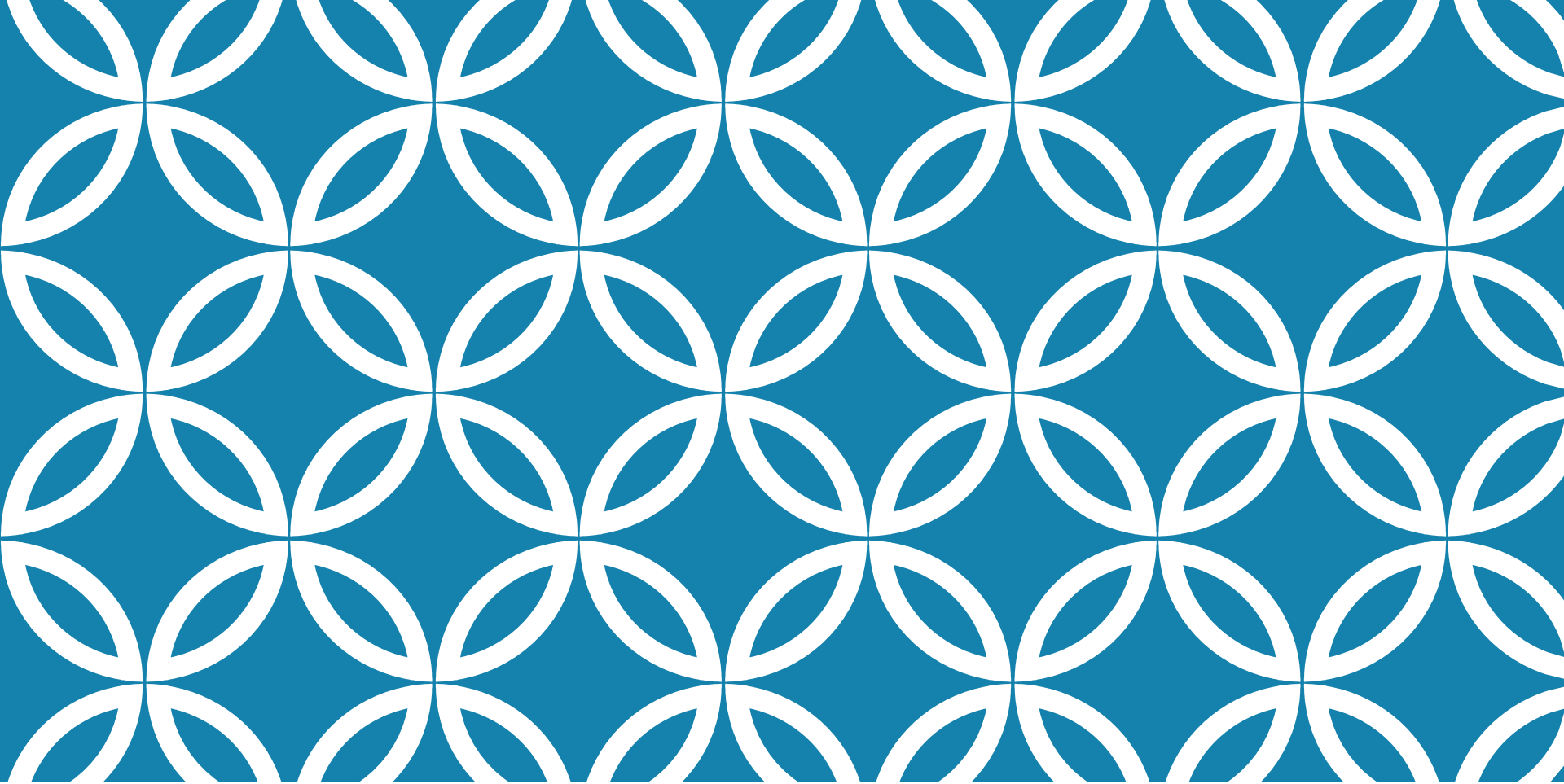


COMPUTER ARCHITECTURE LAB 8

TESTING MIPS PROCESSOR

FCIS Ainshams University
Spring 2021

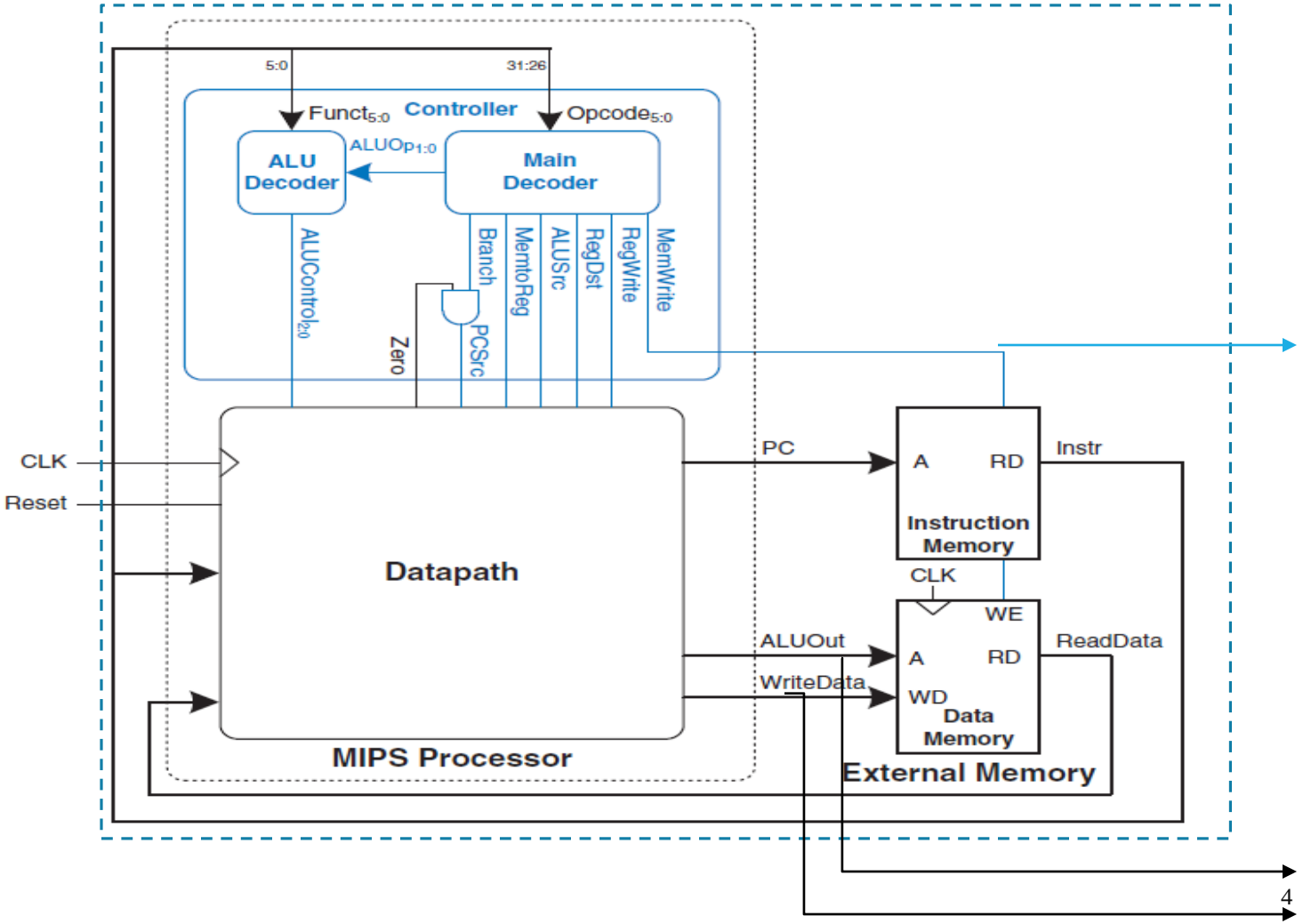
AGENDA

- Hands-On 1
  - Implement top module.
  - Test the Top module (MIPS Processor)

TOP MODULE

- Much of what needs to be done to implement the instructions is the same, independent of the exact class of instruction
- For every instruction, the first two implementation steps are identical:
  - use PC to fetch the instruction from the instruction memory
  - use the instruction fields to select one or two registers to read:
    - lw requires reading only one register
    - j does not require accessing any register
    - All other instructions require reading two registers
- After these two steps, the actions required to complete the instruction depend on the instruction class

TOP M0DULE BLOCK



DATA MEMORY

For simplicity, dmem has 64 words (locations)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use STD.TEXTIO.all;
use IEEE.NUMERIC_STD_UNSIGNED.all;
entity dmem is -- data memory
port(clk, we: in STD_LOGIC;
a, wd: in STD_LOGIC_VECTOR (31 downto 0);
rd: out STD_LOGIC_VECTOR (31 downto 0));
end;
architecture behave of dmem is
begin
process is
type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
variable mem: ramtype;
begin
-- read or write memory
loop
if rising_edge(clk) then
if (we='1') then mem (to_integer(a(7 downto 2))):= wd;
end if;
end if;
rd <= mem (to_integer(a (7 downto 2)));
wait on clk, a;
end loop;
end process;
end;
```

Given to you

INSTRUCTION MEMORY

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use STD.TEXTIO.all;
use IEEE.NUMERIC_STD_UNSIGNED.all;
entity imem is -- instruction memory
port(a: in STD_LOGIC_VECTOR(5 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end;
architecture behave of imem is
begin
process is
file mem_file: TEXT;
variable L: line;
variable ch: character;
variable i, index, result: integer;
type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
variable mem: ramtype;
begin
-- initialize memory from file
for i in 0 to 63 loop -- set all contents low
mem(i) := (others => '0');
end loop;
index := 0;
FILE_OPEN (mem_file, "C:/docs/DDCA2e/hdl/memfile.dat", READ_MODE);
while not endfile(mem_file) loop
readline(mem_file, L);
result := 0;
for i in 1 to 8 loop
read (L, ch);
if '0' <= ch and ch <= '9' then
result := character'pos(ch) - character'pos('0');
elsif 'a' <= ch and ch <= 'f' then
result := character'pos(ch) - character'pos('a')+10;
else report "Format error on line" & integer'
image(index) severity error;
end if;
mem(index)(35-i*4 downto 32-i*4) :=to_std_logic_vector(result,4);
end loop;
index := index + 1;
end loop;
-- read memory
loop
rd <= mem(to_integer(a));
wait on a;
end loop;
end process;
end;
```

For simplicity, imem has 6 bits address input and 64 words (locations)

Given to you

HANDS-ON1: TOP MODULE

I. Add the MIPS, imem, dmem Modules as components in your package

```
component mips
port(clk, reset: in STD_LOGIC;
pc: out STD_LOGIC_VECTOR(31 downto 0);
instr: in STD_LOGIC_VECTOR(31 downto 0);
memwrite: out STD_LOGIC;
aluout, writedata: out STD_LOGIC_VECTOR(31 downto 0);
readdata: in STD_LOGIC_VECTOR(31 downto 0));
end component;
component imem
port(a: in STD_LOGIC_VECTOR(5 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end component;
component dmem
port(clk, we: in STD_LOGIC;
a, wd: in STD_LOGIC_VECTOR(31 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end component;
```

Component imem
..
End component
Component dmem
..
End component

HANDS-ON1: TOP MODULE

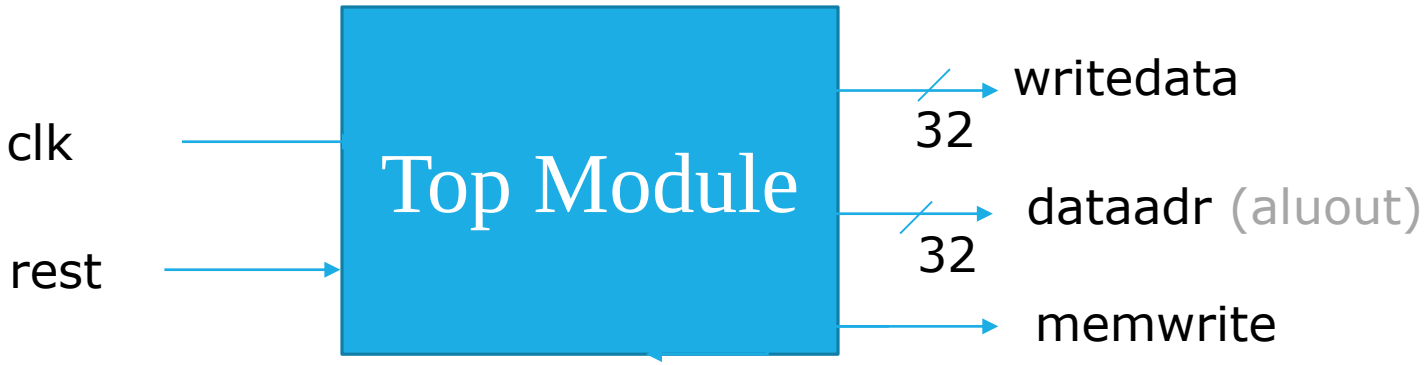
All vhd files and components till now are:

- MIPS
- Imem, dmem
- Controller
- Maindec
- Aludec
- Datapath
- Registerfile
- ALU
- Mux2
- Sl2
- Adder
- Signext
- Flopr

HANDS-ON1: TOP MODULE

II. Now: Create main module for Top

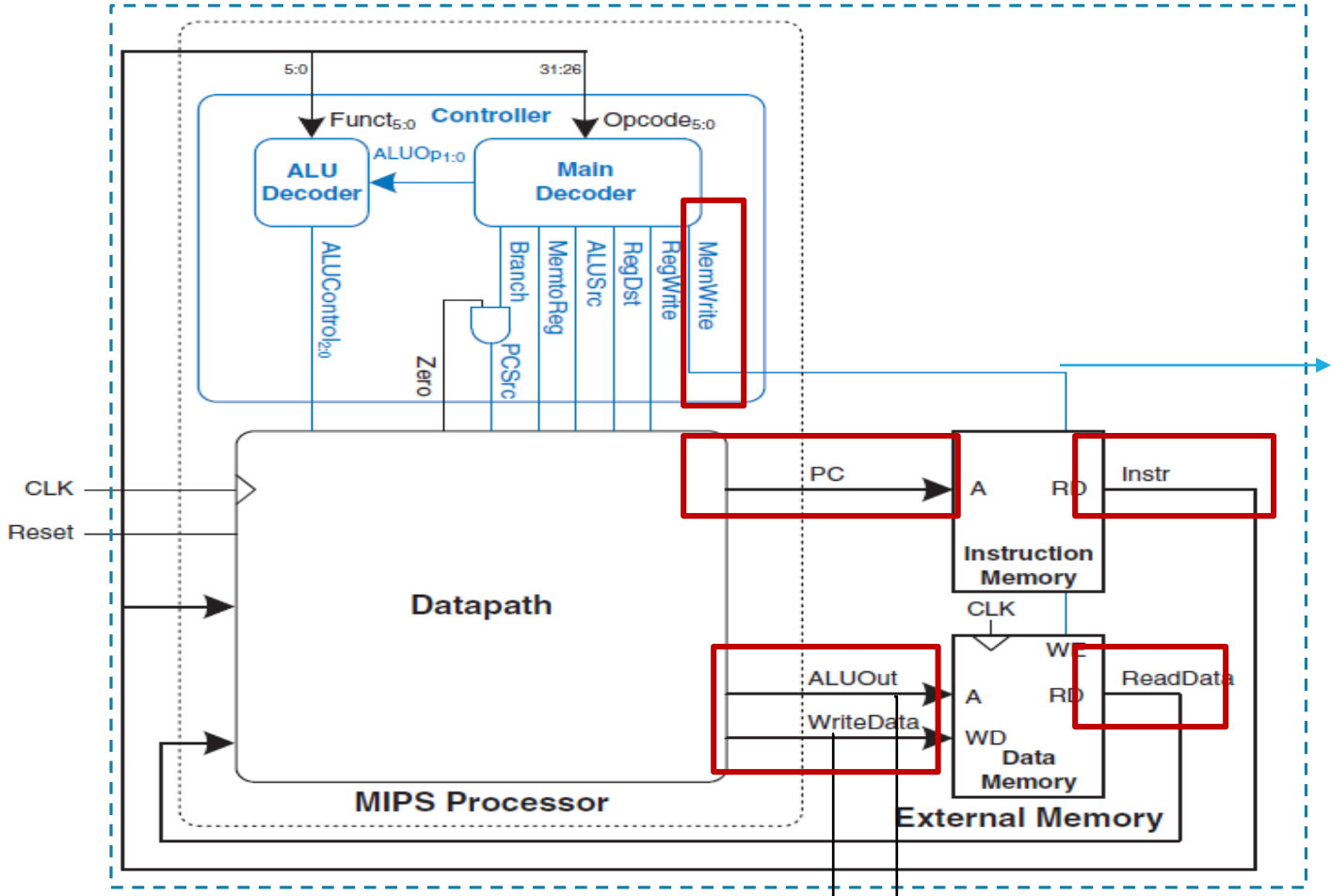
HANDS-ON1: TOP MODULE




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--use IEEE.NUMERIC_STD_UNSIGNED.all;
entity top is -- top-level design for testing
port(clk, reset: in STD_LOGIC;
writedata, dataadr: out  STD_LOGIC_VECTOR(31 downto 0);
memwrite: out STD_LOGIC);
end;
```

HANDS-ON1: TOP MODULE

We will need some signals

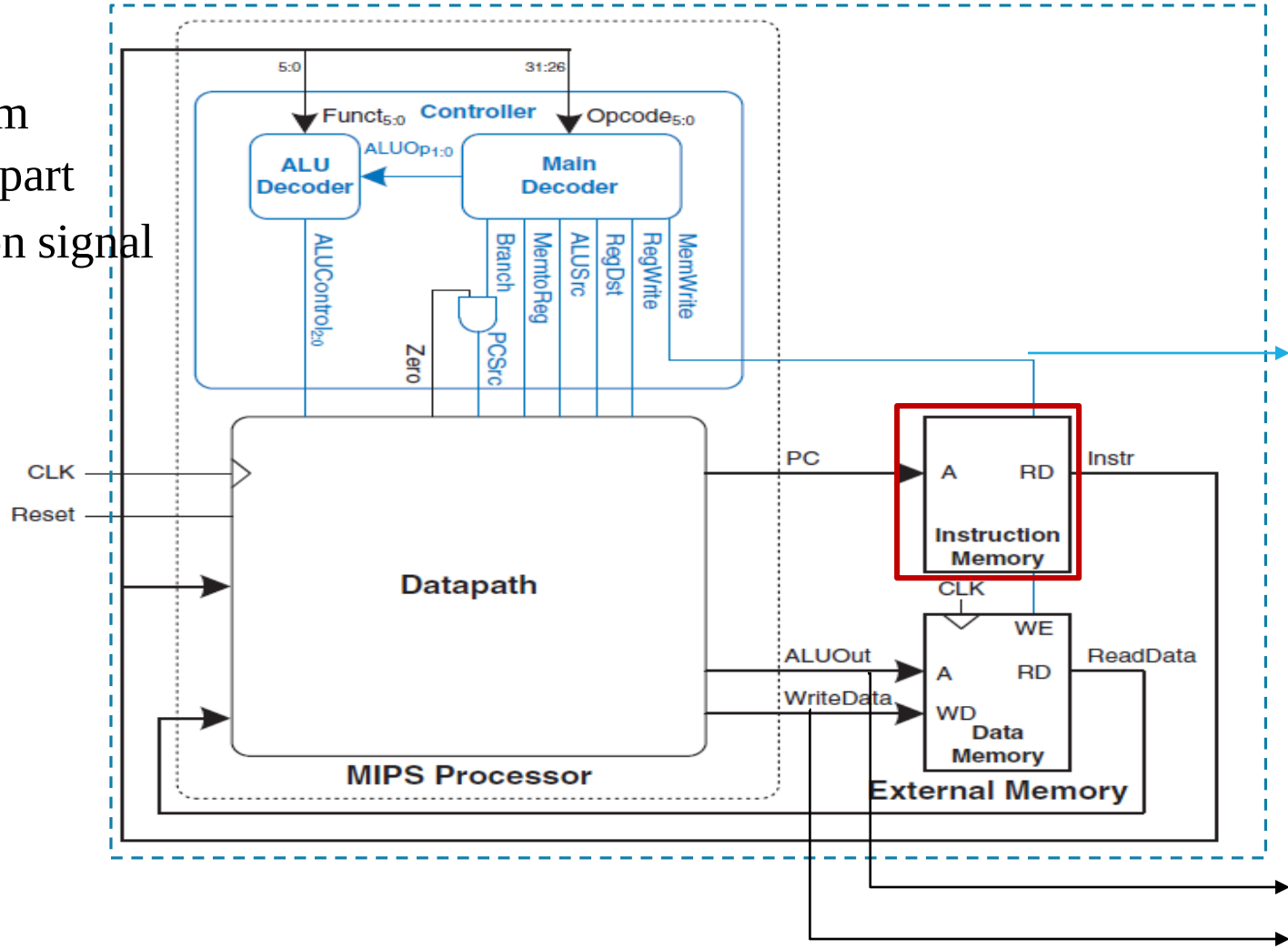



```
signal memwritet:  STD_LOGIC;
signal pc, instr, readdata ,dataadrt,writedatat: STD_LOGIC_VECTOR(31 downto 0);
```

HANDS-ON1: TOP MODULE

Behavior:

A. Port Map the imem
   - I/P is PC signal part
   - O/P is instruction signal



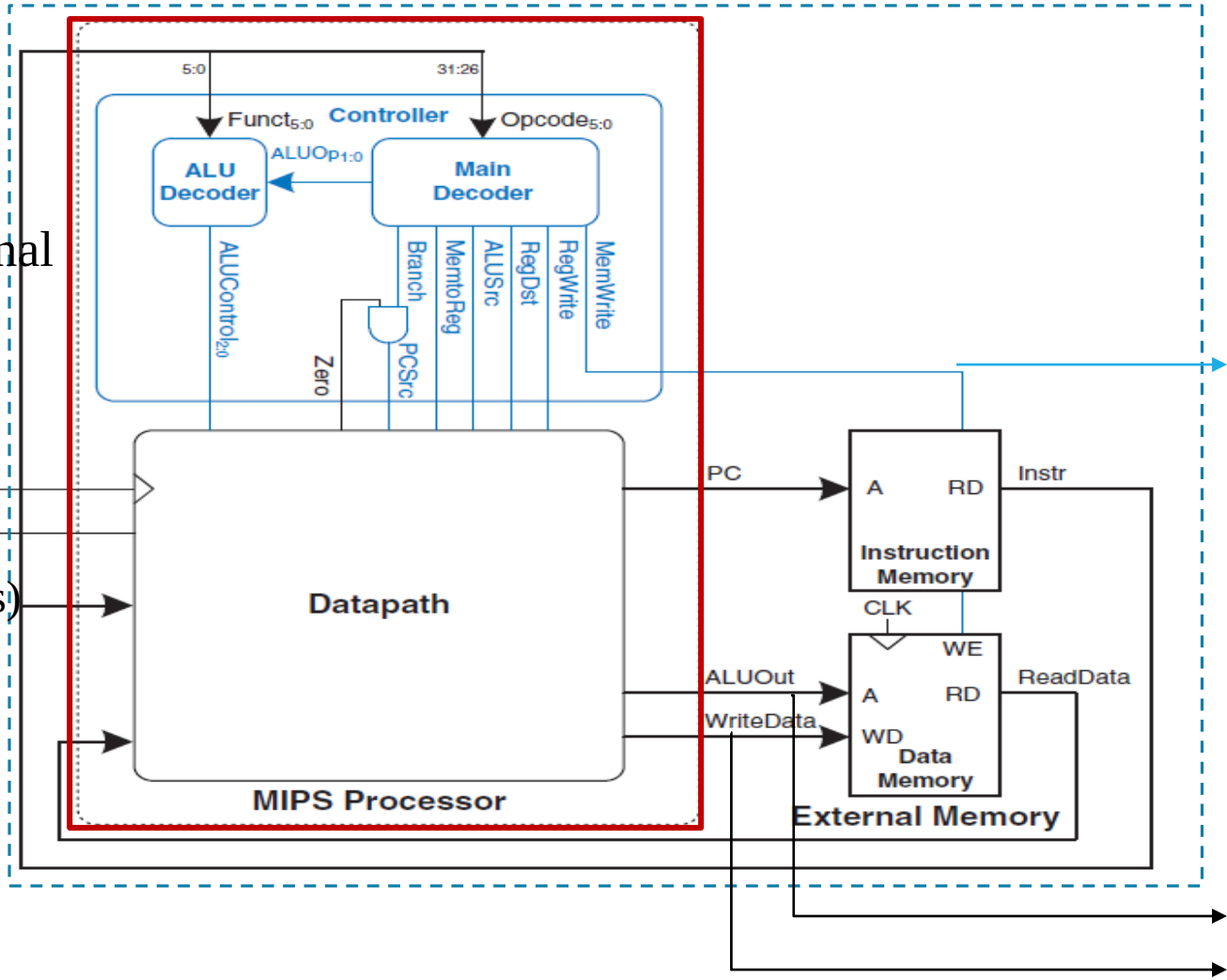
HANDS-ON1: TOP MODULE

Behavior:

A. Port Map the imem
- I/P is PC signal part
- O/P is instruction signal

B. Port MAP the MIPS
- Control I/Ps (clk, rest)
- Data I/Ps

(instruction, readdata)
- O/Ps (remaining signals)



HANDS-ON1: TOP MODULE

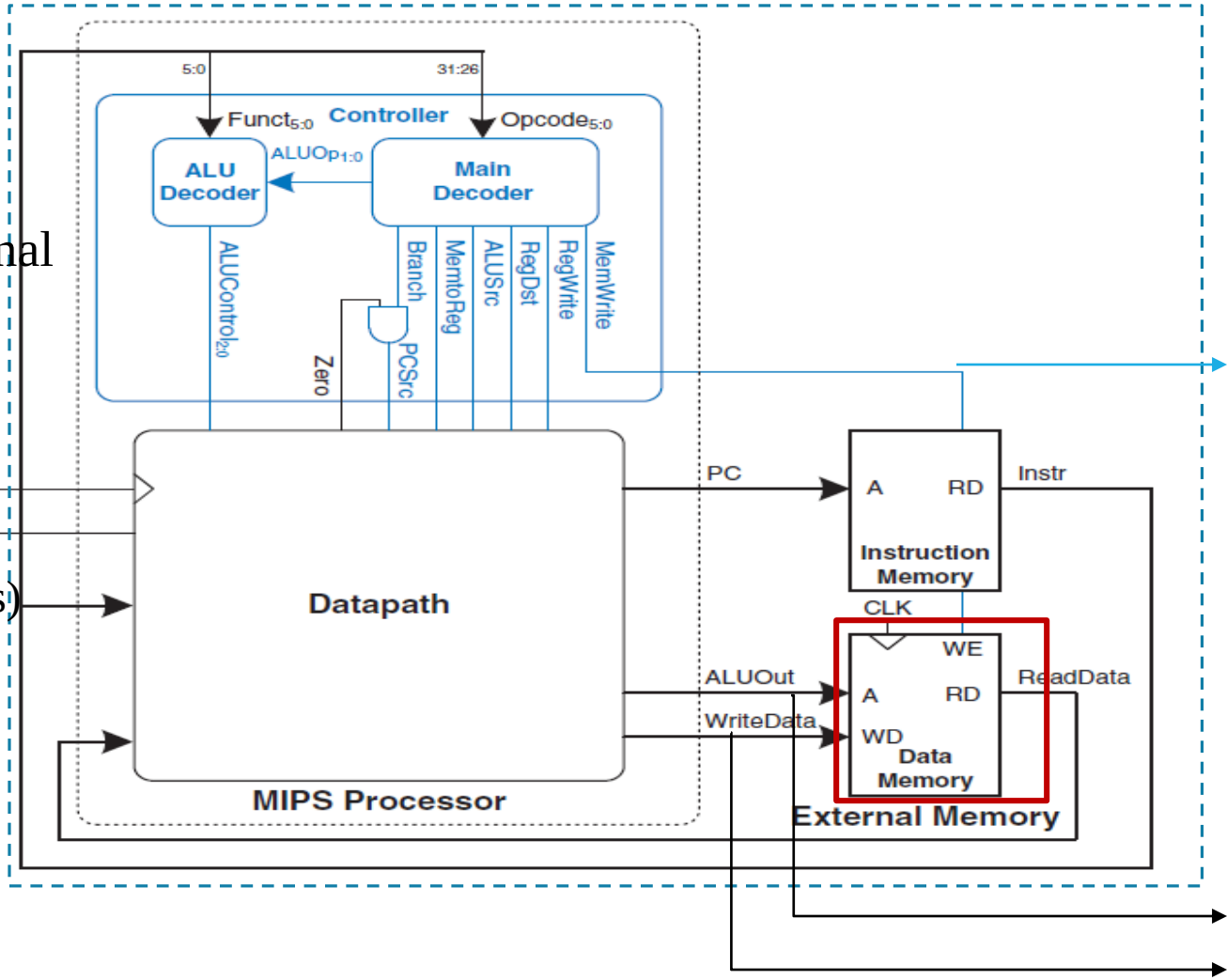
Behavior:

A. Port Map the imem
- I/P is PC signal part
- O/P is instruction signal

B. Port MAP the MIPS
- Control I/Ps (clk, rest)
- Data I/Ps

(instruction, readdata)
- O/Ps (remaining signals)

C. Port Map the dmem ?



HANDS-ON1: TOP MODULE

Behavior:

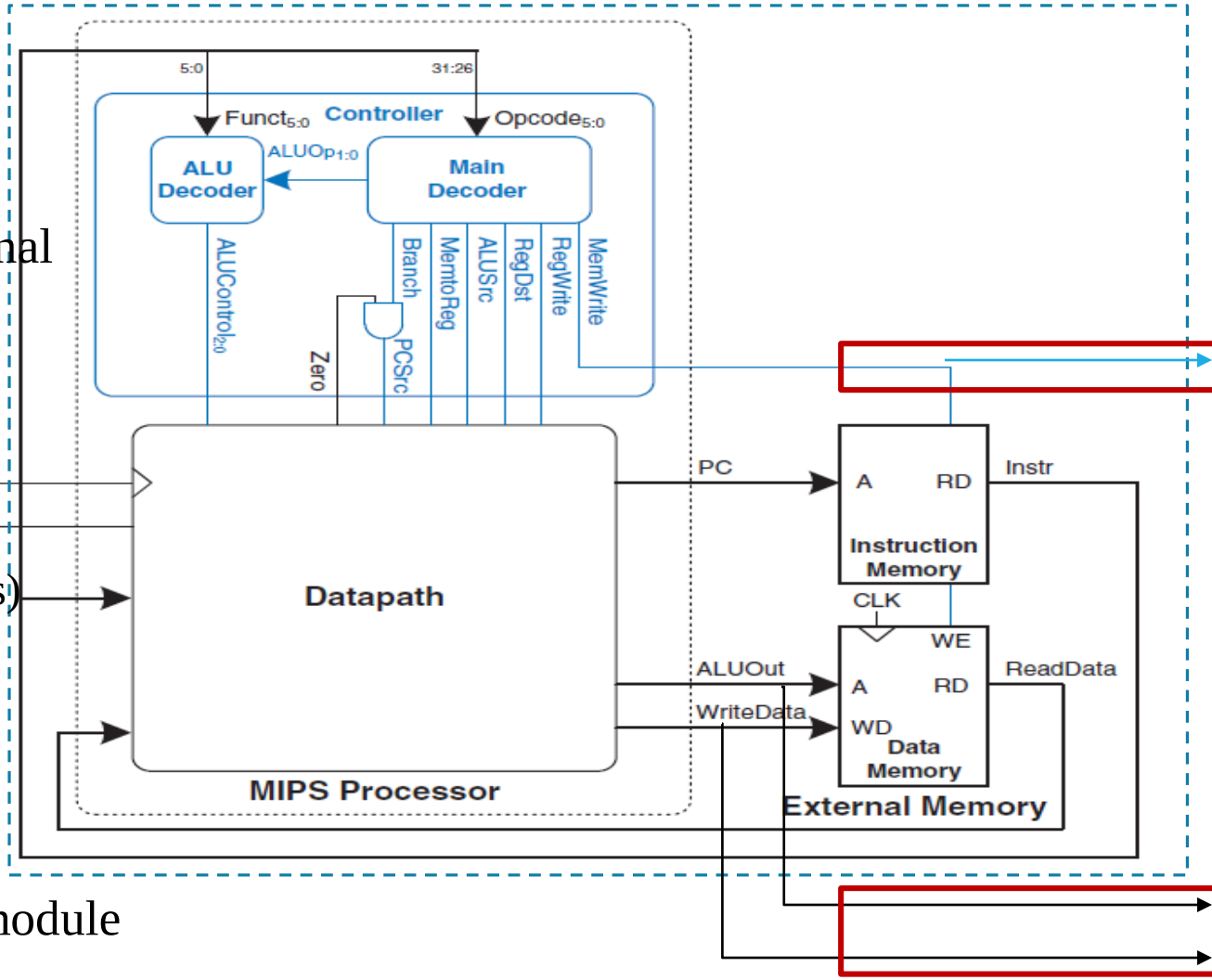
A. Port Map the imem
- I/P is PC signal part
- O/P is instruction signal

B. Port MAP the MIPS
- Control I/Ps (clk, rest)
- Data I/Ps

(instruction, readdata)
- O/Ps (remaining signals)

C. Port Map the dmem ?

D. Assign output of Top module

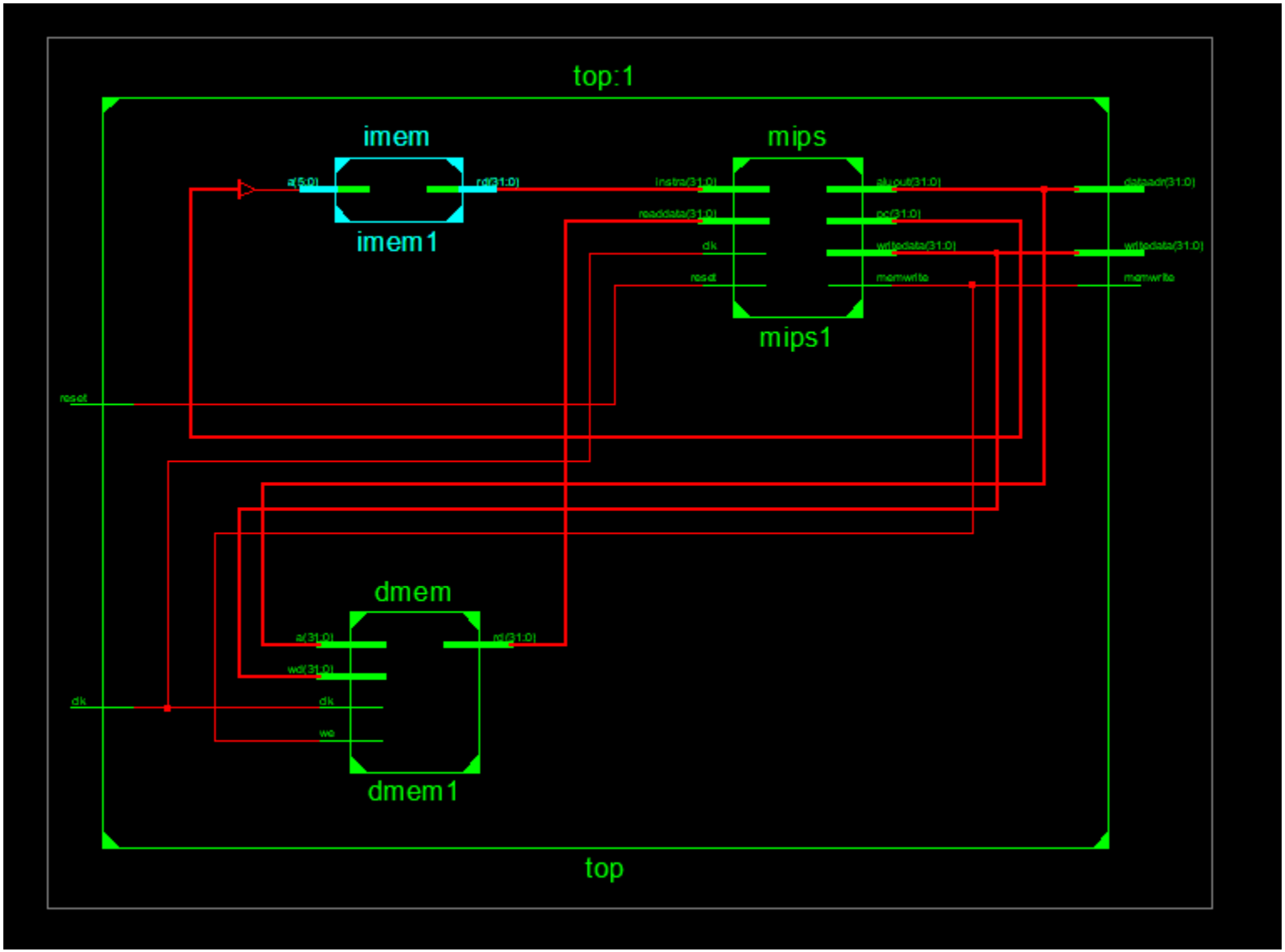


HANDS-ON1: TOP MODULE SOLUTION

```
-- instantiate processor and memories
mips1: mips port map(clk, reset, pc, instr, memwritet,
dataadrt, writedatat, readdata);
imem1: imem port map(pc(7 downto 2), instr);
dmem1: dmem port map(clk, memwritet, dataadrt,
writedatat, readdata);

--assign output
memwrite<=memwritet;
dataadr<=dataadrt;
writedata<=writedatat;
```

HANDS-ON1: TOP MODULE

RTL



HANDS-ON1: TOP MODULE TEST CASE

```
main:   addi $2, $0, 5 # initialize $2 = 5 0 20020005
        addi $3, $0, 12 # initialize $3 = 12 4 2003000c
        addi $7, $3, -9 # initialize $7 = 3 8 2067fff7
        or $4, $7, $2 # $4 = (3 OR 5) = 7 c 00e22025
        and $5, $3, $4 # $5 = (12 AND 7) = 4 10 00642824
        add $5, $5, $4 # $5 = 4 + 7 = 11 14 00a42820
        beq $5, $7, end # shouldn't be taken 18 10a7000a
        slt $4, $3, $4 # $4 = 12 < 7 = 0 1c 0064202a
        beq $4, $0, around # should be taken 20 10800001
        addi $5, $0, 0 # shouldn't happen 24 20050000
around: slt $4, $7, $2 # $4 = 3 < 5 = 1 28 00e2202a
        add $7, $4, $5 # $7 = 1 + 11 = 12 2c 00853820
        sub $7, $7, $2 # $7 = 12 - 5 = 7 30 00e23822
        sw $7, 68($3) # [80] = 7 34 ac670044
        lw $2, 80($0) # $2 = [80] = 7 38 8c020050
        j end # should be taken 3c 08000011
        addi $2, $0, 1 # shouldn't happen 40 20020001
end:    sw $2, 84($0) # write mem[84] = 7 44 ac020054
```

Test the MIPS processor.
add, sub, and, or, slt, addi, lw, sw, beq, j
If successful, it should write the value 7 to address 84

HANDS-ON1: TOP MODULE TEST CASE MEMFILE.DAT

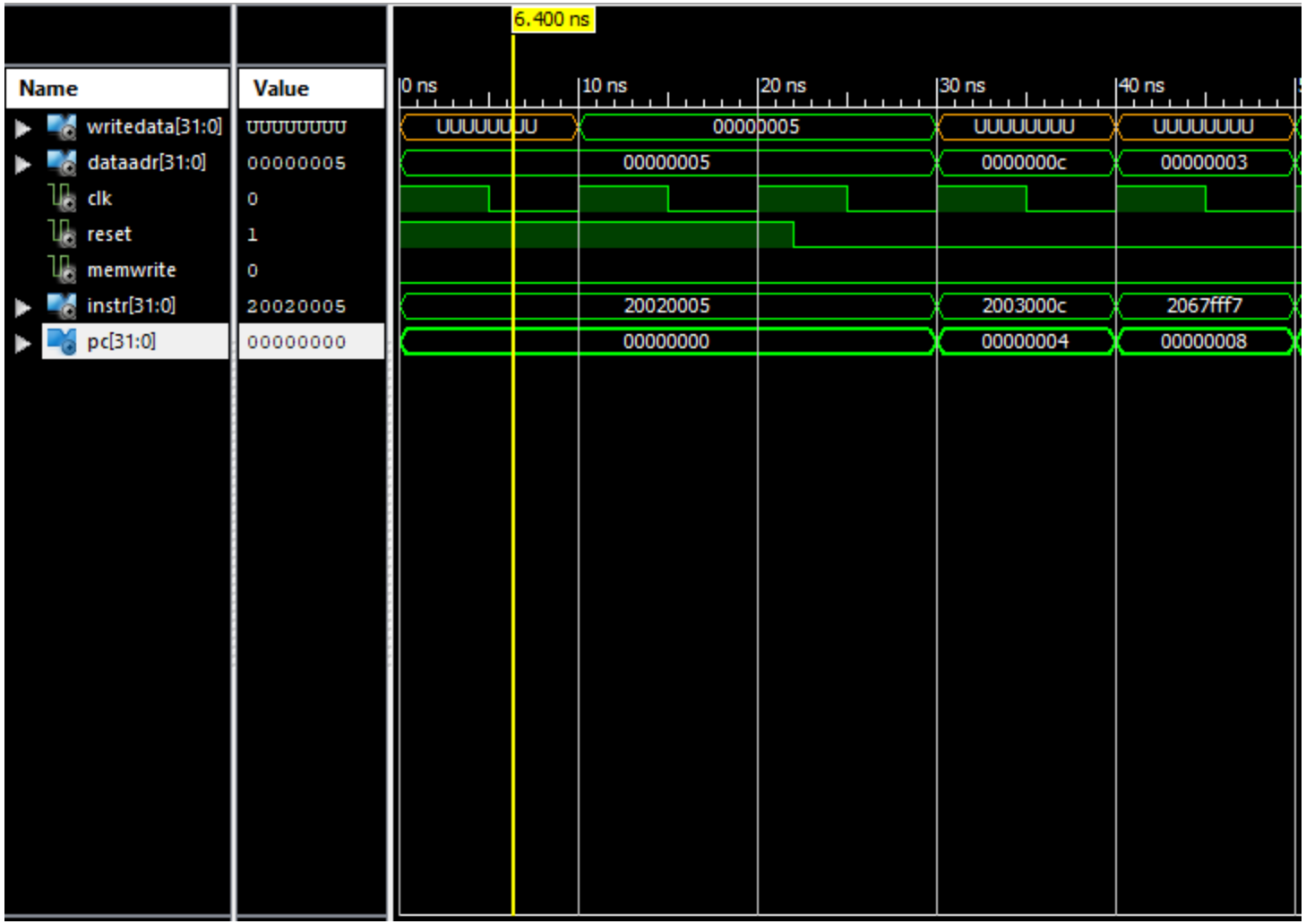
20020005
2003000c
2067fff7
00e22025
00642824
00a42820
10a7000a
0064202a
10800001
20050000
00e2202a
00853820
00e23822
ac670044
8c020050
08000011
20020001
ac020054

Given to you

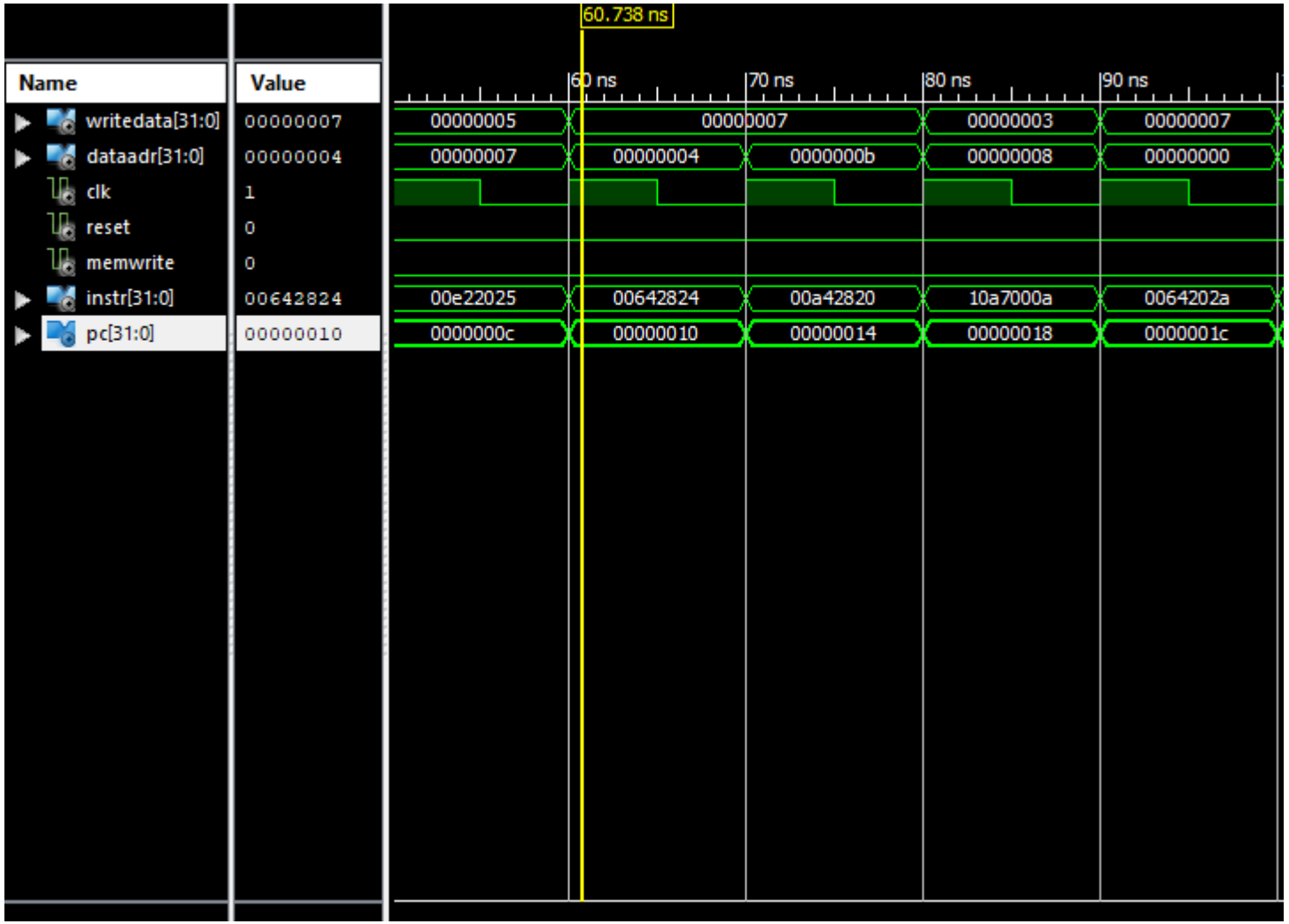
HANDS-ON1: TOP MODULE TEST CASE TEST BENCH

```
process begin
reset <= '1';
wait for 22 ns;
reset <= '0';
wait;
end process;

-- check that 7 gets written to address 84 at end of program
process(clk) begin
if (clk'event and clk = '0' and memwrite = '1') then
if (to_integer(dataadr) = 84 and to_integer
(writedata) = 7) then
report "NO ERRORS: Simulation succeeded" severity failure;
elsif (dataadr /= 80) then
report "Simulation failed" severity failure;
end if;
end if;

end process;
```

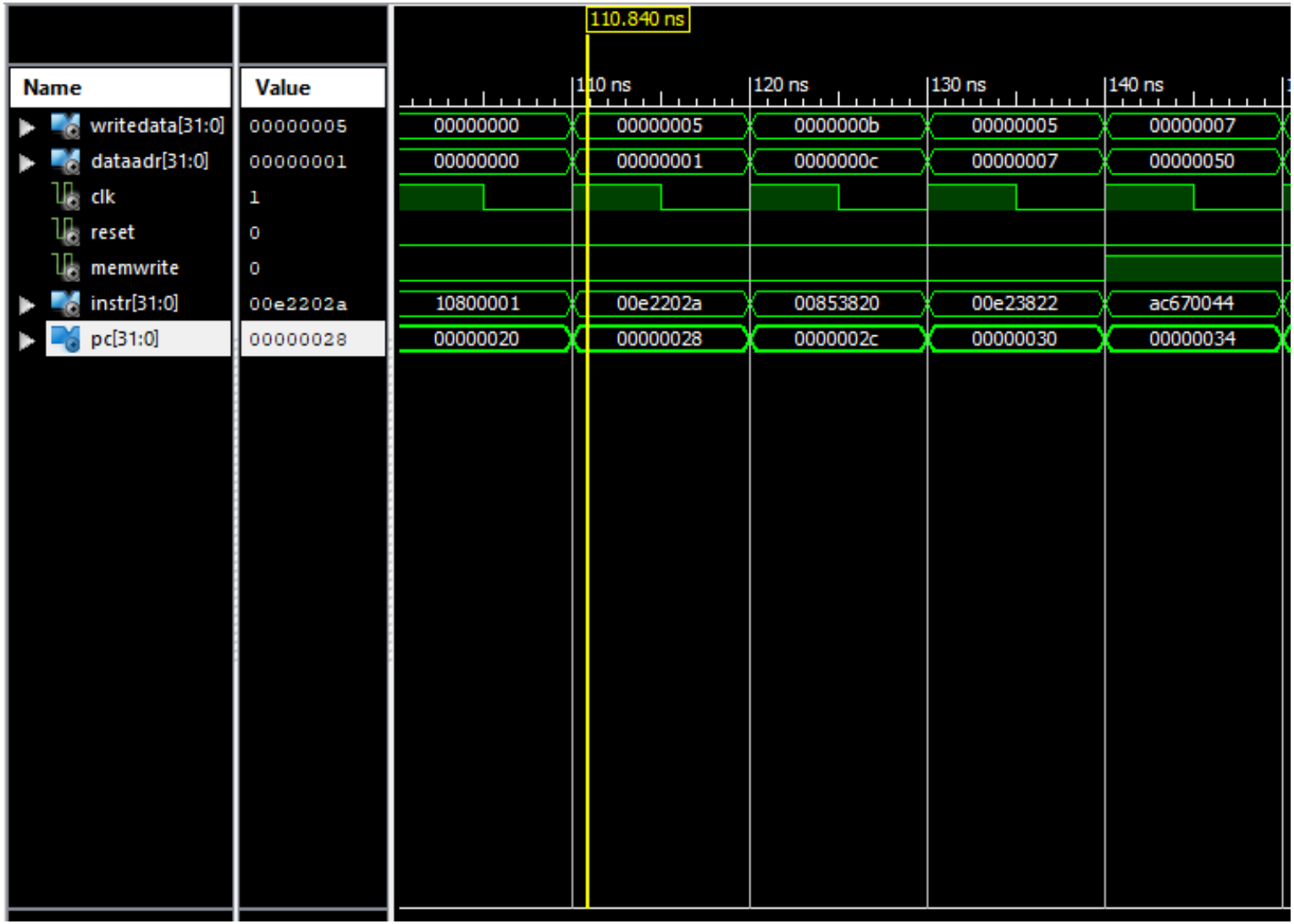
HANDS-ON1: TOP MODULE SIM. RESULT (INSTRUCTIONS 1-5: 1ST 5 CYCLES)



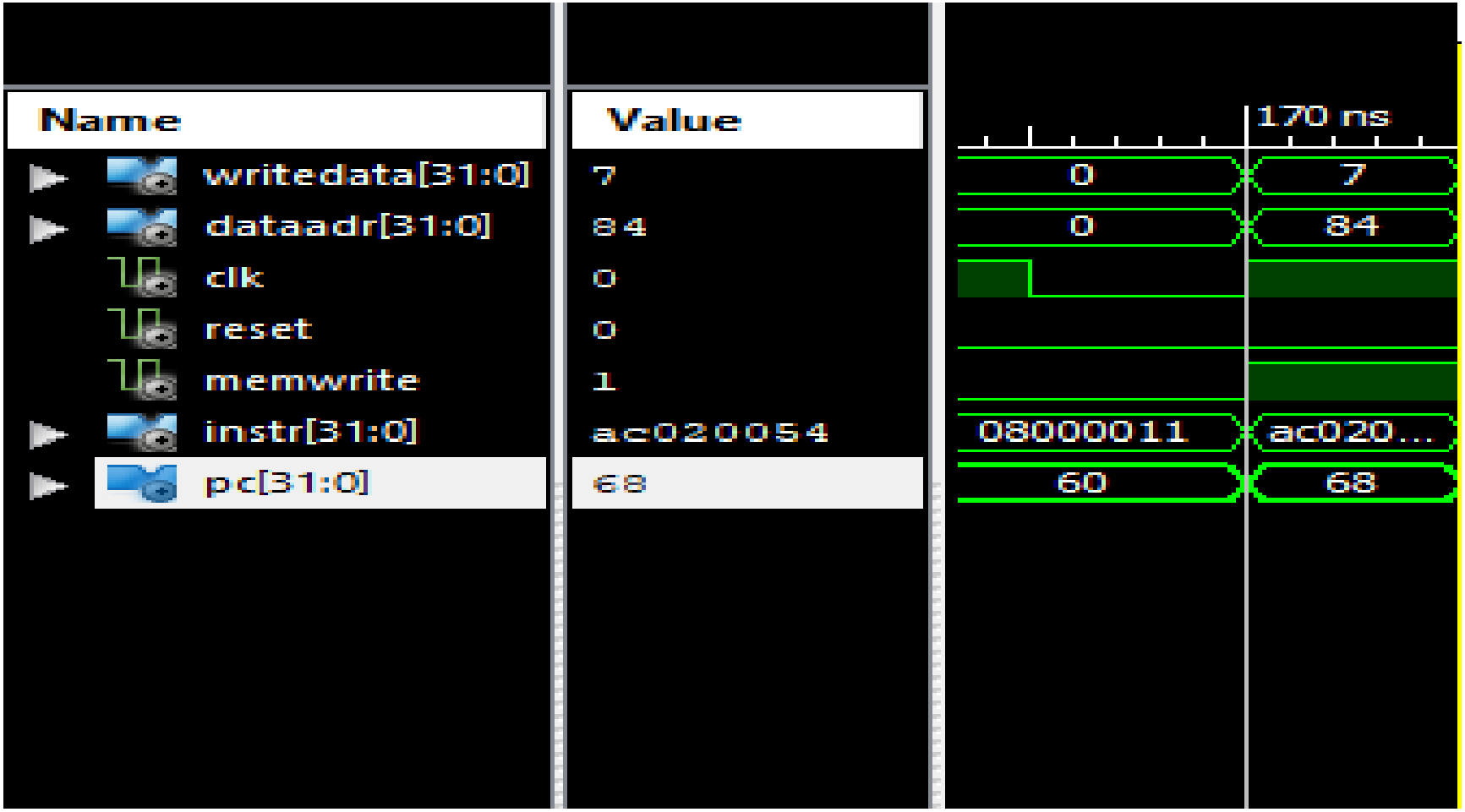
HANDS-ON1: TOP MODULE SIM. RESULT (INSTRUCTIONS 6-10: 2ND 5 CYCLES)



HANDS-ON1: TOP MODULE SIM. RESULT (INSTRUCTIONS 11-15: 3RD 5 CYCLES)



HANDS-ON1: TOP MODULE SIM. RESULT (LAST INSTRUCTION: CYCLE # 16)



Thanks