

Purpose	To apply our understanding of digital circuit synthesis using VHDL to a synthesis of our own CPU
Files Required	Summary of VHDL syntax you need can be found here.
ILO Contribution	LO 5
Send to Me	nix
Homework	Read chapter 14

Section A. Synthesizing and Simulating the CPU2S Components


For each of the following components you will

- Complete the component VHDL code template.
- Check your component code using the testbench provided.

Please note, words in **bold** are the names of signals in each Component.

How to select the correct testbench. Open up Simulation Sources and you'll get a list like this,



Say you want to use **mux_tb**. Right click this line and then choose  **Set as Top** from the dialogue box.

1. The MUX

[HINT] The code in chapter 13 page 6 will help here.
See also Chapter 14.6.4

The MUX output **op** should be **inA** when **sel** is high otherwise, it should be **inB**.

*Note: You can't use a conventional if-then-else construct since we are not in a **process** block.*

2. Registers

[HINT] The code in chapter 13 page 11 will help here.
See also Chapter 14.6.1.

The Register **data_out** should be **data_in** if:
(i) there is a rising **clk** edge and if
(ii) **load** is high,

3. Input Port

[HINT] The code in chapter 13 page 6 will help here.
See also Chapter 14.6.5.

The Port output **data_out** should be **data_in** when **load** is high.
In all other cases it should be low.

Here we introduce some new syntax. Here it is abstractly.

```
out <= in when (condition)
      else (others => '0')
```

4. The ALU

[HINT] The code in chapter 13 page 6 will help here.
See also Chapter 14.6.2.

Here we need a chain of 'when-else's *We can't use a conventional if-then-else construct since we are not in a process block.*

Here's a table of what the **op** will be according to the **sel_ALUf** signal which directs the ALU what to do:

sel_ALUf =	op
"001"	inB
"010"	inB + 1
"011"	inA + inB
"100"	inA - inB

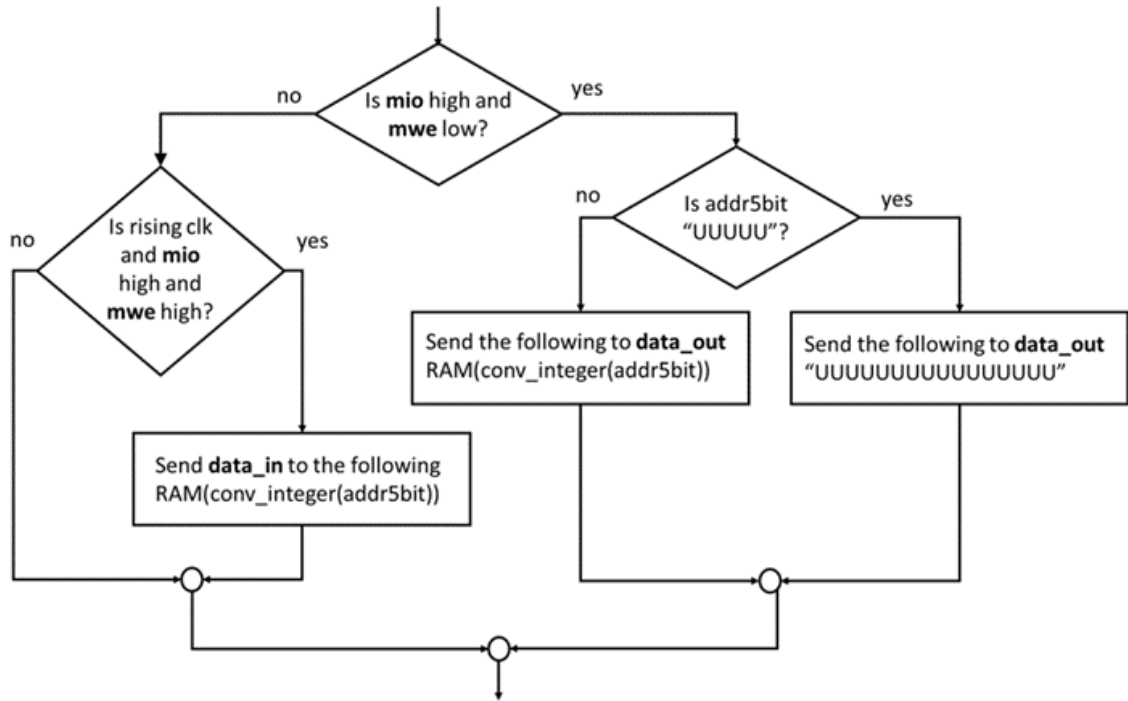
after your final else you should probably include the following to catch any user programming errors:

```
else (others => '0');
```

5. Memory – RAM

This needs thought, concentration, a southerly wind, and faith.
[HINT1] The code in chapter 13 page 14 will help, but this is certainly not the whole story.
See also Chapter 14.6.3.

The solution is best expressed through this flow diagram,



Section B. Programming our CPU2S

This will use the **NoC_CPU2S** file where all the individual components have been assembled and connected. You will only need to change the **compRAM** component.

1. A First Program

Let's investigate a program to input a number (could come from switches), then add a number from memory, then output the number (could go to LEDs). Here's the program

IN	8000	<i>Input into ACC</i>
ADD 11	2011	<i>Add data from memory address 11 (hex) to the ACC</i>
OUT	9000	<i>Send data from ACC to the output port</i>
HLT	7000	<i>Halt processing.</i>

The left shows the mnemonics, the right shows the opcodes you must type into RAM code-segment. Open the **ISE_RAM.vhd** design source, and you'll see the code-segment

```
27 |      -- code segment
28 |      X"0000",
29 |      X"0011",
30 |      X"0000",
31 |      X"0000",
32 |      X"0000",
33 |      X"0000",
34 |      "0000"
```

(a) Replace the first 4 rows with the opcodes for the program. Do not add or delete rows.

You can find the data segment further down; it looks like this.

```
44 |      -- data segment
45 |      X"0002", X"0003", X"0004", X"0005",
46 |      X"0001", X"0001", X"0002", X"0006",
47 |      X"0000", X"0000", X"0000", X"0000",
48 |      X"0000", X"0000", X"0000", X"0000"
```

So your instruction ADD 11 will get the data at address 11 (hex) which is 17 (dec). That's the "0003" on line 45.

(b) Synthesize your design.

(c) Run the testbench **NoC_CPU2S_tb_mini** and you'll get a waveform. Your job is to interpret this. Remember the program does an input into ACC, then adds a number to ACC then outputs the ACC to the output port.

Here's some basic checks

- (i) Check the instruction pointer **IP** increments 0, 1, 2, 3, ...
- (ii) Check when each instruction is loaded into the instruction register **IR**
- (iii) Check when the state is EXEC (state = 1)

Now let's follow the data. You could refer to clock cycles – perhaps add numbers to your snip.

- (iv) Find out when the input data appears on **inPort** and when it changes
 - (v) Find out when the data is outputted from the input port – **oeInPort** (high or low ?)
 - (vi) Find out when this data appears in the accumulator **ACC**. You should revisit how the **ACC** works (it's a register) Chapter 14.6.1
 - (vii) Check that the addition has been done correctly
 - (iix) Find out when the sum appears in the accumulator **ACC**
 - (ix) Find out when the data appears at the output port, see Chapter 14.6.6
 - (x) Check that the **mio** signal is correct
-

2. Optional Programs

Here's the full instruction set. The 'aa' in the opcode column is an address in hex.

LDA addr	00aa	<i>load ACC with data from memory at address aa(hex)</i>
STO addr	10aa	<i>Store ACC into memory at address aa</i>
IN	8000	<i>Input into ACC</i>
ADD addr	20aa	<i>Add data from memory address 11 (hex) to the ACC</i>
OUT	9000	<i>Send data from ACC to the output port</i>
JMP addr	40aa	<i>Set IP to code segment address aa (i.e. jump execution there)</i>
HLT	7000	<i>Halt processing.</i>

You might want to investigate the following and work out what they do. Note Program 1 does not halt!

Program 1	IN
	ADD 19 (hex)
	JMP 1

Program 2	LDA 10
	ADD 11
	STO 12
	OUT
	HLT
