Toronto Metropolitan University

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

Course Title:	Computer Organization and Architecture
Course Number:	COE 608
Semester/Year (e.g. F2017)	W2023

Instructor	Demetres Kostas

Assignment/Lab Number:	6
Assignment/Lab Title:	The Complete CPU (Overall Project)

Submission Date:	April 12, 2023
Due Date:	April 12, 2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Hernandez	Spencer	501033228	11	Spencer H
Dumlao	Carlo	501018239	11	C.D

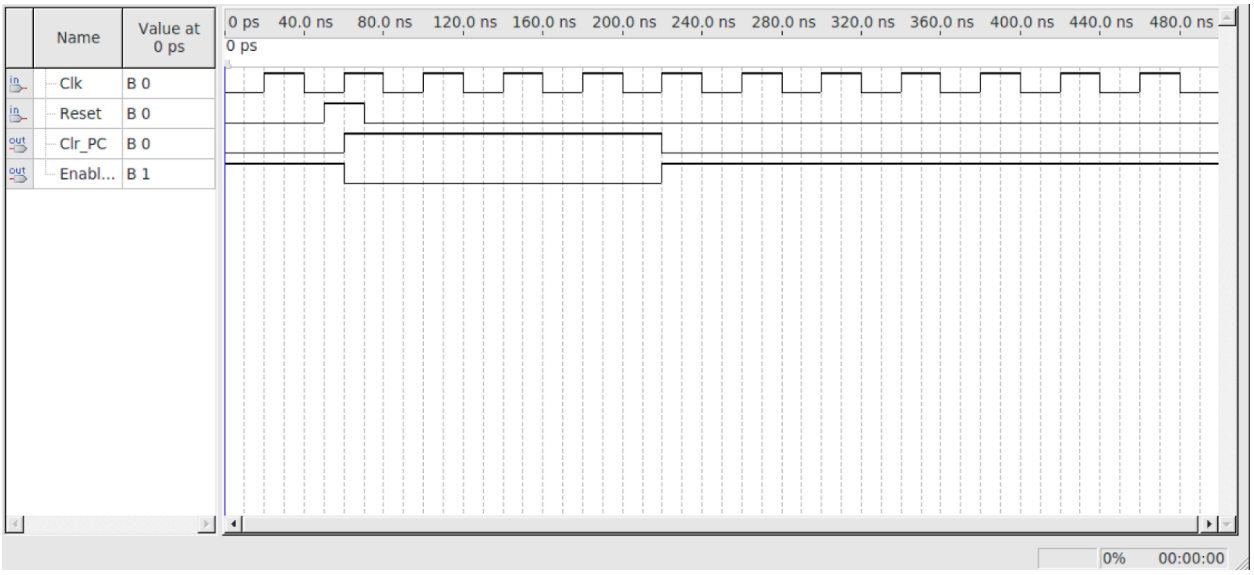
*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the 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.pdf

1.0 CPU Reset Circuitry

1.1 VHDL

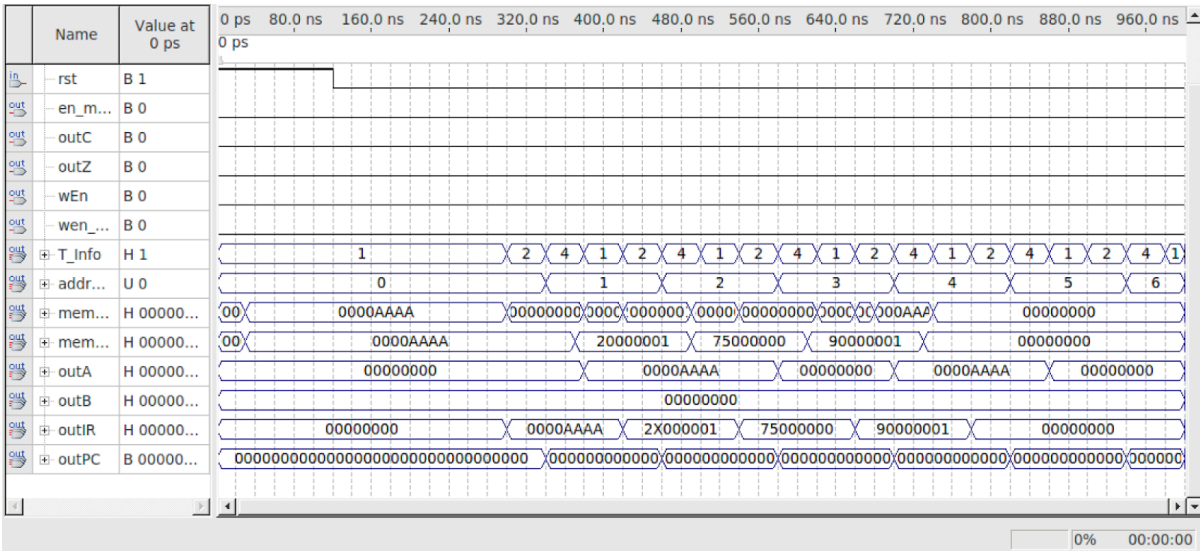
```
1. LIBRARY ieee;
2. USE ieee.std_logic_1164.ALL;
3. USE ieee.std_logic_arith.ALL;
4. USE ieee.std_logic_unsigned.ALL;
5.
6. ENTITY reset_circuit IS
7.       PORT(
8.                   Reset : IN STD_LOGIC;
9.                   Clk : IN STD_LOGIC;
10.                  Enable_PD : OUT STD_LOGIC := '1';
11.                  Clr_PC : OUT STD_LOGIC
12.      );
13.END reset_circuit;
14.
15.ARCHITECTURE Behavior OF reset_circuit IS
16.      TYPE clkNum IS (clk0, clk1, clk2, clk3);
17.      SIGNAL present_clk: clkNum;
18.BEGIN
19.      process(Clk) begin
20.            if rising_edge(Clk) then
21.                  if Reset = '1' then
22.                        Clr_PC <= '1';
23.                        Enable_PD <= '0';
24.                        present_clk <= clk0;
25.                  elsif present_clk <= clk0 then
26.                        present_clk <= clk1;
27.                  elsif present_clk <= clk1 then
28.                        present_clk <= clk2;
29.                  elsif present_clk <= clk2 then
30.                        present_clk <= clk3;
31.                  elsif present_clk <= clk3 then
32.                        Clr_PC <= '0';
33.                        Enable_PD <= '1';
34.                  end if;
35.            end if;
36.      end process;
37.END Behavior;
```

1.2 Timing Simulation

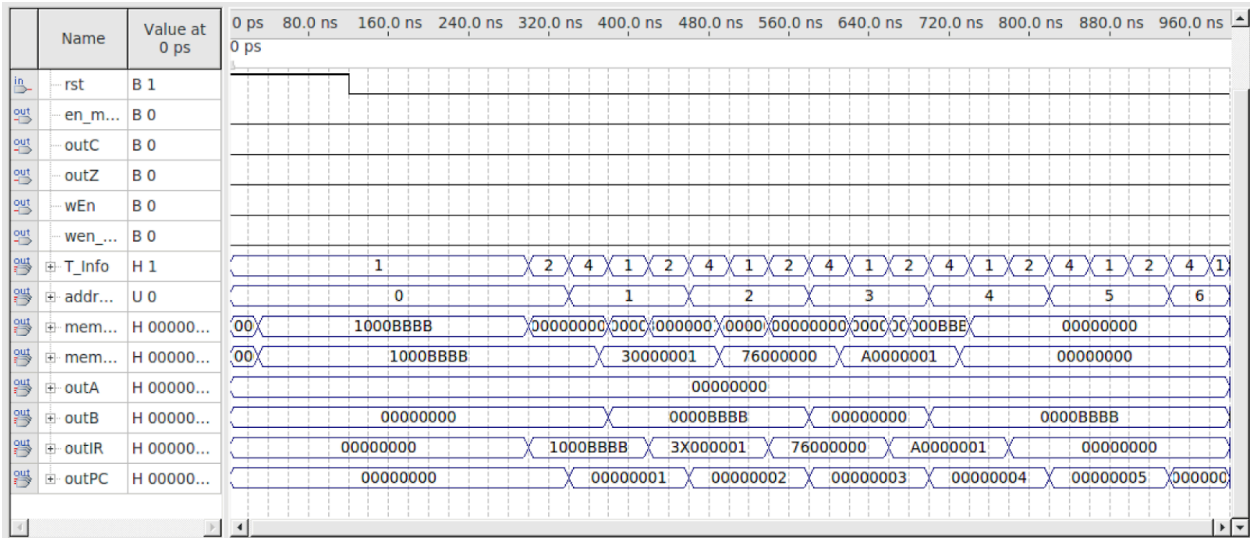


2.0 The Complete CPU System

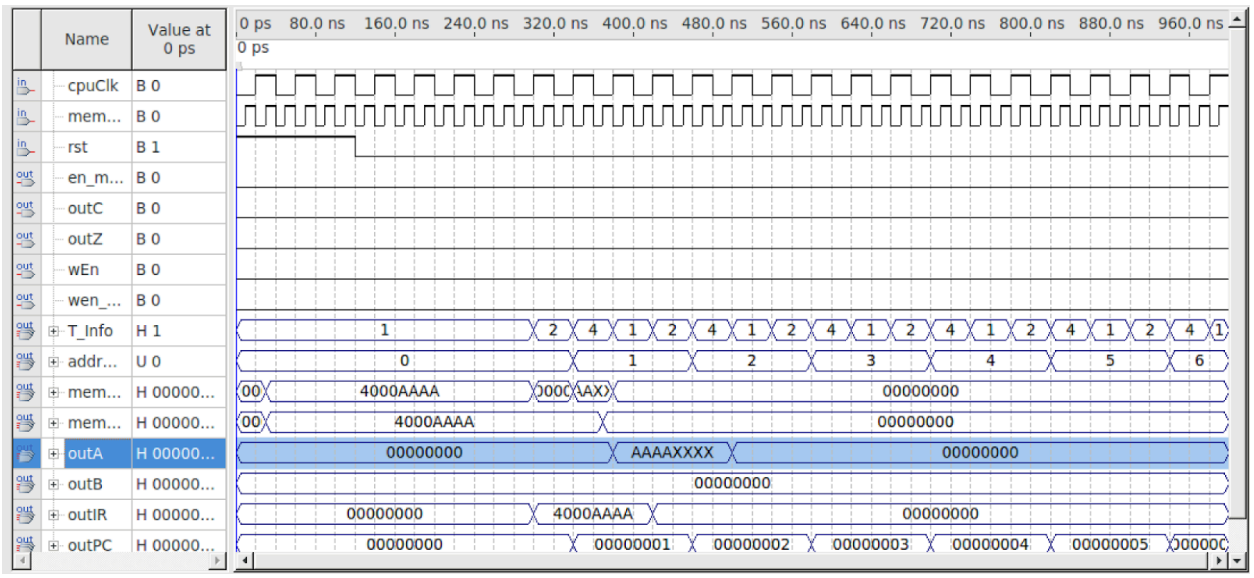
2.1 LDAI, STA, CLRA, LDA



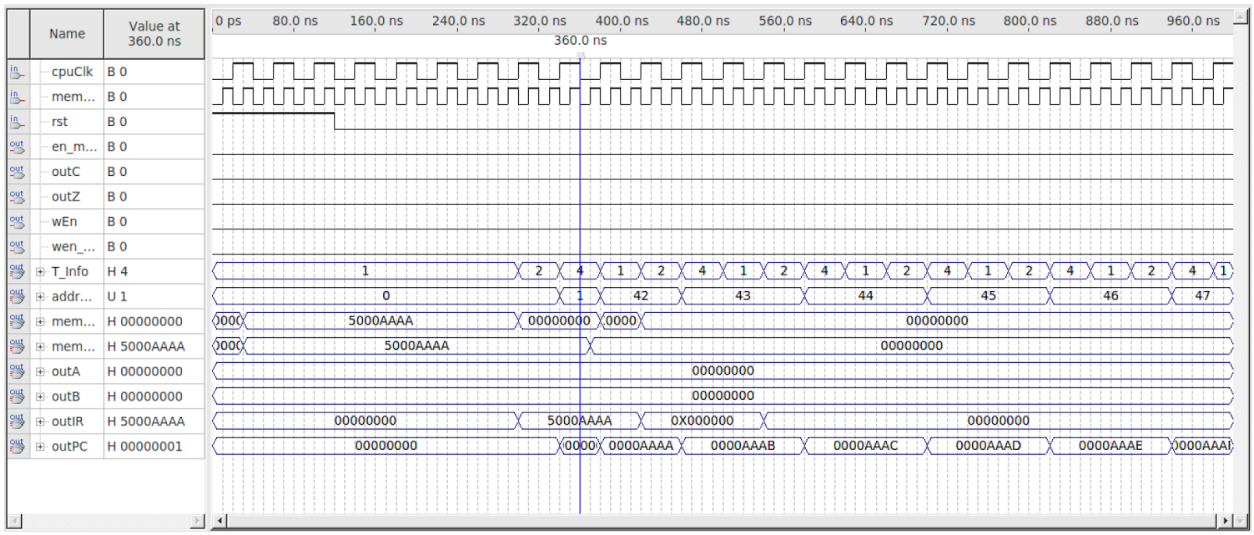
2.2 LDBI, STB, CLRB, LDB


Name
Value at 0 ps
rst
en_m...
outC
outZ
wEn
wen_...
T_Info
addr...
mem...
mem...
outA
outB
outIR
outPC
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

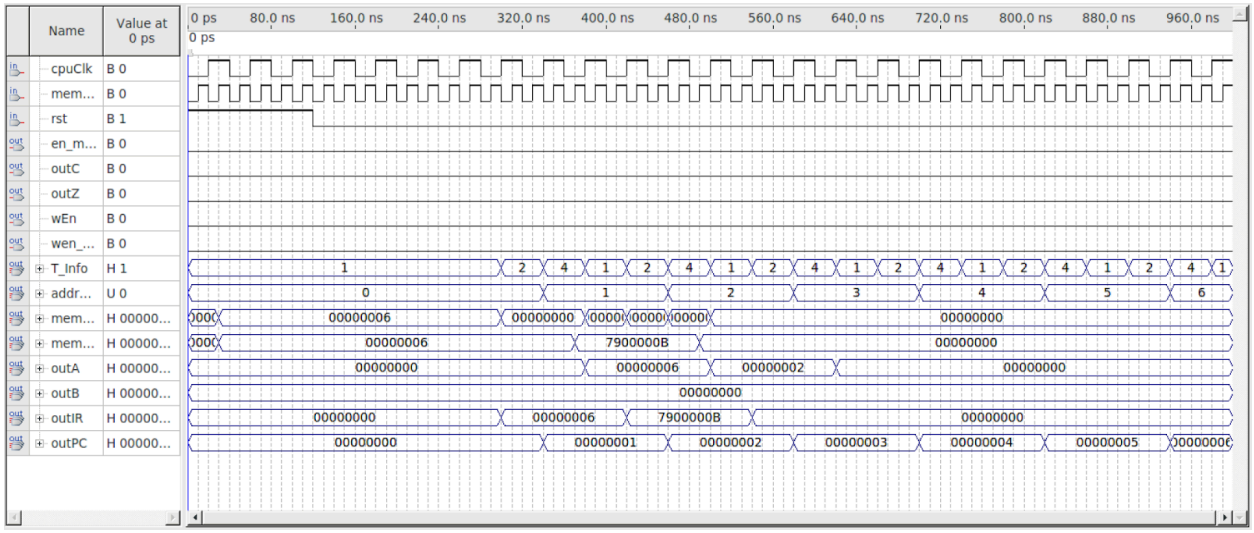
2.3 LUI


Name
Value at 0 ps
cpuClk
mem...
rst
en_m...
outC
outZ
wEn
wen_...
T_Info
addr...
mem...
mem...
outA
outB
outIR
outPC
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

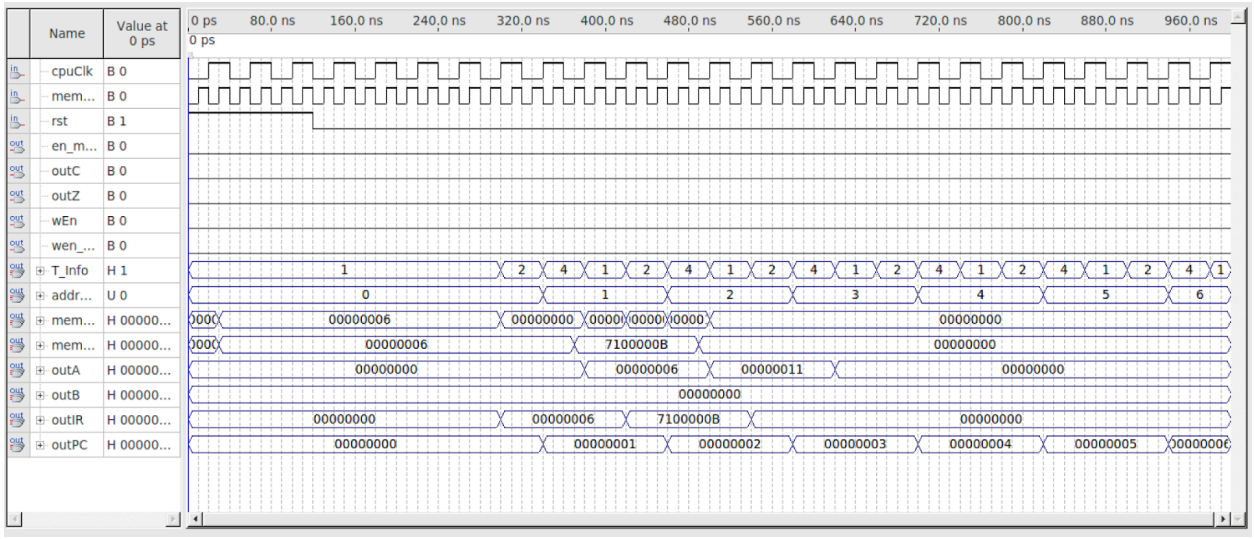
2.4 JMP


Name
Value at 360.0 ns
cpuClk B 0
mem... B 0
rst B 0
en_m... B 0
outC B 0
outZ B 0
wEn B 0
wen_... B 0
T_Info H 4
addr... U 1
mem... H 00000000
mem... H 5000AAAA
outA H 00000000
outB H 00000000
outIR H 5000AAAA
outPC H 00000001
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
360.0 ns

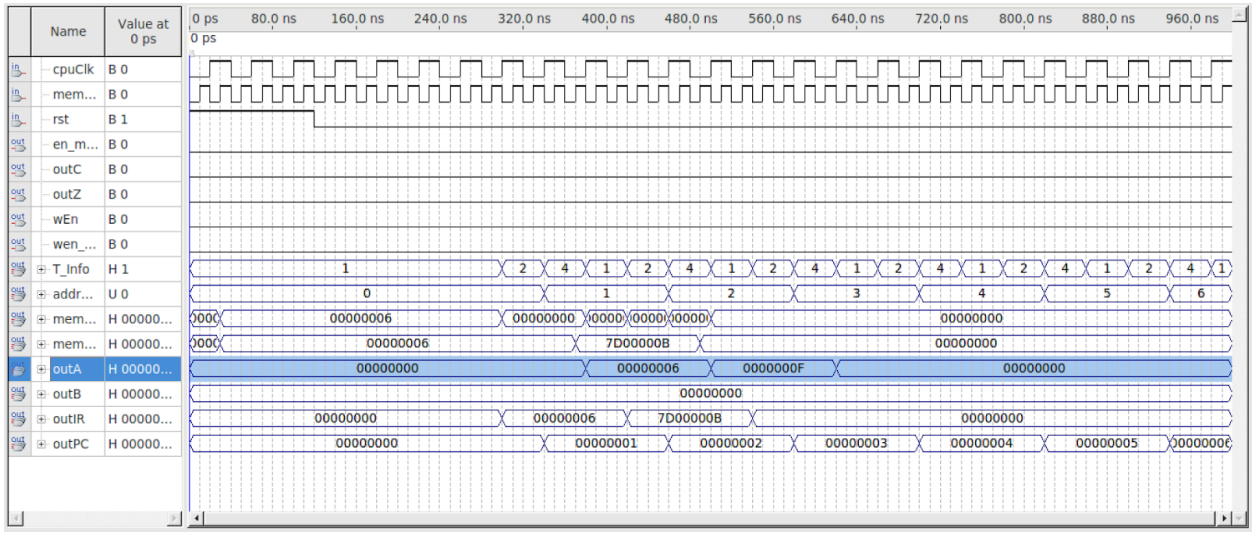
2.5 ANDI


Name
Value at 0 ps
cpuClk B 0
mem... B 0
rst B 1
en_m... B 0
outC B 0
outZ B 0
wEn B 0
wen_... B 0
T_Info H 1
addr... U 0
mem... H 00000...
mem... H 00000...
outA H 00000...
outB H 00000...
outIR H 00000...
outPC H 00000...
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
0 ps

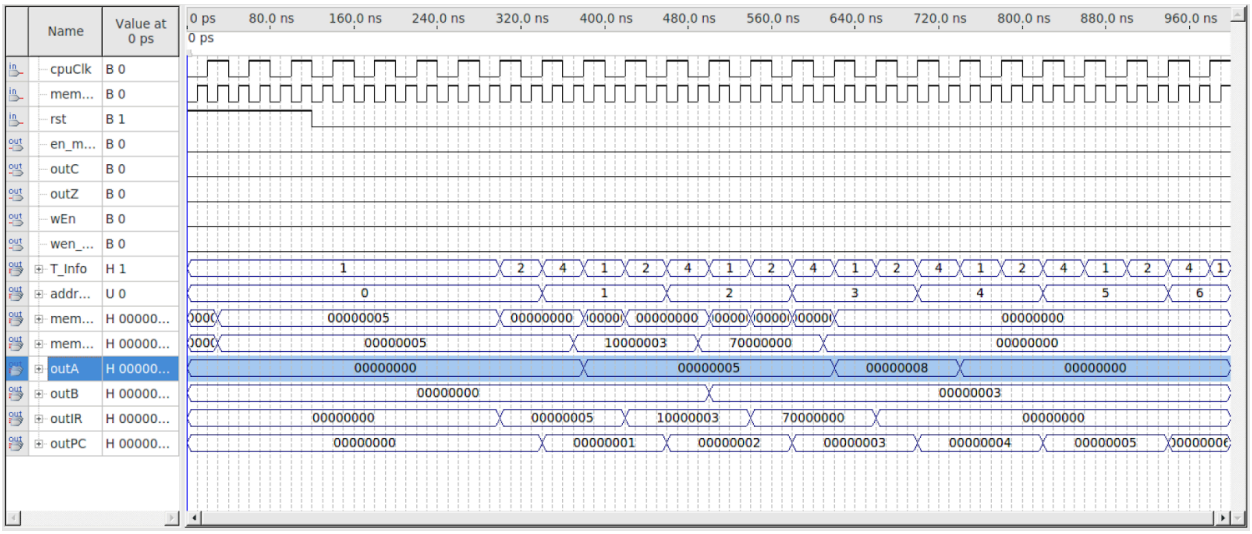
2.6 ADDI



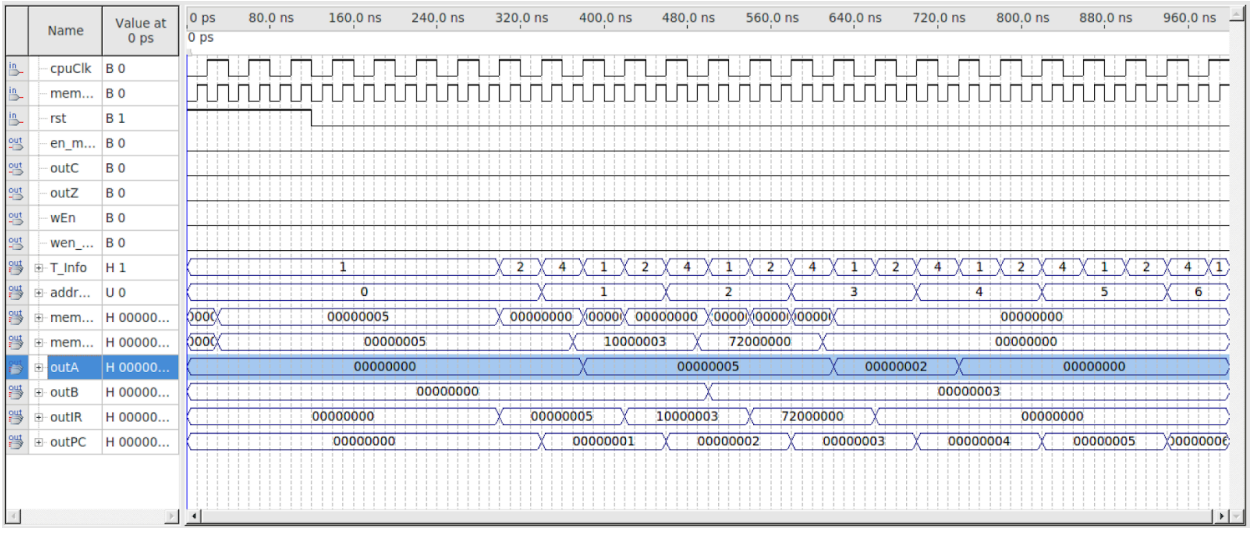
2.7 ORI



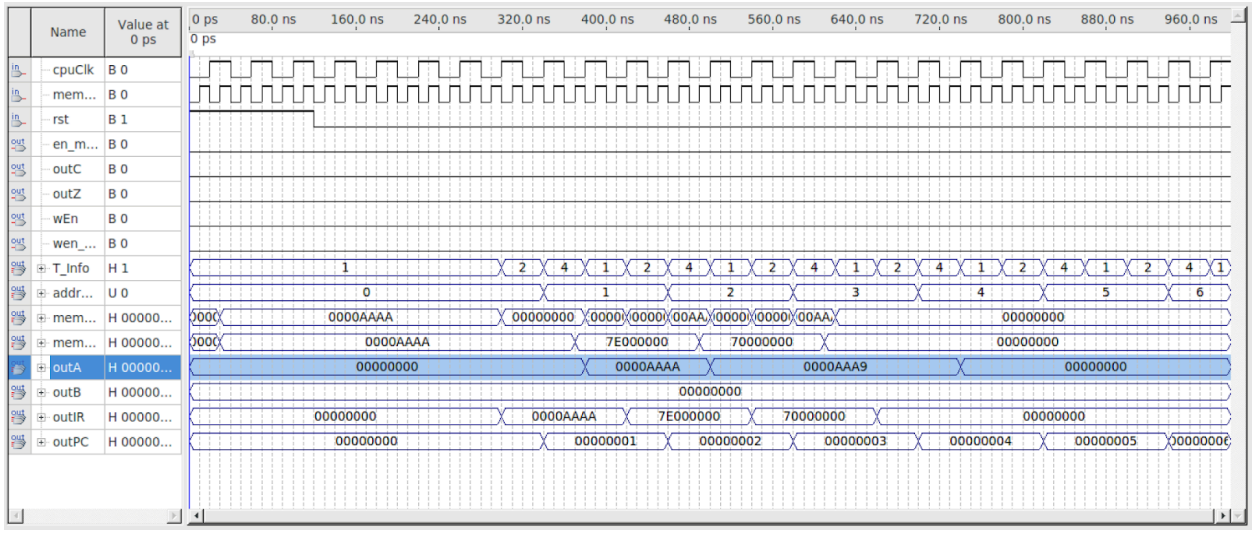
2.8 ADD



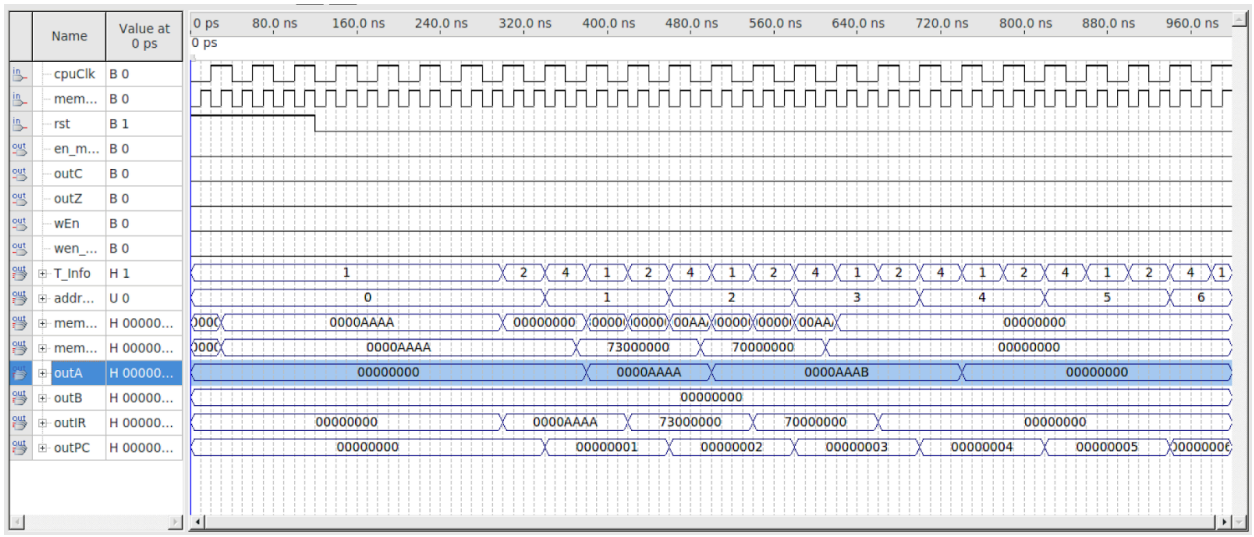
2.9 SUB



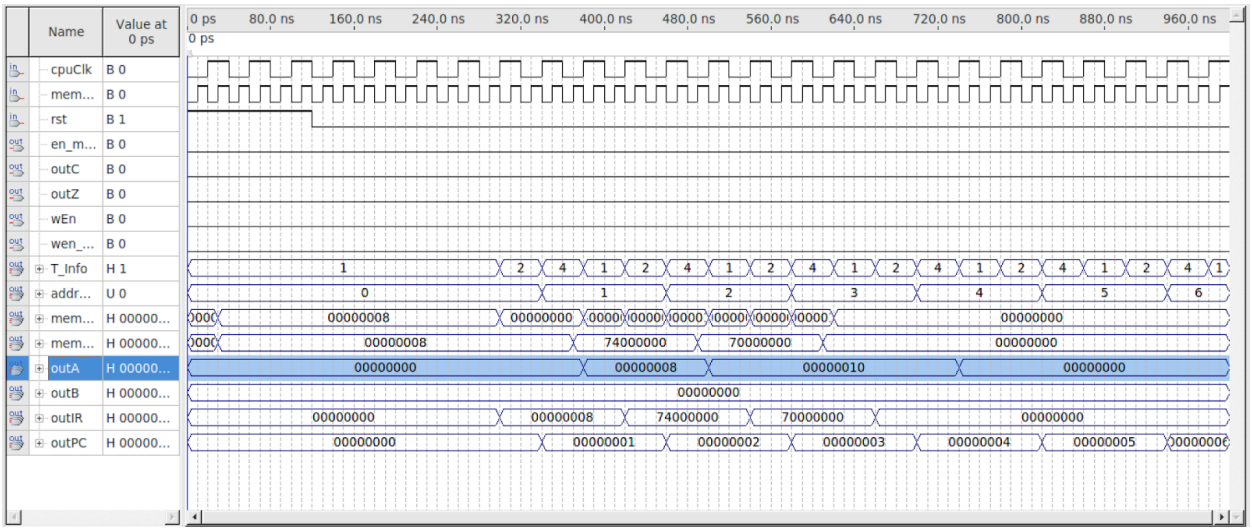
2.10 DECA


Name
Value at 0 ps
cpuClk B 0
mem... B 0
rst B 1
en_m... B 0
outC B 0
outZ B 0
wEn B 0
wen_... B 0
T_Info H 1
addr... U 0
mem... H 00000...
mem... H 00000...
outA H 00000...
outB H 00000...
outIR H 00000...
outPC H 00000...
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
0 ps

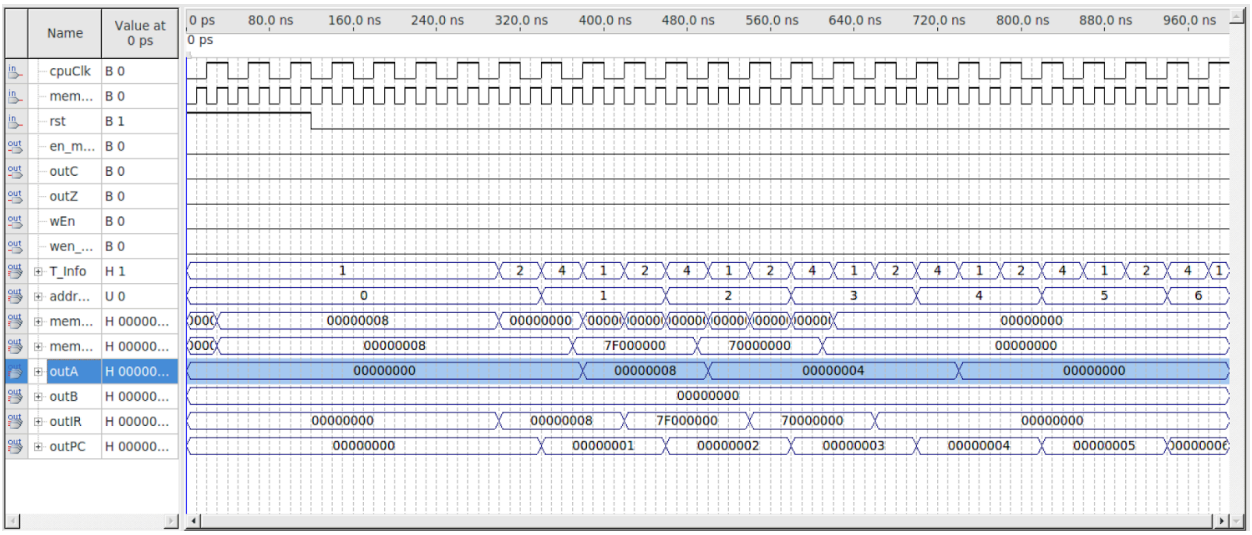
2.11 INCA


Name
Value at 0 ps
cpuClk B 0
mem... B 0
rst B 1
en_m... B 0
outC B 0
outZ B 0
wEn B 0
wen_... B 0
T_Info H 1
addr... U 0
mem... H 00000...
mem... H 00000...
outA H 00000...
outB H 00000...
outIR H 00000...
outPC H 00000...
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
0 ps

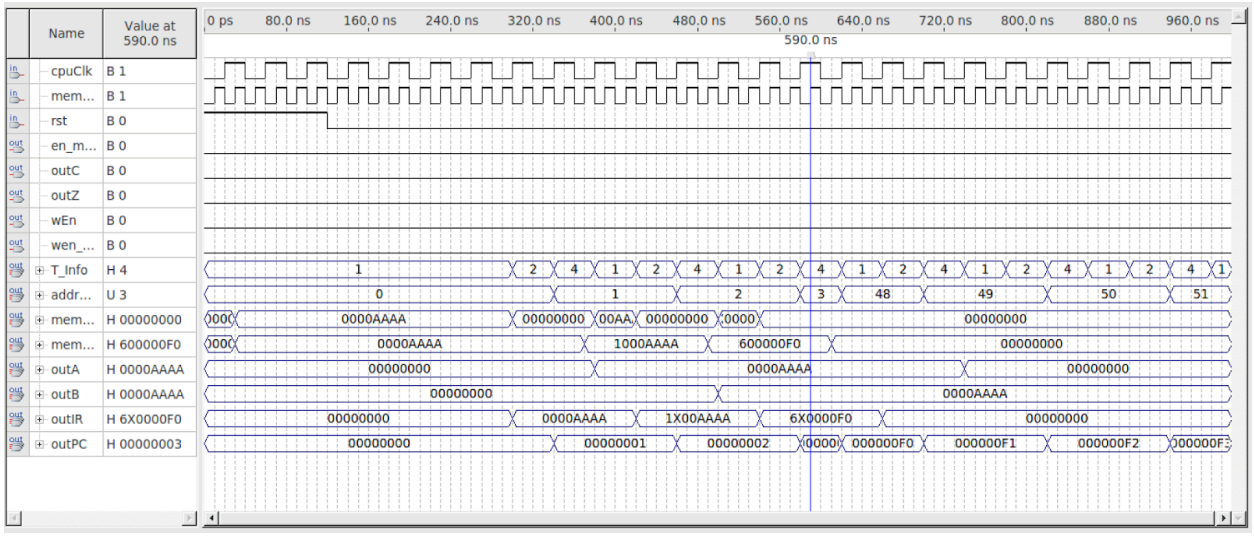
2.12 ROL



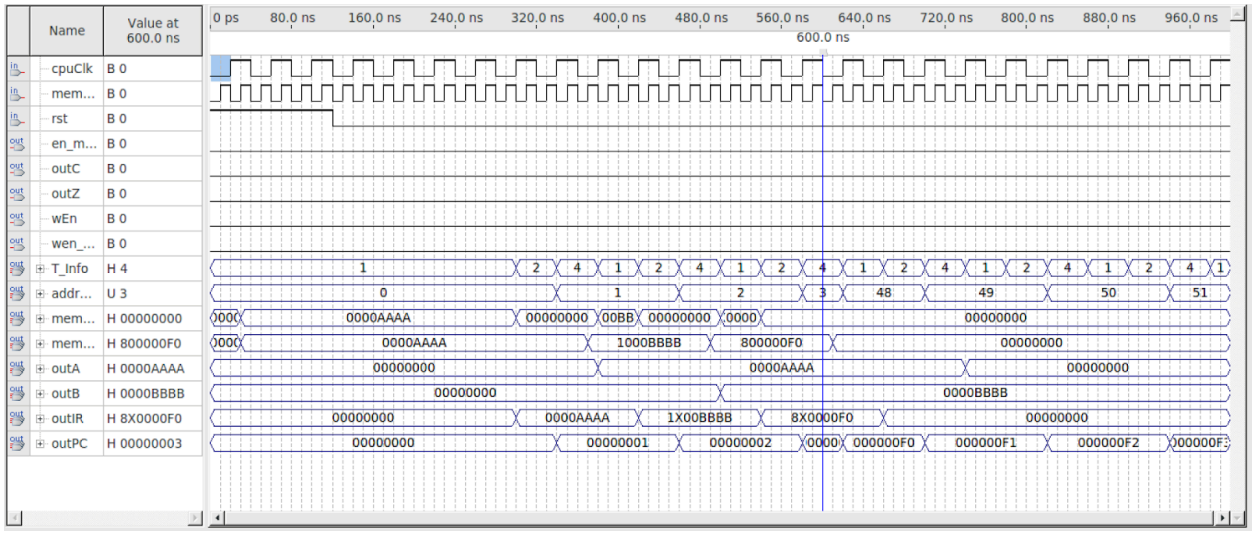
2.13 ROR



2.14 BEQ



2.15 BNE



3.0 Bonus Project

3.1 Problem 1

The given problem can be written as,

```
if (a == b){PC  = branch1();}        //If a is b, go to branch1()
if(a > b){PC = branch2();}           //If a > b, go to branch2()
PC = continue();                     //Otherwise, go to continue()
```

Note that the condition $a > b$ is also equivalent to $((a - b)\ \&\ 0x80000000) == 0$. The assembly code is therefore,

```
; Constant used for the condition "a > b"
LUI 0x8000
STA 0x0

; a and b are assumed to be in Register A and B respectively
LDAI  a
LDBI  b

; ACTUAL ASSEMBLY CODE
BEQ branch1          ;if a == b, go to branch1
SUB                  ;if a > b, go to branch2
LDB 0x0              ;       "
AND                  ;       "
CLRB                 ;       "
BEQ branch2          ;       "
JMP continue         ;Otherwise, go to continue

; the subroutines are filled with an instruction for testing
branch1:
   LDAI 0x1          ;A = 1
   JMP endif
branch2:
   LDAI 0x2          ;A = 2
   JMP endif
continue:
   LDAI 0x3          ;A = 3
endif:
```

For each subroutine, the Register A is loaded with a unique value to indicate the "type" of branching (i.e. branch1, branch2, and continue have a value of 1, 2, and 3 respectively). The correctness of the designed assembly code are presented in Figure 3.1, Figure 3.2, and Figure 3.3 below.

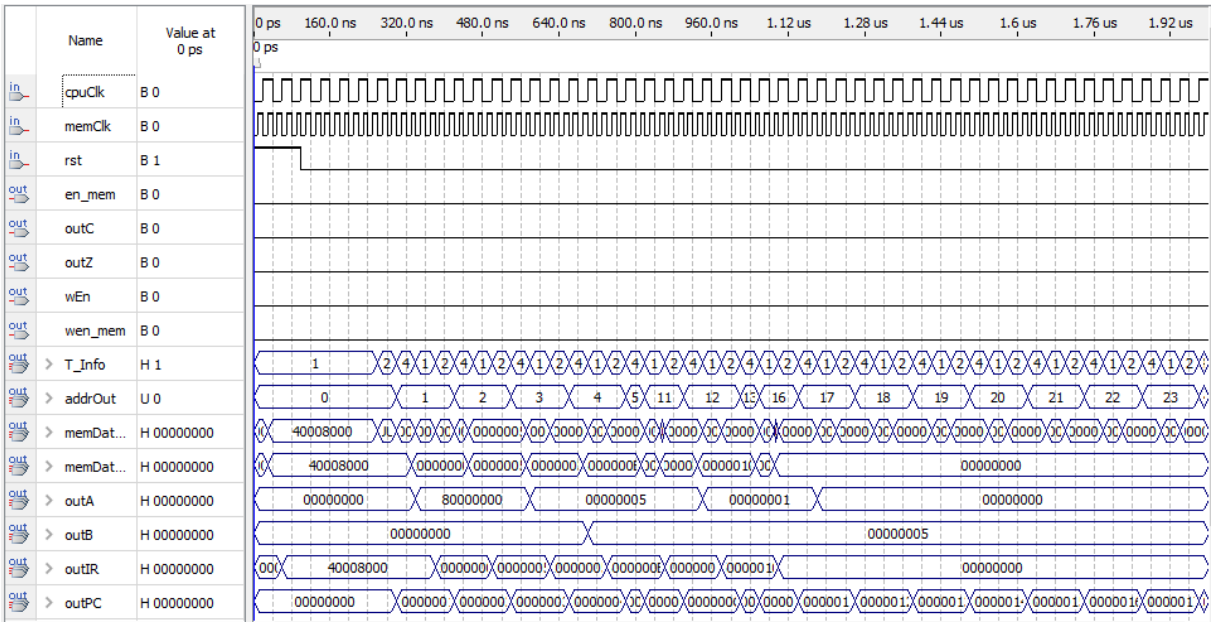


Figure 3.1: Value of Register A (shown at $1\ us$) when $a = b = 5$.

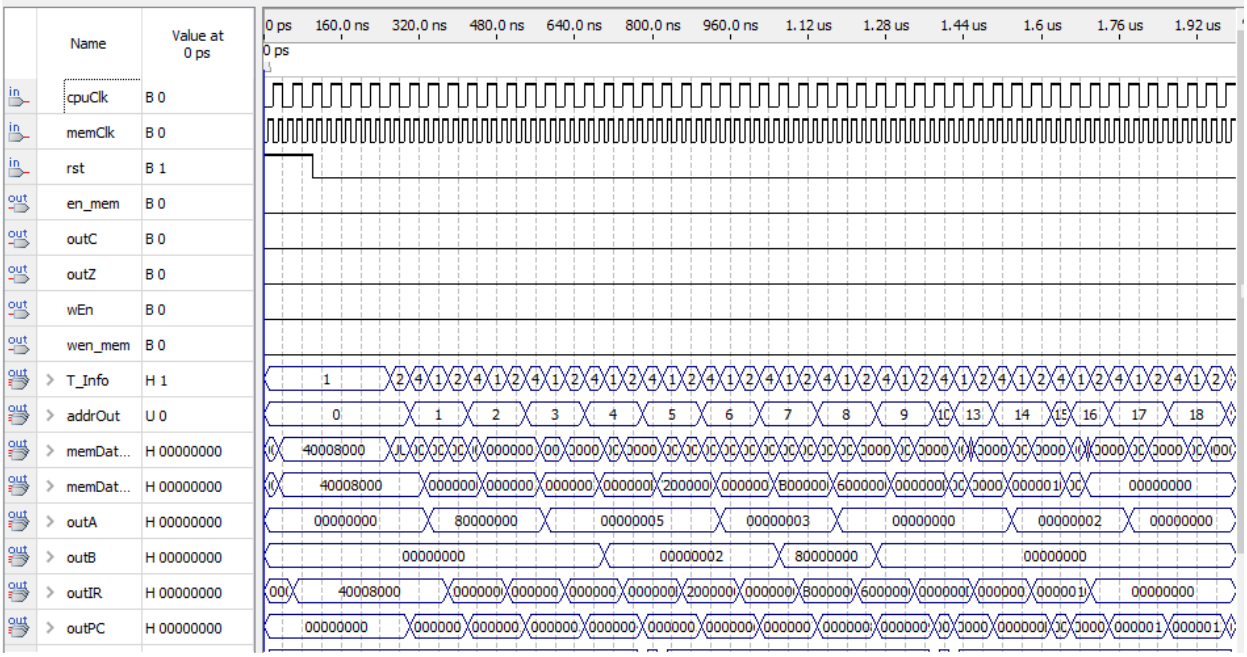


Figure 3.2: Value of Register A (shown at $1.7\ us$) when $a = 5$ and $b = 2$.

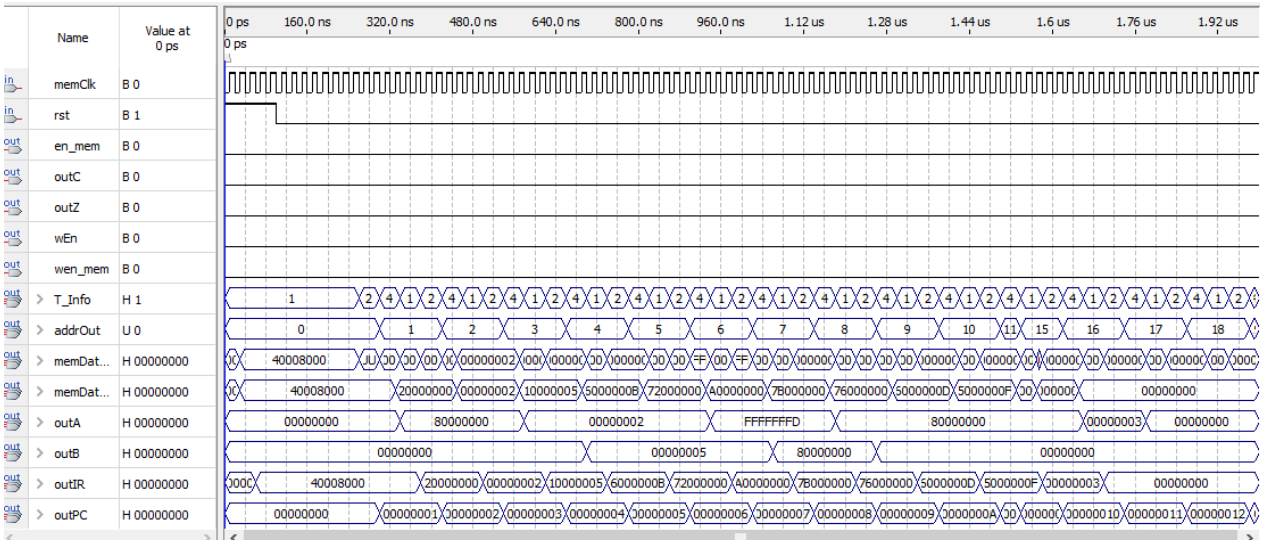


Figure 3.3: Value of Register A (shown at $1.7\ us$) when $a = 2$ and $b = 5$.

3.2 Problem 2

The given problem can be written as,

```
a = 1;
for (i = 1; i != 6; i++){
   for(j = 0; j != i; j++){      //a = a*2^i
      a = a << 1;               //     "
   }
}
```

Since the CPU is limited in terms of the number of registers in the register file, the data memory component is heavily utilized to properly solve this problem. The variables a, i, and j are stored in 0x0, 0x1, and 0x2 respectively within the assembly code below,

```
; Initialize variable a and i
LDAI 0x1
STA 0x0              ; a = 1
STA 0x1              ; i = 1

loop1:
    LDBI 0x6         ; if i == 6, break
    BEQ exit1        ;      "
    CLRA             ; let j = 0
    STA 0x2          ;    "
    LDB 0x1          ;    "
    loop2:
        BEQ exit2    ; if j == i, break
        LDA 0x0      ; a << 1
        ROL          ;    "
        STA 0x0      ;    "
```

```
        LDA 0x2       ; j++
        INCA          ;   "
        STA 0x2       ;   "
        JMP loop2     ; repeat
    exit2:
    LDA 0x1           ; i++
    INCA              ;   "
    STA 0x1           ;   "
    JMP loop1         ; repeat
exit1:
     LDA 0x0          ; get the final value of “a” (i.e. A = a)
```