ADDR    : out unsigned(7 downto 0);
40
41          --Various MUX controls
42          DATA_MUX    : in std_logic_vector(1 downto 0);
43          REG_MUX     : in std_logic;
44          A_MUX,
45          B_MUX       : in std_logic;
46          IM_MUX1     : in std_logic;
47          IM_MUX2     : in std_logic_vector(1 downto 0);
48
49          --ALU Operations
50          ALU_Op      : in std_logic_vector(2 downto 0)
51      );
52  end DataPath;
53
54
55  architecture Behavior of DataPath is
56
57      -- Components
58      -- data memory
59      component data_mem is
60          port(
61              clk      : in  std_logic;
62              addr     : in  unsigned(7 downto 0);
63              data_in  : in  std_logic_vector(31 downto 0);
64              wen      : in  std_logic;
65              en       : in  std_logic;
66              data_out : out std_logic_vector(31 downto 0)
67          );
68      end component;
69
70      -- register32
71      component register32 is
72          port(
73              d     :  in  std_logic_vector(31 DOWNTO 0);
74              ld    :  in  std_logic;
75              clr   :  in  std_logic;
76              clk   :  in  std_logic;
77              Q     :  out std_logic_vector(31 DOWNTO 0)
78          );
79      end component;
80
81      -- program counter
82      component pc is
83          port (
84              clr   :  in    std_logic;
85              clk   :  in    std_logic;
86              ld    :  in    std_logic;
87              inc   :  in    std_logic;
88              d     :  in    std_logic_vector(31 downto 0);
89              q     :  out   std_logic_vector(31 downto 0)
90          );
91      end component;
92
93      -- lze
94      component LZE is
95          port (
96              LZE_in : in std_logic_vector(31 downto 0);
97              LZE_out : out std_logic_vector(31 downto 0)
98          );
99      end component;
100
101     -- uze
102     component UZE is
103         port(
104             UZE_in   :  in std_logic_vector(31 downto 0);
105             UZE_out  :  out std_logic_vector(31 downto 0)
106         );
107     end component;
108
```

```
100
101     -- uze
102     component UZE is
103         port(
104             UZE_in    :  in std_logic_vector(31 downto 0);
105             UZE_out   :  out std_logic_vector(31 downto 0)
106         );
107     end component;
108
109     -- red
110     component RED is
111         port (
112             RED_in    :  in std_logic_vector(31 downto 0);
113             RED_out   :  out unsigned(7 downto 0)
114         );
115     end component;
116
117     -- mux2to1
118     component mux2to1 is
119         port (
120             s         : in  std_logic;
121             w0, w1    : in  std_logic_vector(31 downto 0);
122             f         : out std_logic_vector(31 downto 0)
123         );
124     end component;
125
126     -- mux4to1
127     component mux4to1 is
128         port(
129             s               :  in  std_logic_vector(1 downto 0);
130             X1, X2, X3, X4  :  in  std_logic_vector(31 downto 0);
131             f               :  out std_logic_vector(31 downto 0)
132         );
133     end component;
134
135     -- alu
```

```
136     component alu is
137         port (
138             a         : in  std_logic_vector(31 downto 0);
139             b         : in  std_logic_vector(31 downto 0);
140             op        : in  std_logic_vector(2 downto 0);
141             result    : out std_logic_vector(31 downto 0);
142             cout      : out std_logic;
143             zero      : out std_logic
144         );
145     end component;
146
147     -- Signals
148     signal IR_OUT            : std_logic_vector(31 downto 0);
149     signal data_bus_s        : std_logic_vector(31 downto 0);
150     signal LZE_out_PC        : std_logic_vector(31 downto 0);
151     signal LZE_out_A_Mux     : std_logic_vector(31 downto 0);
152     signal LZE_out_B_Mux     : std_logic_vector(31 downto 0);
153     signal RED_out_Data_Mem  : unsigned(7 downto 0);
154     signal A_Mux_out         : std_logic_vector(31 downto 0);
155     signal B_Mux_out         : std_logic_vector(31 downto 0);
156     signal reg_A_out         : std_logic_vector(31 downto 0);
157     signal reg_B_out         : std_logic_vector(31 downto 0);
158     signal reg_Mux_out       : std_logic_vector(31 downto 0);
159     signal data_mem_out      : std_logic_vector(31 downto 0);
160     signal UZE_IM_MUX1_out   : std_logic_vector(31 downto 0);
161     signal IM_MUX1_out       : std_logic_vector(31 downto 0);
162     signal LZE_IM_MUX2_out   : std_logic_vector(31 downto 0);
163     signal IM_MUX2_out       : std_logic_vector(31 downto 0);
164     signal ALU_out           : std_logic_vector(31 downto 0);
165     signal zero_flag         : std_logic;
166     signal carry_flag        : std_logic;
167     signal temp              : std_logic_vector(30 downto 0) := (others => '0');
168     signal out_pc_sig        : std_logic_vector(31 downto 0);
169 begin
170     IR:   register32 port map(
171         data_bus_s,
```

```
        data_bus_s,
        Ld_IR,
        ClrIR,
        Clk,
        IR_OUT
    );

    LZE_PC:  LZE port map(
        IR_OUT,
        LZE_out_PC
    );

    PC0:  PC port map(
        CLRPC,
        Clk,
        ld_PC,
        INC_PC,
        LZE_out_PC,
        --ADDR_OUT,
        out_pc_sig
    );

    LZE_A_Mux:  LZE port map(
        IR_OUT,
        LZE_out_A_Mux
    );

    A_mux0:  mux2to1 port map(
        A_MUX,
        data_bus_s,
        LZE_out_A_Mux,
        A_Mux_out
    );

    Reg_A:   register32 port map(
        A_Mux_out,
```

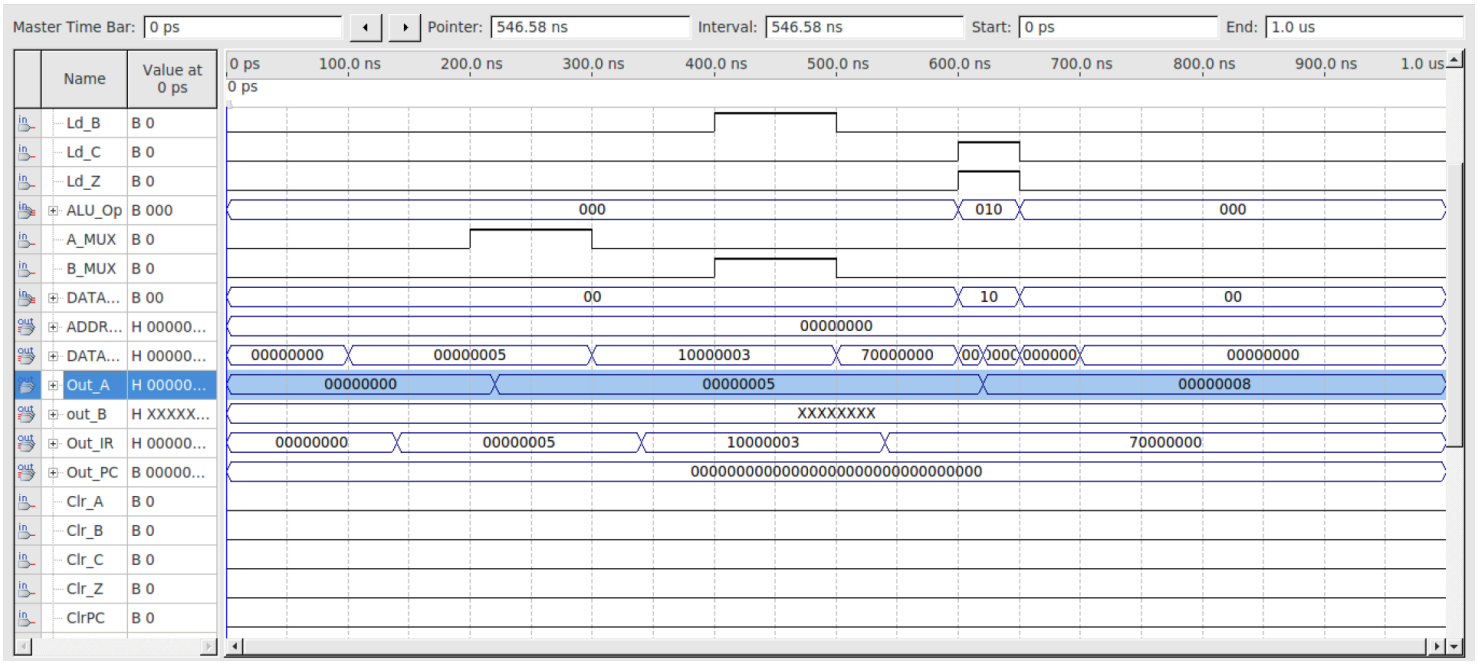
```
    Reg_A:   register32 port map(
        A_Mux_out,
        Ld_A,
        Clr_A,
        Clk,
        reg_A_out
    );

    LZE_B_Mux:  LZE port map(
        IR_OUT,
        LZE_out_B_Mux
    );

    B_Mux0:  mux2to1 port map(
        B_MUX,
        data_bus_s,
        LZE_out_B_Mux,
        B_mux_out
    );

    Reg_B:   register32 port map(
        B_mux_out,
        Ld_B,
        Clr_B,
        Clk,
        reg_B_out
    );

    Reg_Mux0:   mux2to1 port map(
        REG_MUX,
        Reg_A_out,
        Reg_B_out,
        Reg_Mux_out
    );

    RED_Data_Mem:  RED port map(
```

```
241         IR_OUT,
242         RED_out_Data_Mem
243     );
244 
245     Data_mem0:  data_mem port map(
246         mClk,
247         RED_out_Data_Mem,
248         Reg_Mux_out,
249         WEN,
250         EN,
251         data_mem_out
252     );
253 
254     UZE_IM_MUX1:   UZE port map(
255         IR_OUT,
256         UZE_IM_MUX1_out
257     );
258 
259     IM_MUX1a:   mux2to1 port map(
260         IM_MUX1,
261         reg_A_out,
262         UZE_IM_MUX1_out,
263         IM_MUX1_out
264     );
265 
266     LZE_IM_MUX2:   LZE port map(
267         IR_OUT,
268         LZE_IM_MUX2_out
269     );
270 
271     IM_MUX2a:   mux4to1 port map(
272         IM_MUX2,
273         reg_B_out,
274         LZE_IM_MUX2_out,
275         (temp & '1'),
276         (others => '0'),
```

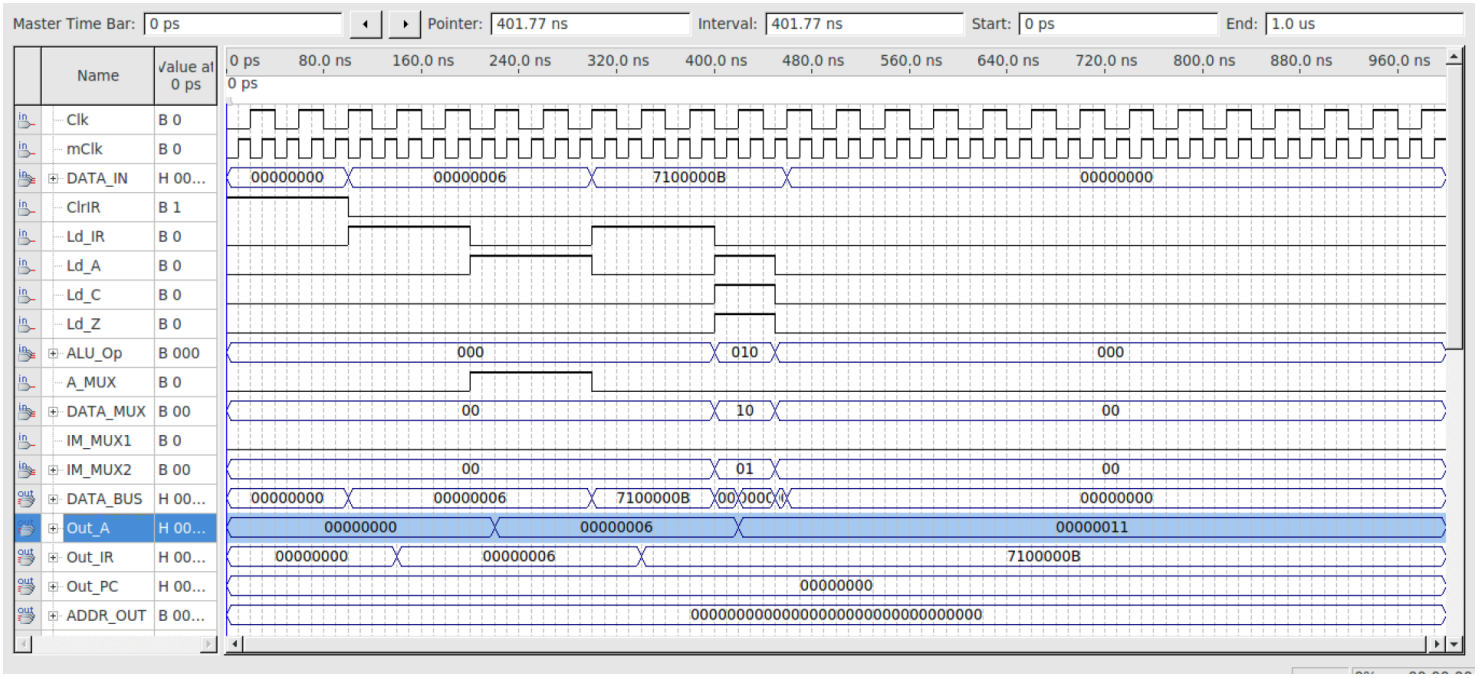
```
276         (others => '0'),
277         IM_MUX2_out
278     );
279 
280     ALU0: ALU port map(
281         IM_MUX1_out,
282         IM_MUX2_out,
283         ALU_Op,
284         ALU_out,
285         zero_flag ,
286         carry_flag
287     );
288 
289     DATA_MUX0:  mux4to1 port map(
290         DATA_MUX,
291         DATA_IN,
292         data_mem_out,
293         ALU_out,
294         (others => '0'),
295         data_bus_s
296     );
297 
298     DATA_BUS <= data_bus_s;
299     OUT_A <= reg_A_out;
300     OUT_B <= reg_B_out;
301     OUT_IR <= IR_OUT;
302     ADDR_OUT <= out_pc_sig;
303     OUT_PC <= out_pc_sig;
304 
305     MEM_ADDR <= RED_out_Data_Mem;
306     MEM_IN <= Reg_Mux_out;
307     MEM_OUT <= data_mem_out;
308 
309     OUT_C <= carry_flag ;
310     OUT_Z <= zero_flag ;
311 end Behavior;
```

Results: The waveform below displays the function of the Data path VHDL code. The waveform includes the clock, the data address, the data input, the enable signal input, the write enable signal and the data output.

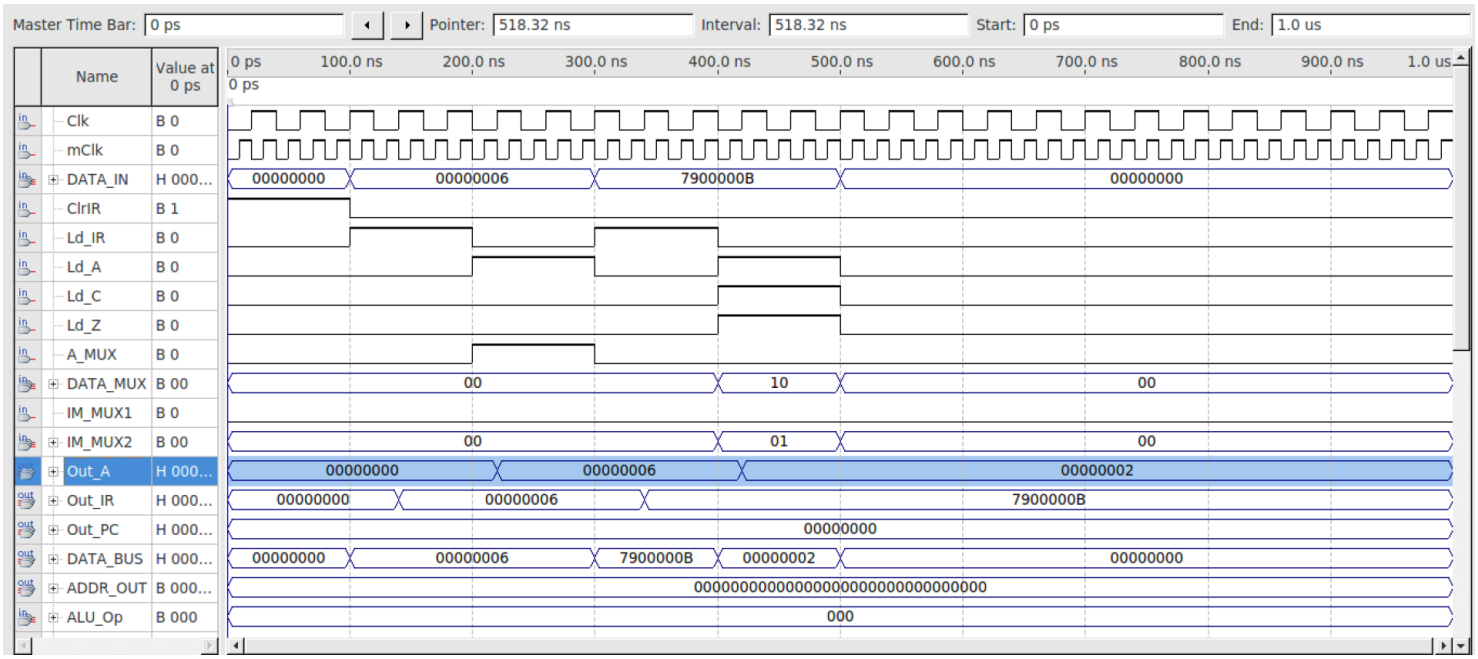
ADD:



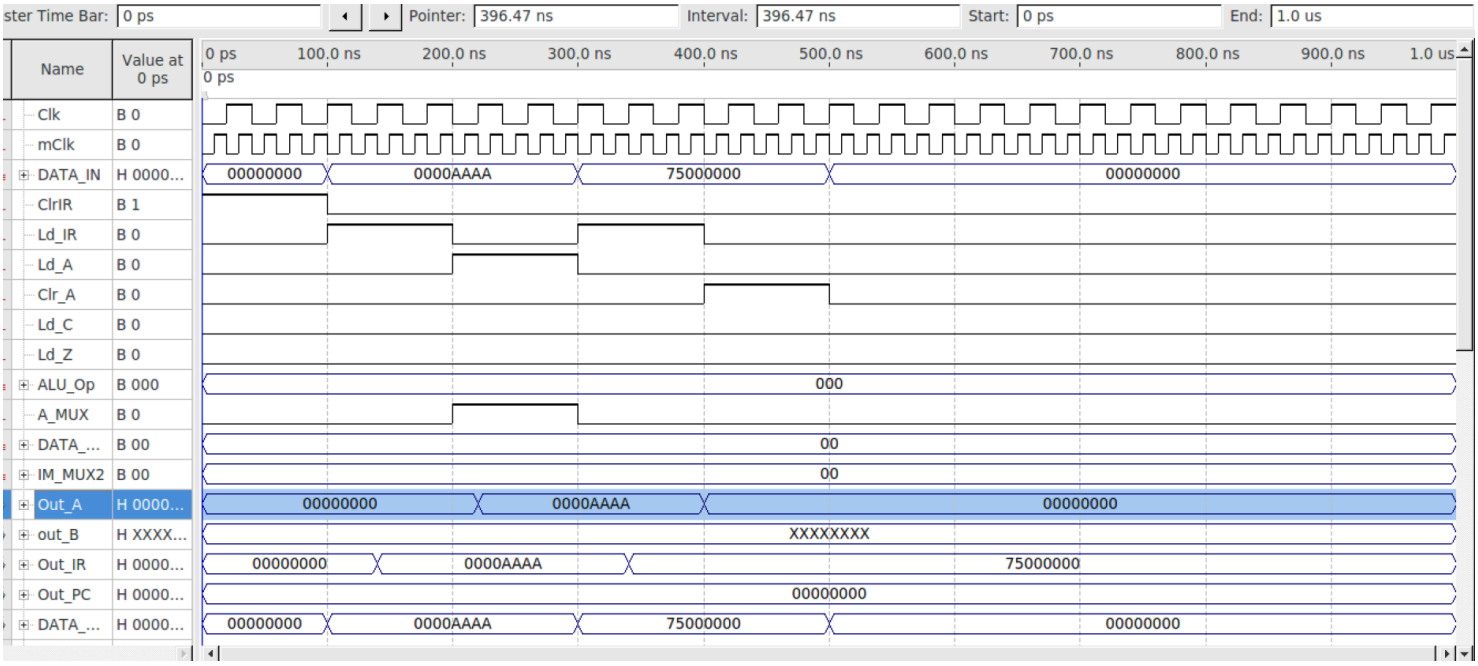
ADDI:



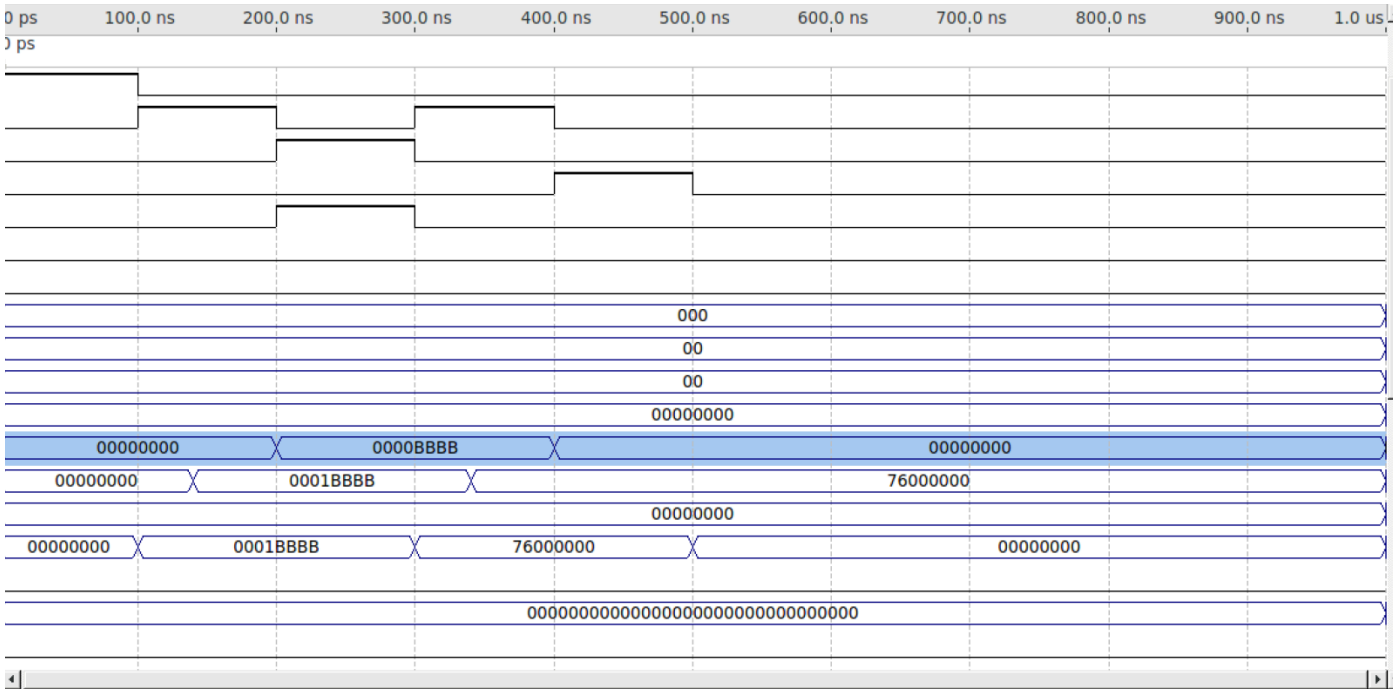
ANDI:


Master Time Bar: 0 ps
Pointer: 518.32 ns
Interval: 518.32 ns
Start: 0 ps
End: 1.0 us
Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk B 0
mClk B 0
DATA_IN H 000...
00000000
00000006
7900000B
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_C B 0
Ld_Z B 0
A_MUX B 0
DATA_MUX B 00
00
10
00
IM_MUX1 B 0
IM_MUX2 B 00
00
01
00
Out_A H 000...
00000000
00000006
00000002
Out_IR H 000...
00000000
00000006
7900000B
Out_PC H 000...
00000000
DATA_BUS H 000...
00000000
00000006
7900000B
00000002
00000000
ADDR_OUT B 000...
00000000000000000000000000000000
ALU_Op B 000
000

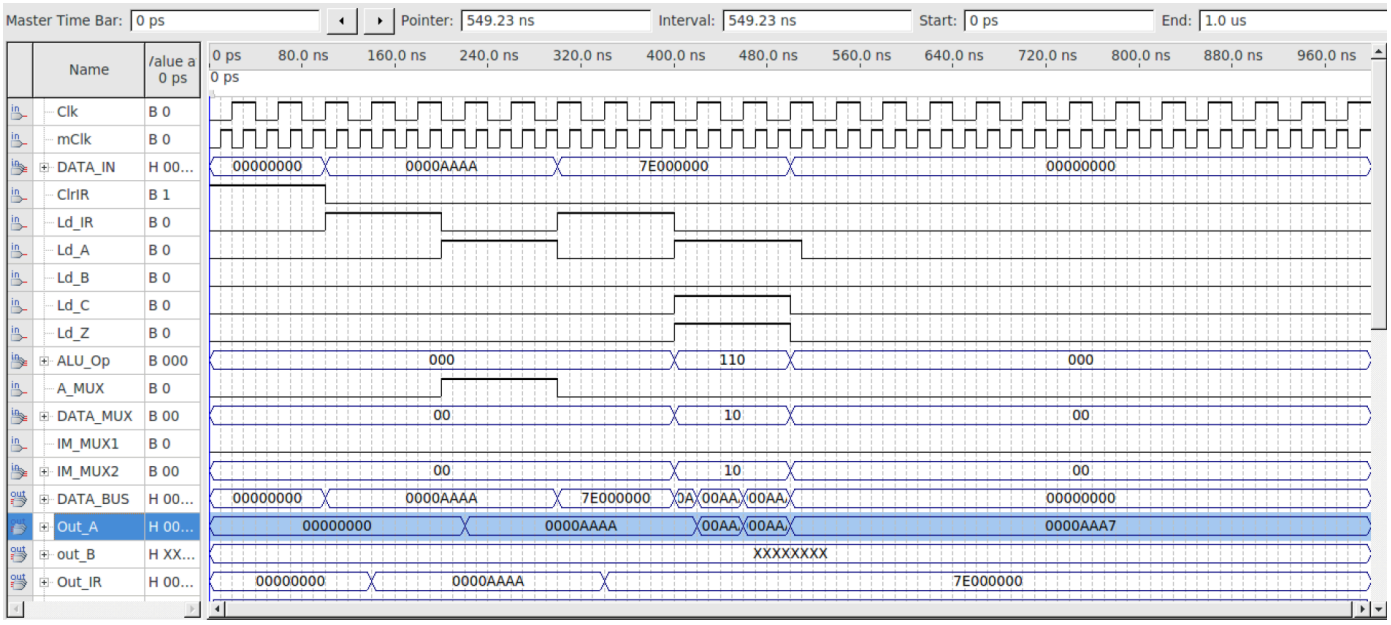
CLRA:


ster Time Bar: 0 ps
Pointer: 396.47 ns
Interval: 396.47 ns
Start: 0 ps
End: 1.0 us
Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk B 0
mClk B 0
DATA_IN H 0000...
00000000
0000AAAA
75000000
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Clr_A B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
000
A_MUX B 0
DATA_... B 00
00
IM_MUX2 B 00
00
Out_A H 0000...
00000000
0000AAAA
00000000
out_B H XXXX...
XXXXXXXX
Out_IR H 0000...
00000000
0000AAAA
75000000
Out_PC H 0000...
00000000
DATA_... H 0000...
00000000
0000AAAA
75000000
00000000

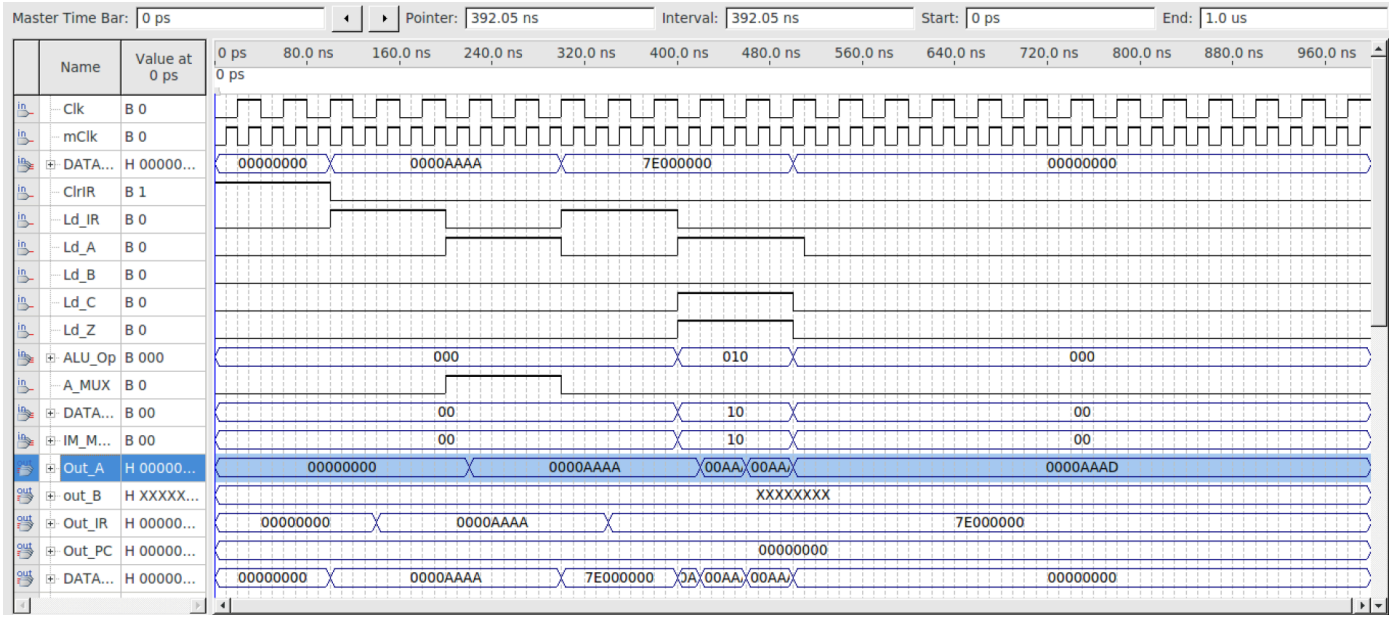
CLRB:


0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
000
00
00
00000000
00000000
0000BBBB
00000000
00000000
0001BBBB
76000000
00000000
00000000
0001BBBB
76000000
00000000
00000000000000000000000000000000

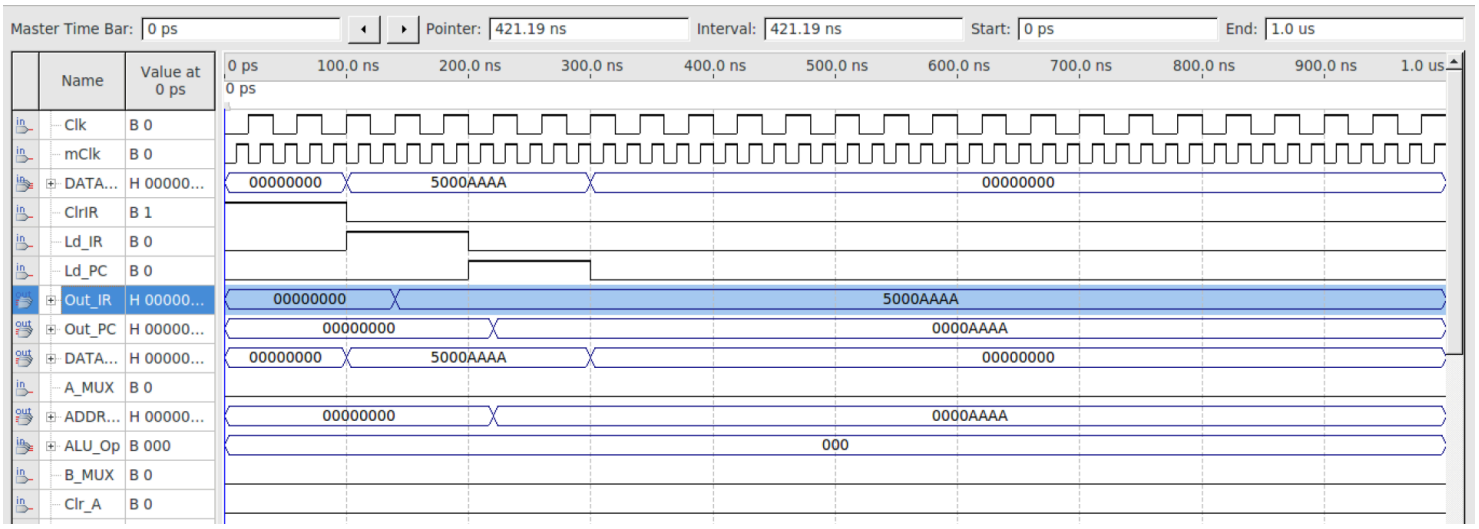
DECA:


Master Time Bar: 0 ps
Pointer: 549.23 ns
Interval: 549.23 ns
Start: 0 ps
End: 1.0 us
Name
/alue a 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_A
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
DATA_MUX
IM_MUX1
IM_MUX2
DATA_BUS
Out_A
out_B
Out_IR
00000000
0000AAAA
7E000000
00000000
000
110
000
00
10
00
00
10
00
00000000
0000AAAA
7E000000
00AA
00AA
00000000
00000000
0000AAAA
00AA
00AA
0000AAA7
XXXXXXXX
00000000
0000AAAA
7E000000

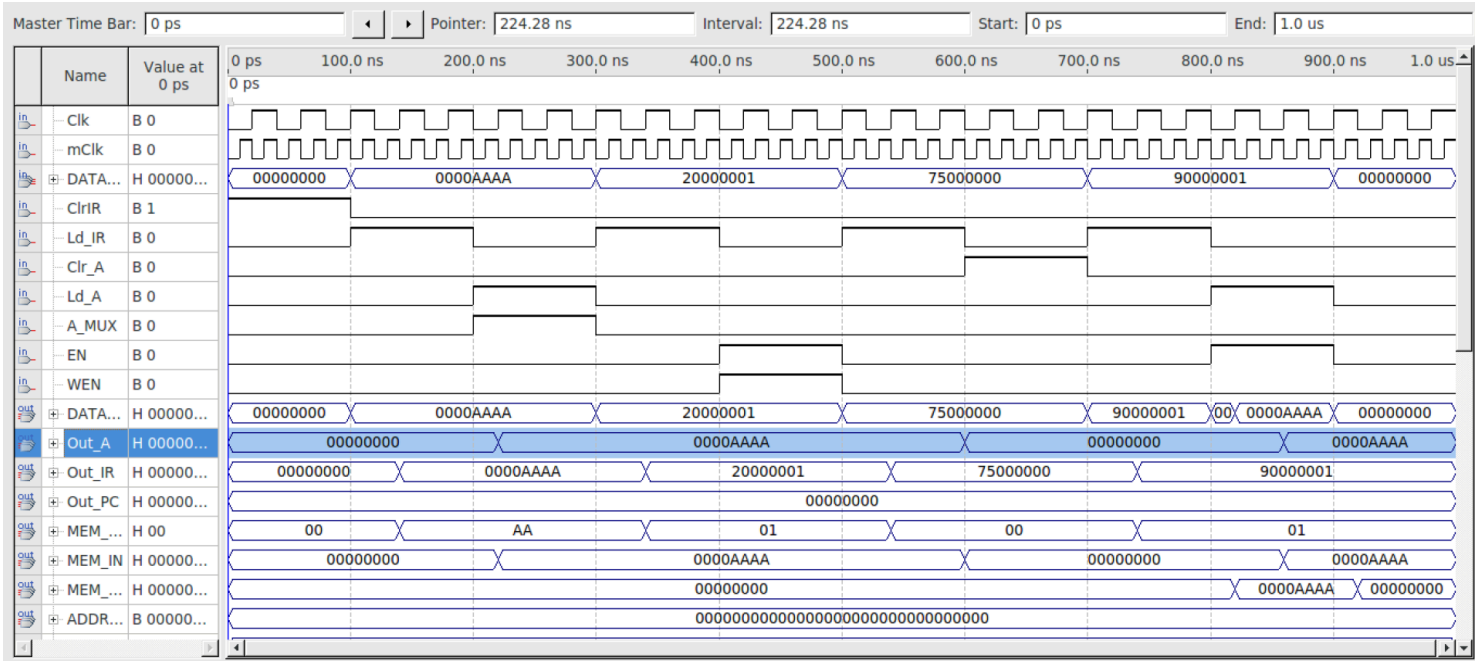
INCA:


Master Time Bar: 0 ps
Pointer: 392.05 ns
Interval: 392.05 ns
Start: 0 ps
End: 1.0 us
Name
Value at 0 ps
Clk B 0
mClk B 0
DATA... H 00000...
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_B B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
A_MUX B 0
DATA... B 00
IM_M... B 00
Out_A H 00000...
out_B H XXXXX...
Out_IR H 00000...
Out_PC H 00000...
DATA... H 00000...
00000000
0000AAAA
7E000000
00000000
000
010
000
00
10
00
00AA
0000AAAD
XXXXXXXX

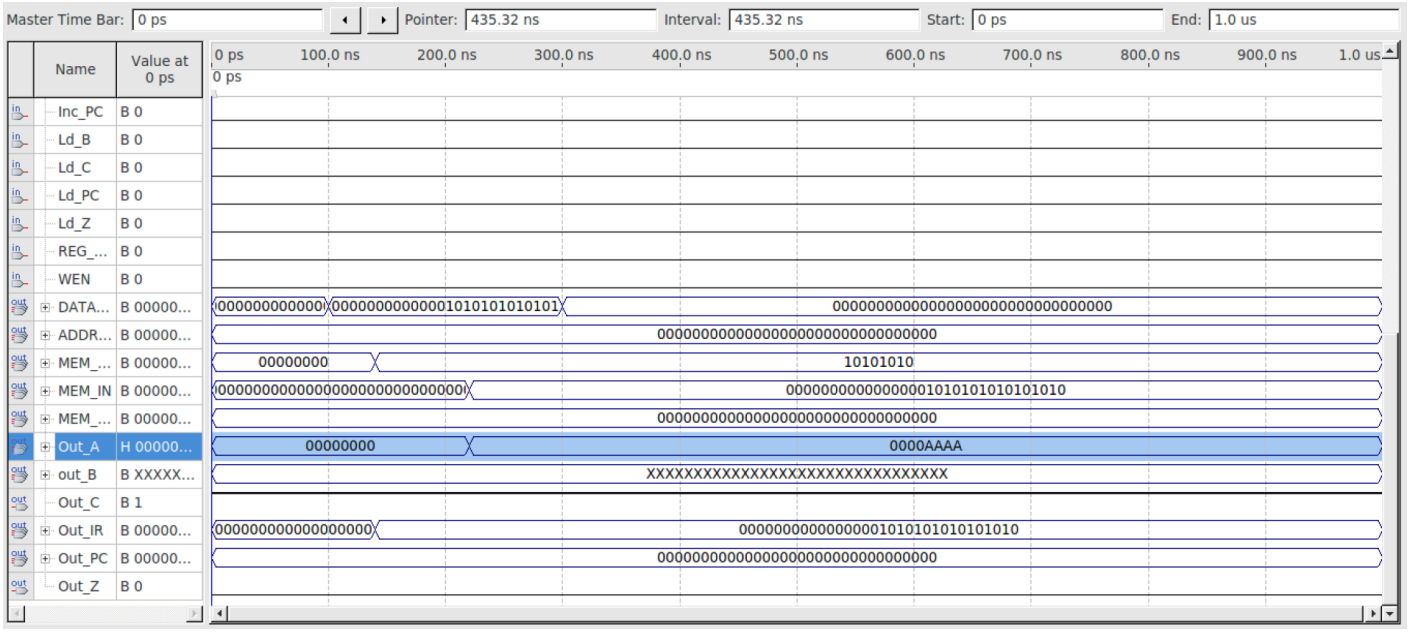
JMP:


Master Time Bar: 0 ps
Pointer: 421.19 ns
Interval: 421.19 ns
Start: 0 ps
End: 1.0 us
Name
Value at 0 ps
Clk B 0
mClk B 0
DATA... H 00000...
ClrIR B 1
Ld_IR B 0
Ld_PC B 0
Out_IR H 00000...
Out_PC H 00000...
DATA... H 00000...
A_MUX B 0
ADDR... H 00000...
ALU_Op B 000
B_MUX B 0
Clr_A B 0
00000000
5000AAAA
0000AAAA
000

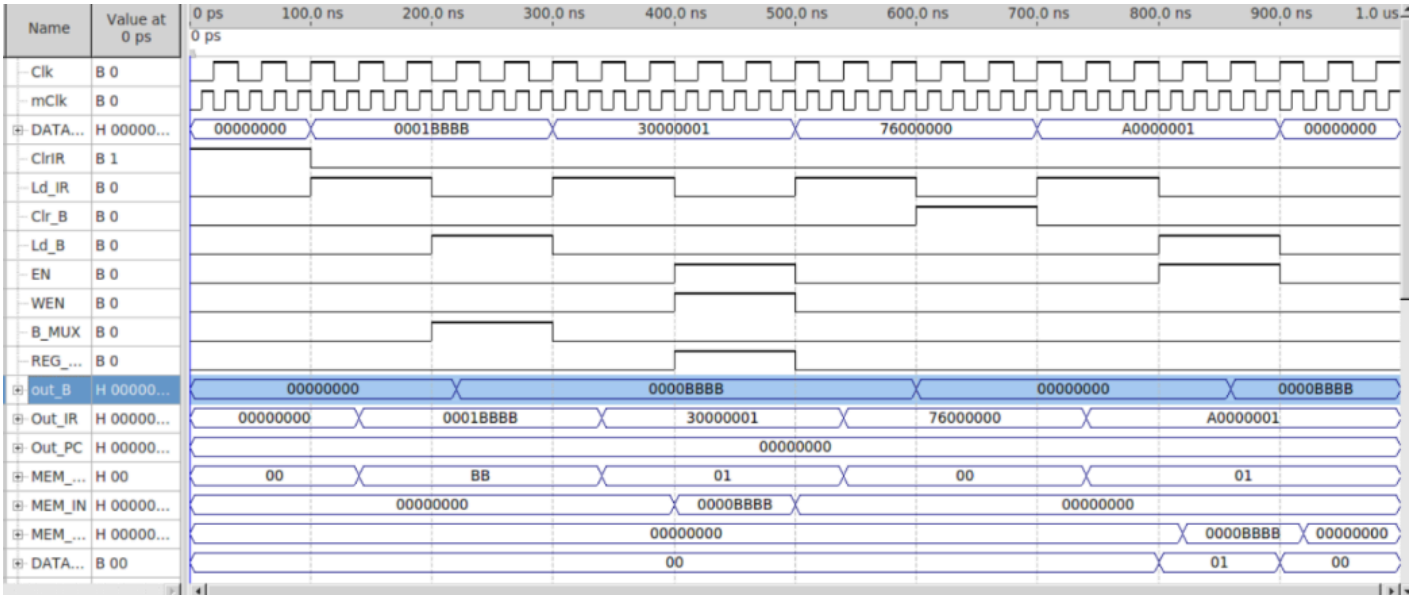
LDA:


Master Time Bar: 0 ps
Pointer: 224.28 ns
Interval: 224.28 ns
Start: 0 ps
End: 1.0 us
Name
Value at 0 ps
Clk B 0
mClk B 0
DATA... H 00000...
ClrIR B 1
Ld_IR B 0
Clr_A B 0
Ld_A B 0
A_MUX B 0
EN B 0
WEN B 0
DATA... H 00000...
Out_A H 00000...
Out_IR H 00000...
Out_PC H 00000...
MEM_... H 00
MEM_IN H 00000...
MEM_... H 00000...
ADDR... B 00000...
00000000
0000AAAA
20000001
75000000
90000001
00
00000000
AA
01

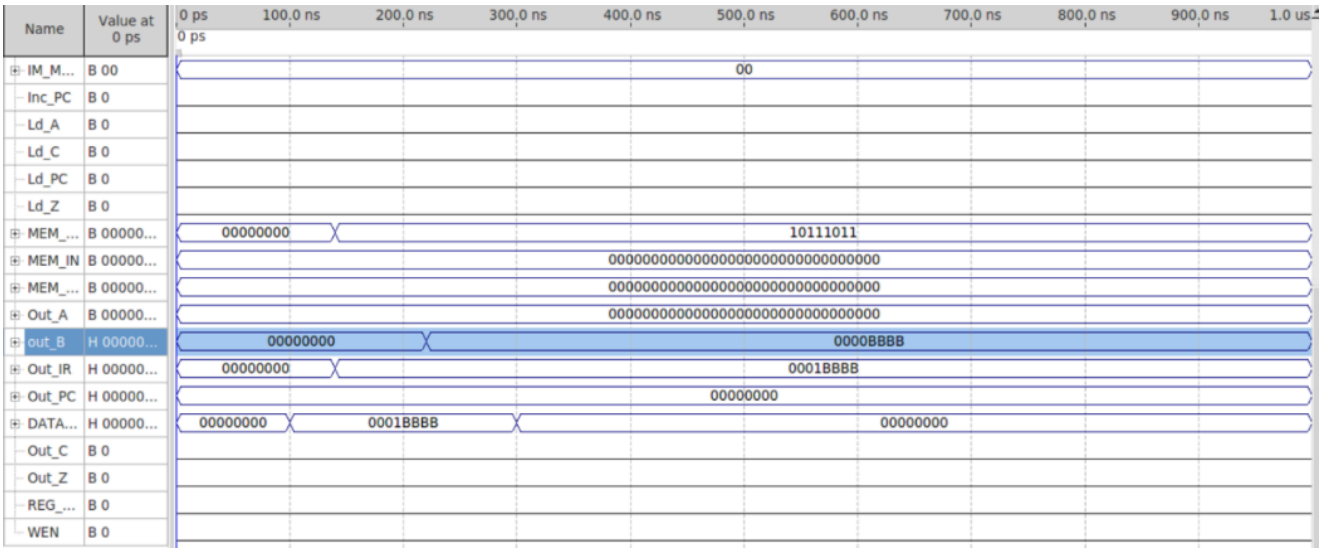
LDAI:



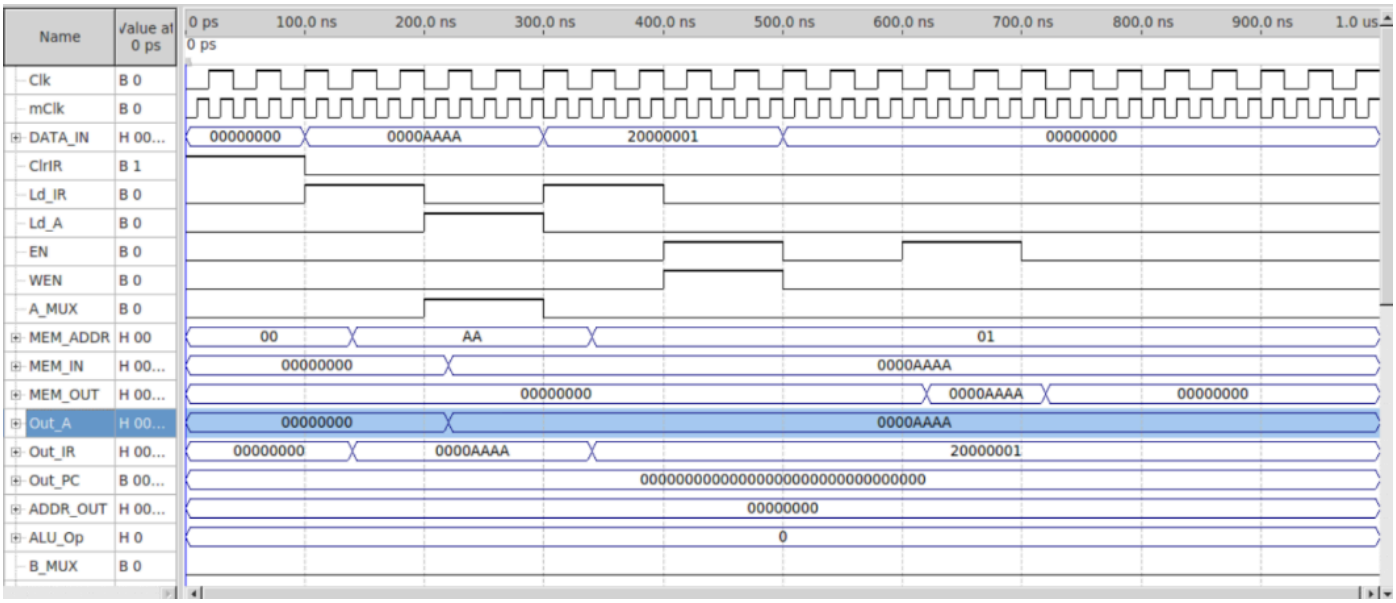
LDB:



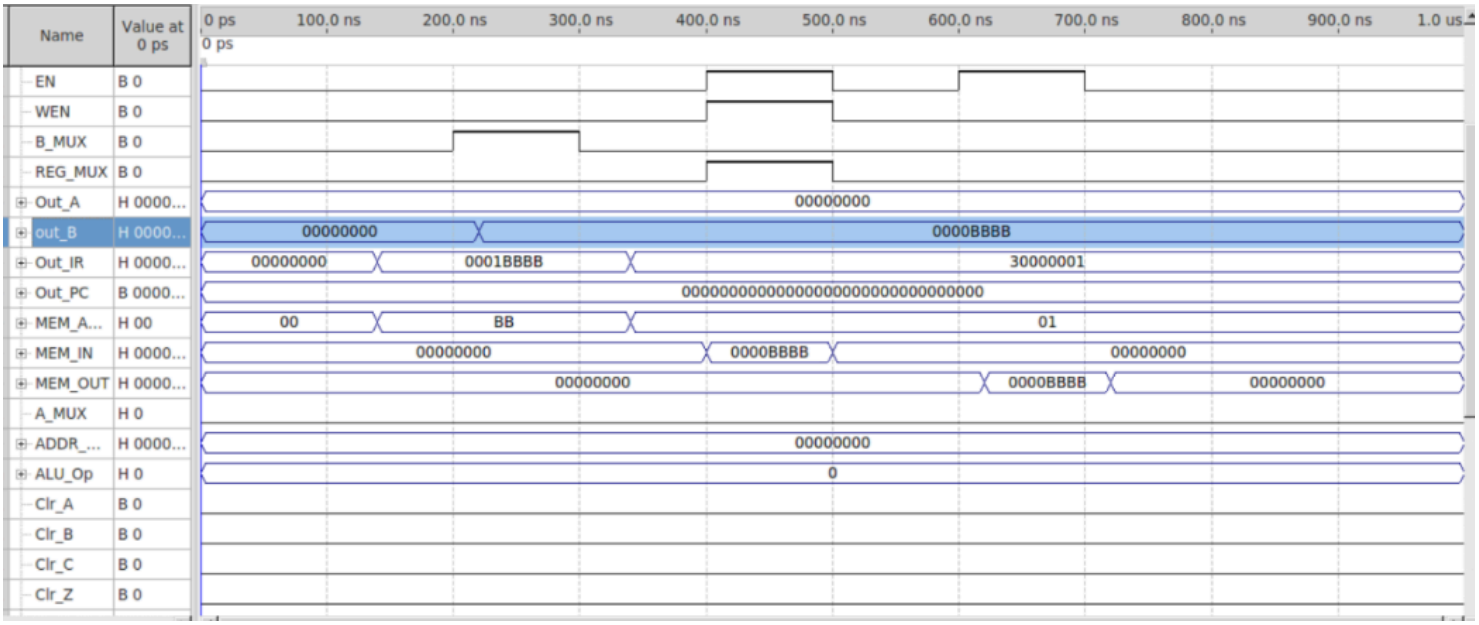
LDBI:



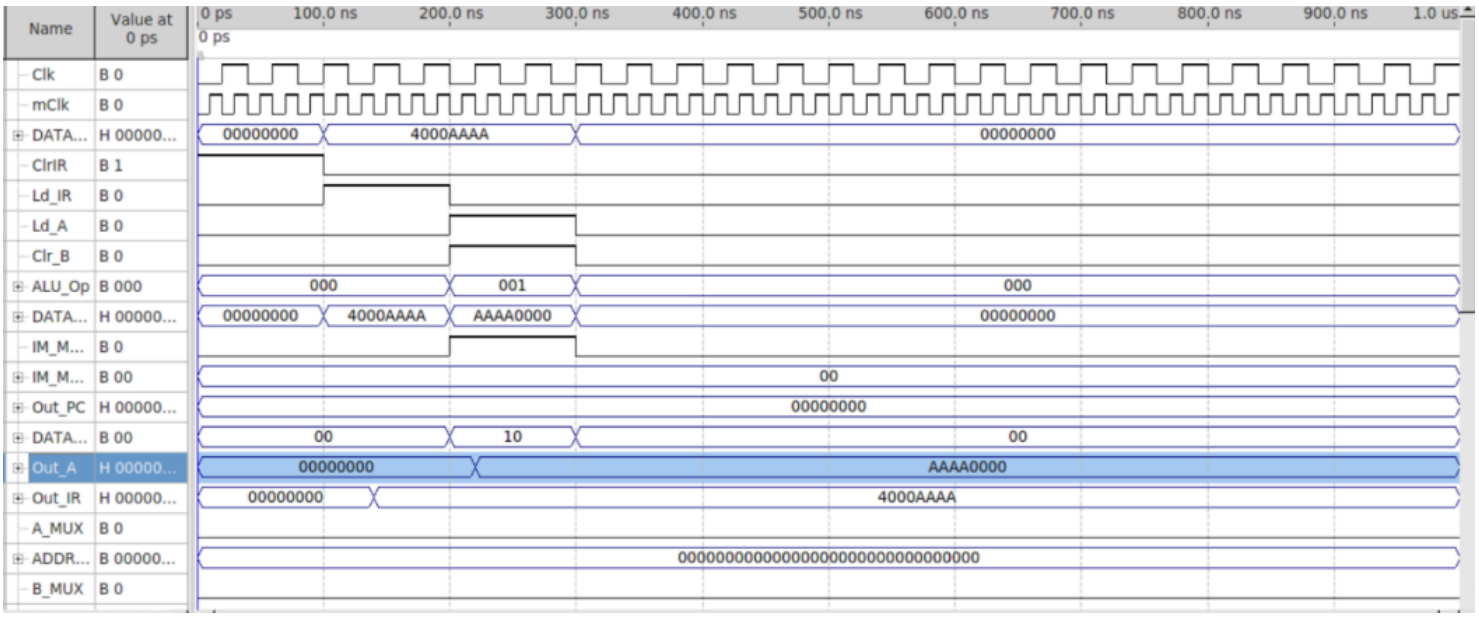
STA:


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_A
EN
WEN
A_MUX
MEM_ADDR
MEM_IN
MEM_OUT
Out_A
Out_IR
Out_PC
ADDR_OUT
ALU_Op
B_MUX
00000000
0000AAAA
20000001
00000000
00
AA
01
00000000
0000AAAA
00000000
0000AAAA
00000000
00000000
0000AAAA
00000000
0000AAAA
20000001
00000000000000000000000000000000
00000000
0

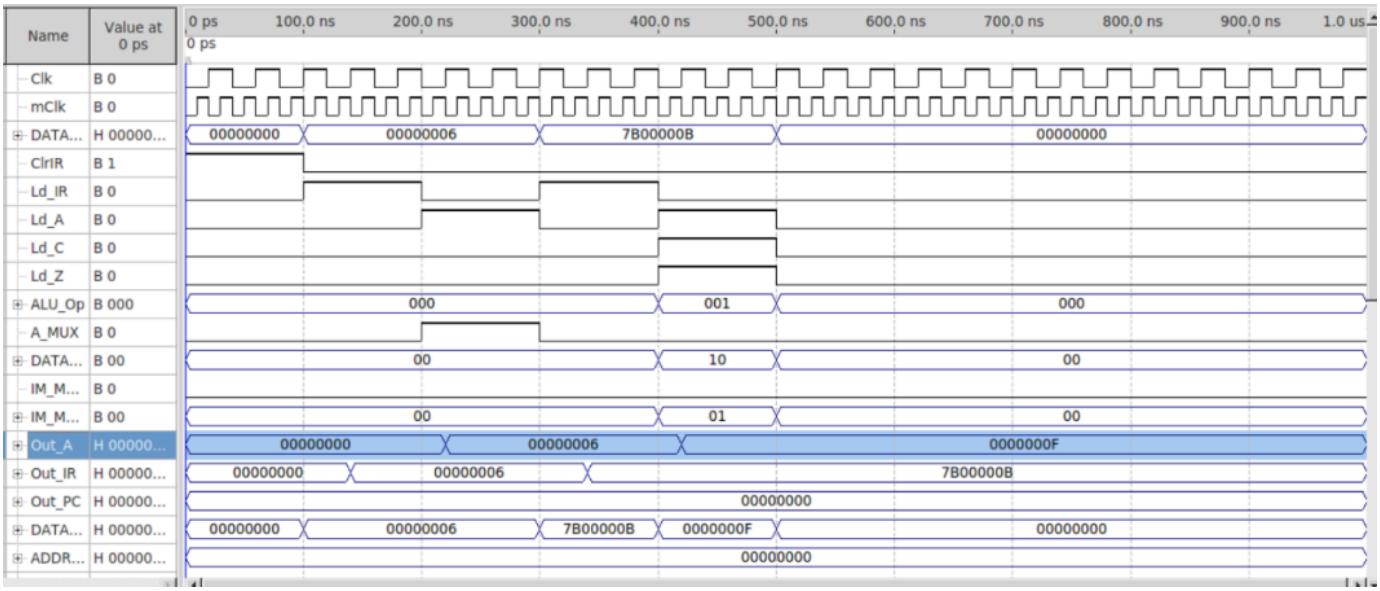
STB:


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
EN
WEN
B_MUX
REG_MUX
Out_A
out_B
Out_IR
Out_PC
MEM_A...
MEM_IN
MEM_OUT
A_MUX
ADDR_...
ALU_Op
Clr_A
Clr_B
Clr_C
Clr_Z
00000000
00000000
0000BBBB
00000000
0001BBBB
30000001
00000000000000000000000000000000
00
BB
01
00000000
0000BBBB
00000000
00000000
0000BBBB
00000000
00000000
0

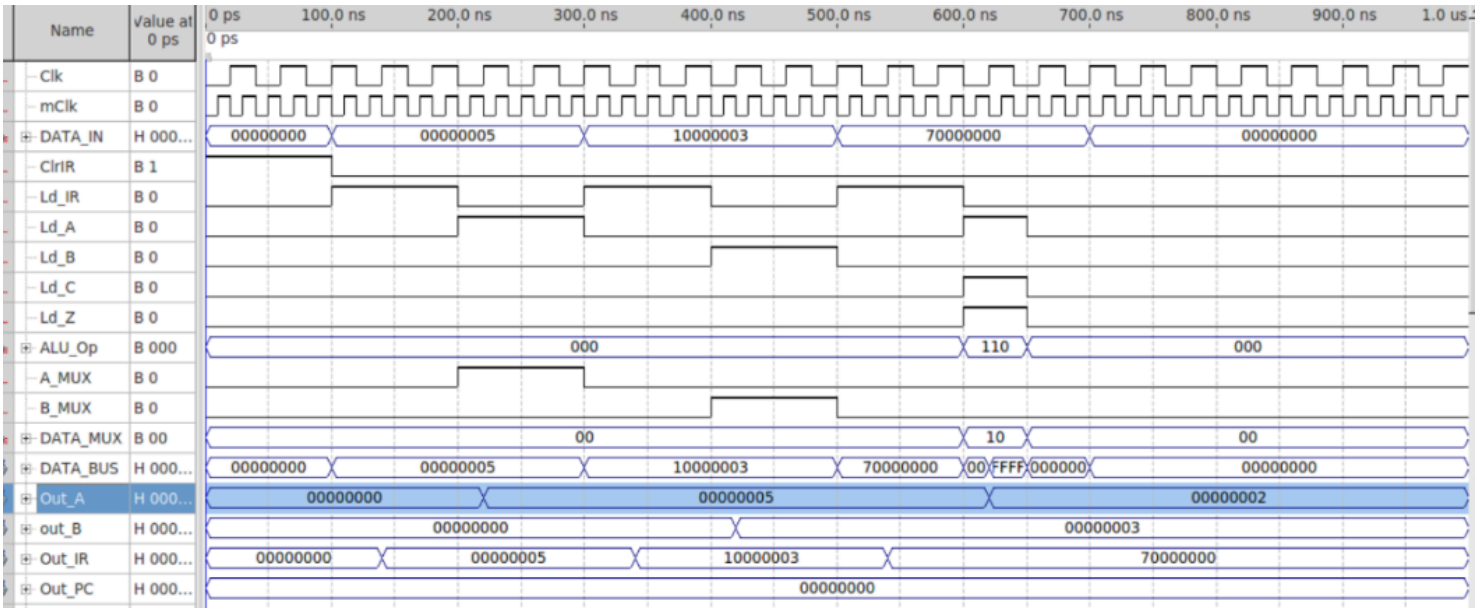
LUI:


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Ld_A
Clr_B
ALU_Op
DATA...
IM_M...
IM_M...
Out_PC
DATA...
Out_A
Out_IR
A_MUX
ADDR...
B_MUX
00000000
4000AAAA
00000000
000
001
000
00000000
4000AAAA
AAAA0000
00000000
00
00000000
00
10
00
00000000
AAAA0000
00000000
4000AAAA
00000000000000000000000000000000

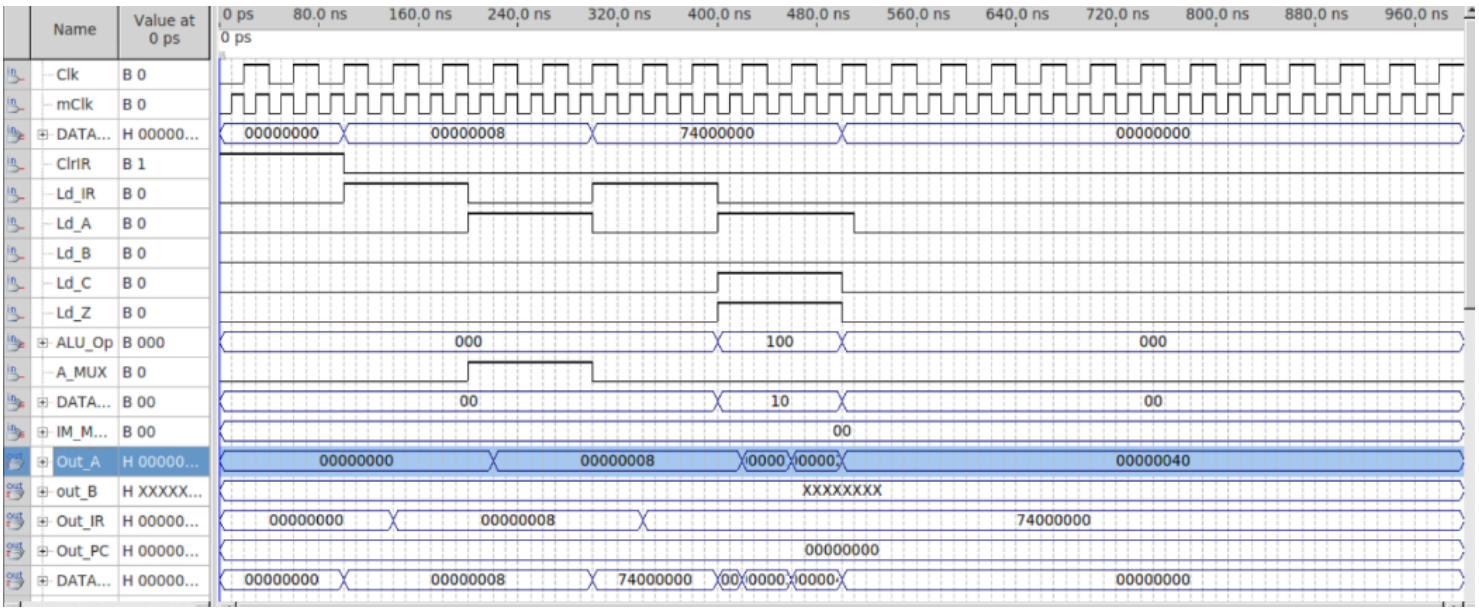
ORI:


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Ld_A
Ld_C
Ld_Z
ALU_Op
A_MUX
DATA...
IM_M...
IM_M...
Out_A
Out_IR
Out_PC
DATA...
ADDR...
00000000
00000006
7B00000B
00000000
0000000F
000
001
10
01

SUB:


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_A
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
B_MUX
DATA_MUX
DATA_BUS
Out_A
out_B
Out_IR
Out_PC
00000000
00000005
10000003
70000000
00000002
00000003
000
110
10
FFFF0000

ROL:


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Clk
mClk
DATA...
ClrIR
Ld_IR
Ld_A
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
DATA...
IM_M...
Out_A
out_B
Out_IR
Out_PC
DATA...
00000000
00000008
74000000
00000040
XXXXXXXX
000
100
00
10

ROR:

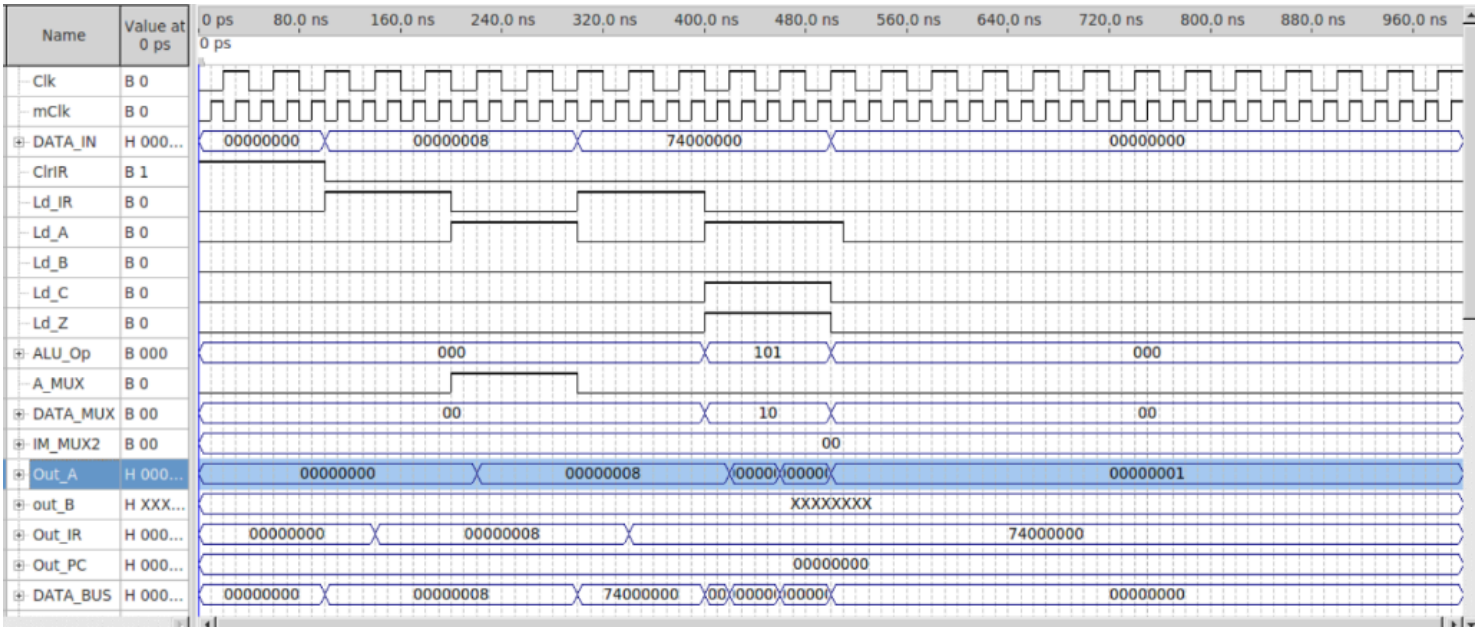


Chart:

INST	CLR_IR// LD_IR	LD_PC// INC_PC	CLR_A// LD_A	CLR_B// LD_B	CLR_C// LD_C	CLR_Z// LD_Z	ALU OP	EN// WEN	A/B // MUX	REG MUX	DATA MUX	IM_MUX1 // IM_MUX2
LDA	0//0	0//0	0//1	0//0	0//0	0//0	xxx	1//0	0//X	X	01	X
LDB	0//0	0//0	0//0	0//0	0//0	0//0	xxx	1//0	X//0	X	01	X
STA	0//0	0//0	0//0	0//0	0//0	0//0	xxx	1//1	X	0	X	X
STB	0//0	0//0	0//0	0//0	0//0	0//0	xxx	1//1	X	1	X	X
LDAI	0//0	0//0	0//1	0//0	0//0	0//0	xxx	X	1//X	X	X	X
LDBI	0//0	0//0	0//0	0//1	0//0	0//0	xxx	X	X//1	X	X	X
LUI	0//0	0//0	0//1	1//0	0//0	0//0	001	X	0//X	X	10	1//X
ANDI	0//0	0//0	0//1	0//0	0//1	0//1	000	X	0//X	X	10	0//01
DECA	0//0	0//0	0//1	0//0	0//1	0//1	110	X	0//X	X	10	0//10
ADD	0//0	0//0	0//1	0//0	0//1	0//1	010	X	0//X	X	10	0//00
SUB	0//0	0//0	0//1	0//0	0//1	0//1	110	X	0//X	X	10	0//00
AND	0//0	0//0	0//1	0//0	0//1	0//1	000	X	0//X	X	10	0//00
INCA	0//0	0//0	0//1	0//0	0//1	0//1	010	X	0//X	X	10	0//10
JMP	0//0	0//1	0//0	0//0	0//0	0//0	xxx	X	X	X	X	X
ADDI	0//0	0//0	0//1	0//0	0//1	0//1	010	X	0//X	X	10	0//01
ORI	0//0	0//0	0//1	0//0	0//1	0//1	001	X	0//X	X	10	0//01
ROL	0//0	0//0	0//1	0//0	0//1	0//1	100	X	0//X	X	10	0//X
ROR	0//0	0//0	0//1	0//0	0//1	0//1	101	X	0//X	X	10	0//X

CLRA	0//0	0//0	1//0	0//0	0//0	0//0	XXX	X	X	X	X	X
CLRZ	0//0	0//0	0//0	0//0	0//0	1//0	XXX	X	X	X	X	X
CLRB	0//0	0//0	0//0	1//0	0//0	0//0	XXX	X	X	X	X	X
CLRC	0//0	0//0	0//0	0//0	1//0	0//0	XXX	X	X	X	X	X
PC <= PC+4	0//0	1//1	0//0	0//0	0//0	0//0	XXX	X	X	X	X	X
IR<= M[INST]	0//1	0//0	0//0	0//0	0//0	0//0	XXX	X	X	X	00	X
IR[15,0]	0//0	1//0	0//0	0//0	0//0	0//0	XXX	X	X	X	X	X

Discussion:

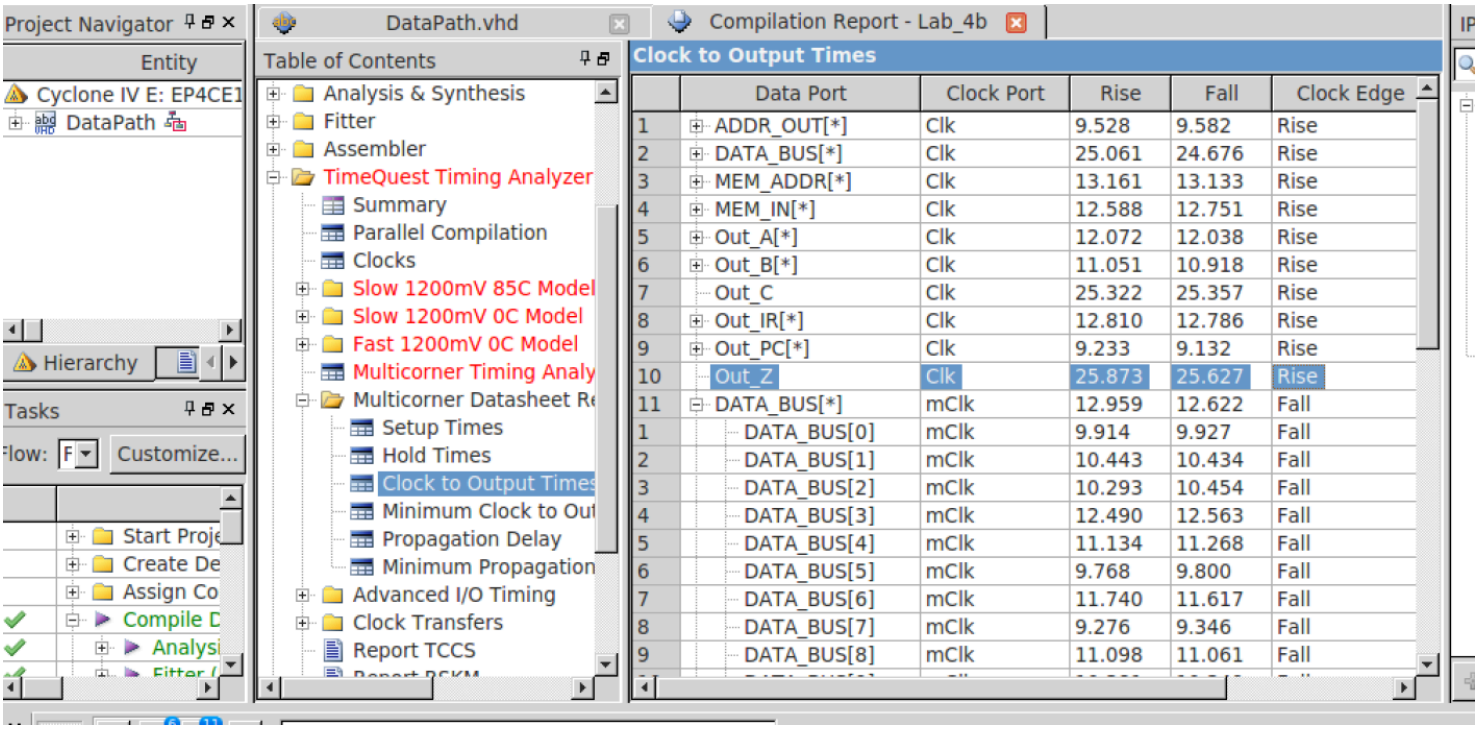
1. **How does this data path implement the INCA, ADDI, LDBI and LDA operations?**

The data path is made up of several different operations, and each operation has its own specific requirements. The INCA function increments a register's contents by using the register and adder files. The ADDI function adds an immediate value to a register using an adder and registers file and a separate register to store the immediate value. The LDBI function loads the immediate value into a register, and a separate register is used to store the immediate value. Finally, the LDA function loads the value from the set memory address into the register. Overall, the data path contains several files and functions, which work together to function like a Central Processing Unit.

2. **The data path has a maximum reliable operating speed (Clk). What determines this speed? (i.e. how would you estimate the data-path circuit clock)?**

There are several factors that can affect the maximum reliable operating speed of a data path in a digital circuit. Propagating delay is the time it takes for a signal to travel through a logic circuit from input to output. Every gate, such as the AND, OR, and NOT, has a specific delay associated with it. The total propagation delay of the circuit is the longest path that a signal will take in the logic. A crucial step in estimating the data path circuit clock is to identify the critical path and determine the total delay along the logic path. The frequency of the clock must be set up in a way that will allow enough time for the propagation delay to be completed before the beginning of the next clock cycle.

3. **What is a reliable limit for your data-path clock?**



Data path clock calculations:
(25.873ns + 2.627ns) / 2 = **25.75ns**
Frequency = 1/25.75 = **0.03883 ns**
Total Time = 0.03883 x 1000 = **38.883 ns**

The reliable limit of the data-path clock can be determined by finding the average of the most time consuming rising edge and falling edge. However, there are factors that may limit and affect the maximum clock frequency. The data path was calculated using the “Out_Z,” which was determined to be 25.75ns. The frequency is 0.03883 ns, and the total time is 38.883 ns.