CSU22022 Computer Architecture I

Prof. Michael Manzke

Processor Assignment: Full Project

Simulation Procedure

27th November 2022

Version 1.2

Simulation Procedure for the following entities:

Number of Instantiations	Instantiation Name	Entity Name
Processor		
1		CPU_Processor_XXXXXXXX
Datapath		
1	Datapath	DP_Datapath_XXXXXXXX
Register File		
1	RegFile	RF_RegisterFile_32_15_XXXXXXXX
1	DestReg_Decoder	RF_DestReg_Decoder_XXXXXXXX
1	DestTempReg_Decoder	RF_TempDestReg_Decoder_XXXXXXXX
32 15 1	RegisterXX, XX=00 to 31 TempRegXX, XX=01 to 15 PC *	RF_Register32Bit_XXXXXXXX
1 1	Mux32_A Mux32_B	RF_Mux32_32Bit_XXXXXXXX
1 1	Mux16_A Mux16_B	RF_Mux16_32Bit_XXXXXXXX
		* used in processor

Number of Instantiations		
	Instantiation Name	**Entity Name**
	Functional unit	
1	FunctionalUnit	DP_FunctionalUnit_XXXXXXXX
1	ALU	DP_ArithmeticLogicUnit_XXXXXXXX
32	BITXX, XX=00 to 31	DP_FullAdder_XXXXXXXX
1 1	Adder Adder *	DP_RippleCarryAdder32Bit_XXXXXXXX
32	BITXX, XX=00 to 31	DP_SingleBit_B_Logic_XXXXXXXX
1	BLogic	DP_32Bit_B_Logic_XXXXXXXX
32	BITXX, XX=00 to 31	DP_SingleBit_LogicCircuit_XXXXXXXX
1	LogicCircuit	DP_32Bit_LogicCircuit_XXXXXXXX
32	BITXX, XX=00 to 31	DP_Mux3_1Bit_XXXXXXXX
1	CFlagMux	DP_ShifterCFlagMux2_1Bit_XXXXXXXX
1	Shifter	DP_Shifter_XXXXXXXX
1	C_Flag	DP_CFlagMux2_1Bit_XXXXXXXX
1	Z_Flag	DP_ZeroDetection_XXXXXXXX
1 1 1 1 1 1 1	MuxB MuxD MuxF ALUMux PL_PI_Mux * ResetMux * MuxM *	CPU_Mux2_32Bit_XXXXXXXX
		* used in processor

Number of Instantiations		
	Instantiation Name	**Entity Name**
	Processor	
1	RAM	CPU_RAM_XXXXXXXX
1	ControlROM	CPU_ControlMemory_XXXXXXXX
1	InstReg	CPU_IR_XXXXXXXX
1	ZeroFill	CPU_ZeroFill_XXXXXXXX
1	PC	CPU_PC_XXXXXXXX
1	SignExt	CPU_SignExtend_XXXXXXXX
1	MuxS	CPU_SMux_XXXXXXXX
1	MuxC	CPU_Mux2_17Bit_XXXXXXXX
1	CAR	CPU_CAR_XXXXXXXX
1 1 1 1	CFlag VFlag NFlag ZFlag	CPU_DFlipFlop_XXXXXXXX
1	StatReg	CPU_StatusRegister_XXXXXXXX

Number of Instantiations	Instantiation Name	Entity Name
Processor Tests		
1		CPU_Processor_Test01_XXXXXXXX
1		CPU_Processor_Test02_XXXXXXXX
1		CPU_Processor_Test03_XXXXXXXX
1		CPU_Processor_Test04_XXXXXXXX

You must provide the following for every entity:

1. Design code e.g., RF_RegisterFile_32_15_XXXXXXXX.vhd
2. A test bench e.g., RF_RegisterFile_32_15_XXXXXXXX_TB.vhd
3. One or more schematics e.g., RF_RegisterFile_32_15_XXXXXXXX_SchematicXX.jpg
4. As many as needed annotated e.g., timing diagrams RF_RegisterFile_32_15_XXXXXXXX_TDXX.jpg
5. Simulation Procedure documentation RF_RegisterFile_32_15_XXXXXXXX_Doc.pdf

You must read "CSU22022 Processor Project 2022-2023 Instruction V1.0.pdf"

Furthermore, you must build a CPU_XXXXXXXX Vivado project and keep a copy of the project. All simulations must be executed in this Vivado project. Your submitted code must be able to reproduce your schematics and waveform diagrams in Vivado.

[01] RF_Mux16_32Bit_XXXXXXXX

Convert your student ID into binary and set:

IN00 = student ID
IN01 = student ID + 1
IN02 = student ID + 2
continue for all inputs

Demonstrate that the multiplexer functions correctly.

[02] RF_Mux32_32Bit_XXXXXXXX

Convert your student ID into binary and set:

IN00 = student ID
IN01 = student ID + 1
IN02 = student ID + 2
continue for all inputs

Demonstrate that the multiplexer functions correctly.

[03] RF_Register32Bit_XXXXXXXX

Convert your student ID into binary.
Write and read the register with your ID.

Demonstrate that the register functions correctly.

[04] RF_DestReg_Decoder_XXXXXXXX

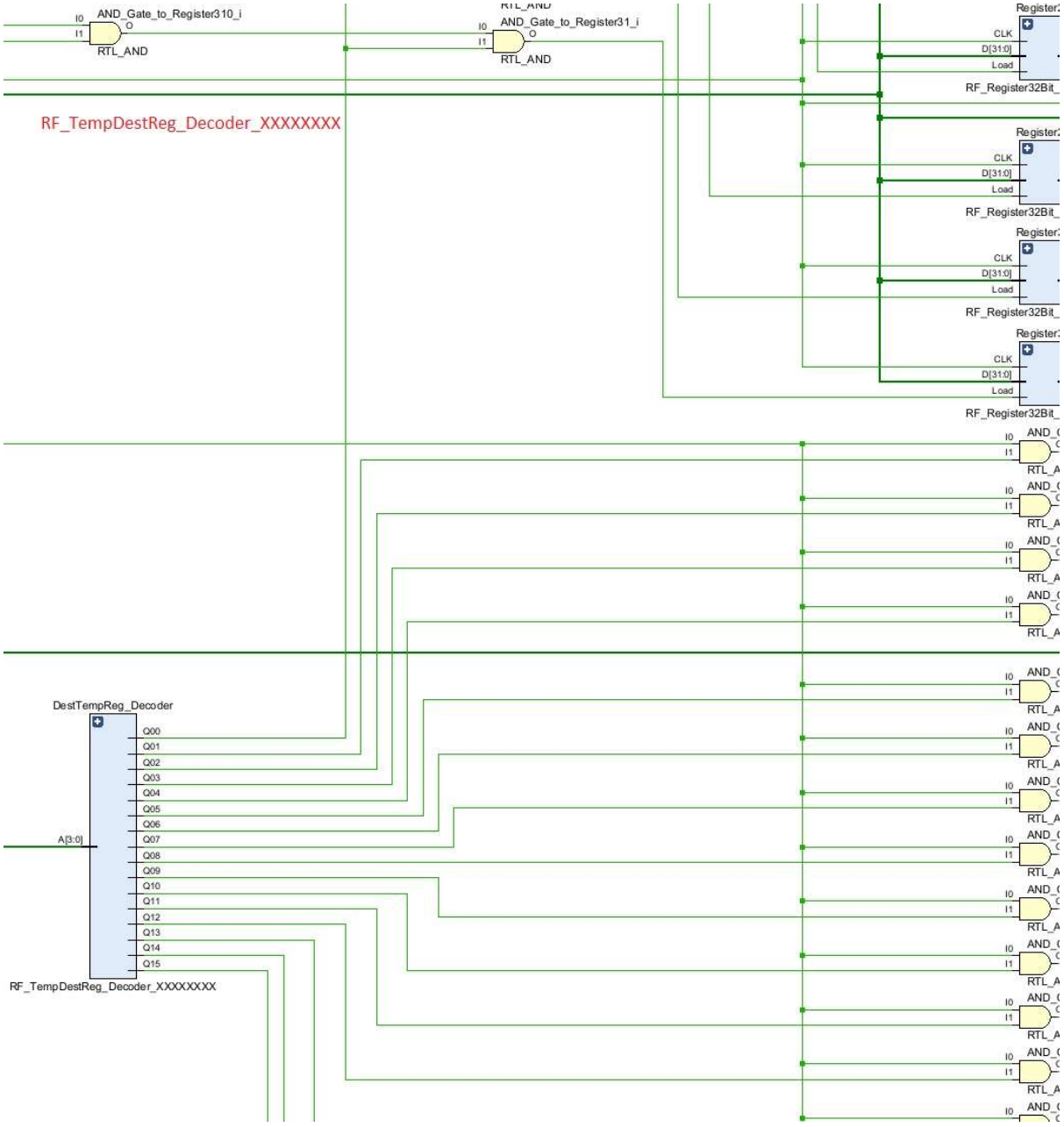
Demonstrate that the decoder functions correctly.

[05] RF_TempDestReg_Decoder_XXXXXXXX

Demonstrate that the decoder functions correctly.

[06] RF_RegisterFile_32_15_XXXXXXXX

In addition to an overview schematic, you must provide zoom in screenshots that allow me to read the entity names. The following screenshots is an example:



Convert your student ID into binary and set:

Load registers as follows:

Register00 = student ID
Register01 = student ID + 1
Register02 = student ID + 2
continue for all Registers (32+15) including the Temp Registers.

Read all registers and confirm that the functions correctly.

[07] DP_FullAdder_XXXXXXXX

Show that your full adder implements the correct truth table for a full adder.

[08] DP_RippleCarryAdder32Bit_XXXXXXXX

All numbers are in 2's complement.

Adapt the 3BitParrallelAdder test procedure to this 32Bit Ripple Carry Adder.

Demonstrate worst case propagation delay. Also how long is the propagation delay?

If you add a 2's complement number to your StudentID, what number would set the C flag and V flag?

[09] DP_SingleBit_B_Logic_XXXXXXXX

Show that the logic works.

[10] DP_32Bit_B_Logic_XXXXXXXX

Simulation your StudentID as input to the B_Logic and demonstrate that you can output all 0's, your StudentID, 1's complement, and all 1's by changing the select signals S_0 and S_1.

[11] DP_SingleBit_LogicCircuit_XXXXXXXX

Demonstrate that our code can perform the following bitwise operations: AND, OR, XOR, NOT.

[12] DP_32Bit_LogicCircuit_XXXXXXXX

Your simulation should provide your StudentID on the "A" 32-bit input vector. You should also select a suitable value for the "B" 32-bit input vector.

Demonstrate that by changing the signals S_0, and S_1, you generate the following bitwise operations: AND, OR, XOR, NOT.

[13] CPU_Mux2_32Bit_XXXXXXXX

Show that the multiplexer works.

[14] DP_ArithmeticLogicUnit_XXXXXXXX

Your simulation should provide your StudentID on the "A" 32-bit input vector. You should also select a suitable value for the "B" 32-bit input vector.

Demonstrate that by changing the signals S_0, S_1, S_2, and C_{in} you generate the following outputs:

A, A + 1, A + B, A + B + 1, A + 1's complement(B), A + 1's complement(B) + 1, A – 1, A, A AND B, A OR B, A XOR B, and NOT A.

[15] DP_Mux3_1Bit_XXXXXXXX

Show that the multiplexer works.

[16] DP_ShifterCFlagMux2_1Bit_XXXXXXXX

Show that the logic works.

[17] DP_Shifter_XXXXXXXX

Your simulation should provide your StudentID as 32-bit input vector. Demonstrate that you can shift your StudentID by one bit to the right or one bit to the left or leave it unchanged by changing the signals S_1 and S_2.

Furthermore, demonstrate that your right shift and left shift operations set and unset the C-Flag.

[18] DP_CFlagMux2_1Bit_XXXXXXXX

Show that the logic works.

[19] DP_ZeroDetection_XXXXXXXX

Show that the logic works.

[20] DP_FunctionalUnit_XXXXXXXX

The order in with you must demonstrate the various operations of your Functional-Unit is determined by the last digit of your student number (ID). Please see the following table on the next page for details.

The simulation timing diagram must show these operations in the correct order and on a single screenshot or two or more overlapping screenshot.

Furthermore, your testbench should provide your StudentID as 32-bit input vector on the "A" input of your Function Unit and your StudentID plus the last digit of your StudentID on the "B" input of your Functional Unit.

		Last Digit of your Student Number (ID)									
		0	1	2	3	4	5	6	7	8	9
Order of micro-operations	1st	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B	F=1's c A
	2nd	F=sIB	F=A OR B	F=A XOR B	F=A OR B	F=1's c A	F=A (FS=00111)	F=A OR B	F=A XOR B	F=1's c A	F=A+1
	3rd	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B
	4th	F=A AND B	F=srB	F=A AND B	F=A XOR B	F=A (FS=00111)	F=A XOR B	F=srB	F=B	F=A+1	F=A (FS=00000)
	5th	F=A+B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B
	6th	F=B	F=1's c A	F=B	F=sIB	F=srB	F=B	F=1's c A	F=sIB	F=A (FS=00111)	F=sIB
	7th	F=A+B+1	F=A+1's c B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B
	8th	F=srB	F=sIB	F=1's c A	F=B	F=sIB	F=srB	F=B	F=A (FS=00111)	F=srB	F=B
	9th	F=A+1's c B	F=A+B+1	F=A+1's c B+1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1
	10th	F=1's c A	F=B	F=srB	F=A+1's c B+1	F=B	F=1's c A	F=sIB	F=srB	F=B	F=A (FS=00111)
	11th	F=A (FS=00111)	F=A+1's c B+1	F=A+1's c B	F=A-1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1
	12th	F=A OR B	F=A XOR B	F=sIB	F=1's c A	F=A XOR B	F=sIB	F=A (FS=00111)	F=1's c A	F=sIB	F=srB
	13th	F=A-1	F=A (FS=00111)	F=A-1	F=A AND B	F=A OR B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B
	14th	F=A+1's c B+1	F=A AND B	F=A (FS=00111)	F=srB	F=A-1	F=A AND B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1
	15th	F=A XOR B	F=A-1	F=A OR B	F=A (FS=00111)	F=A AND B	F=A OR B	F=A XOR B	F=A+1	F=A (FS=00000)	F=A+B

[21] DP_Datapath_XXXXXXXX

Please provide a clock signal for the registers in your register-file. The clock signal must be appropriate for the worst-case propagation delay of your Function Unit.

Load your StudentID into the first of the 32 registers. Then load your StudentID - 1 into the 2nd register, your StudentID - 2 into the 3rd register ... please continue until all 32 registers and the 15 TempReg have a value.

The load operation should be via “Data in” port and “MUX D”.

Once all 32 + 15 registers are loaded, use the last digit of your student number to select the destination-register (D address).

Use the (last digit of your student number + 5) to select the source-register A (A address).

Use the (last digit of your student number + 15) to select the source-register B (B address).

The order in with you must demonstrate the various operations of your Datapath is determined by the last digit of your student number (ID). Please see the table on the previous page for details. The same order as for the simulation of the Function Unit.

Maintain the current configuration but switch the “MUX B” to “Constant in”. Your simulation should provide your StudentID on the “Constant in” port. Execute the various operations of your Datapath in the order determined by the last digit of your student number (ID) but only those 10 operations that require data on the “B” input of your Function Unit.

[22] CPU_RAM_XXXXXXXX

Initialise all 128 memory locations starting with the last two digits of your StudentID at memory address 0 and increment this number by one for all the remaining memory locations e.g. 11, 12, 13 …

Read all 128 memory addresses.

Overwrite 32 memory locations starting at the address of the last digit of your StudentID.

Demonstrate that this doesn’t work if the MW signal is unset.

[23] CPU_ControlMemory_XXXXXXXX

Initialise all 128 memory locations starting with the last two digit of your StudentID at memory address 0 and increment this number by one for all the remaining memory locations e.g. 11, 12, 13 …

Read all 128 memory addresses.

[24] CPU_IR_XXXXXXXX

Demonstrate the correct function of your Instruction Register IR by loading the register with the following values:

- DR = Digit 0 of your StudentID
- SA = Digit 1 of your StudentID
- SB = Digit 2 of your StudentID
- Opcode = Digit 4 and 3 of your StudentID

[25] CPU_ZeroFill_XXXXXXXX

Show that the logic works.

[26] CPU_PC_XXXXXXXX

Demonstrate the correct function of your Program Counter PC by:

- Resetting the PC to the last digit of your StudentID
- Incrementing the PC
- Adding the last two digits of your StudentID as a displacement to the current value of your PC

[27] CPU_SignExtend_XXXXXXXX

Show that the logic works.

[28] CPU_SMux_XXXXXXXX

Show that the logic works.

[29] CPU_Mux2_17Bit_XXXXXXXX

Show that the logic works.

[30] CPU_CAR_XXXXXXXX

Demonstrate the correct function of your Control Address Register CAR by:

- Resetting the CAR to the last digit of your StudentID
- Incrementing the CAR
- Loading the CAR with the last two digits of your StudentID

[31] CPU_DFlipFlop_XXXXXXXX

Show that the logic works.

[32] CPU_StatusRegister_XXXXXXXX

Demonstrate the correct function of your Status Register by:

- Loading C, V, N, and Z flags into the Status Register
- Resetting the C-flag
- Resetting the V-flag
- Resetting the N-flag
- Resetting the Z-flag

[33] CPU_Processor_Test01_XXXXXXXX

Please provide a clock signal for the registers in your register-file and the RAM. The clock signal must be appropriate for the worst-case propagation delay of your Function Unit.

Initialise all 128 memory locations starting with the last two digits of your StudentID at memory address 0 and increment this number by one for all the remaining memory locations e.g., 11, 12, 13 … (This was requested in "[22] CPU_RAM_XXXXXXXX")

Load the first register in your register-file with the content of the first memory location in your RAM, the second register in your register-file with the content of the second memory location in your RAM ... please continue until all 32 registers and the 15 TempReg have a value. Your simulation should provide the correct RAM address through the InstAddresss[31:0] vector.

Once all 32 + 15 registers are loaded, use the last digit of your student number to select the source-register A (A address) and the last digit of your student number + 1 to select the source-register B (B address).

Use the selected registers on the Port A and Port B to demonstrate all 15 micro-operations of the Datapath. The order in with you must demonstrate the various operations of your Datapath is determined by the last digit of your student number (ID). Please see the table on the next page for details. The same order as for the simulation of the Function Unit and the Datapath.

You must use the last digit of your student number + 2 to select the destination-register (D address). The result of your first micro-operation must be written in that destination-register. Increment the destination-register address by 1 before execution the next micro-operation. Continue until you have executed all 15 micro-operations.

Store the content of these 15 destination-registers in 15 consecutive memory locations in your RAM.

Then, load these 15 consecutive memory locations from your RAM into destination-registers 1 by overwriting the content of destination-registers 1.

Order of micro-operations	Last Digit of your Student Number (ID)									
	0	1	2	3	4	5	6	7	8	9
1st	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B	F=1's c A
2nd	F=sIB	F=A OR B	F=A XOR B	F=A OR B	F=1's c A	F=A (FS=00111)	F=A OR B	F=A XOR B	F=1's c A	F=A+1
3rd	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B
4th	F=A AND B	F=srB	F=A AND B	F=A XOR B	F=A (FS=00111)	F=A XOR B	F=srB	F=B	F=A+1	F=A (FS=00000)
5th	F=A+B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B
6th	F=B	F=1's c A	F=B	F=sIB	F=srB	F=B	F=1's c A	F=sIB	F=A (FS=00111)	F=sIB
7th	F=A+B+1	F=A+1's c B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B
8th	F=srB	F=sIB	F=1's c A	F=B	F=sIB	F=srB	F=B	F=A (FS=00111)	F=srB	F=B
9th	F=A+1's c B	F=A+B+1	F=A+1's c B+1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1
10th	F=1's c A	F=B	F=srB	F=A+1's c B+1	F=B	F=1's c A	F=sIB	F=srB	F=B	F=A (FS=00111)
11th	F=A (FS=00111)	F=A+1's c B+1	F=A+1's c B	F=A-1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1
12th	F=A OR B	F=A XOR B	F=sIB	F=1's c A	F=A XOR B	F=sIB	F=A (FS=00111)	F=1's c A	F=sIB	F=srB
13th	F=A-1	F=A (FS=00111)	F=A-1	F=A AND B	F=A OR B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B
14th	F=A+1's c B+1	F=A AND B	F=A (FS=00111)	F=srB	F=A-1	F=A AND B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1
15th	F=A XOR B	F=A-1	F=A OR B	F=A (FS=00111)	F=A AND B	F=A OR B	F=A XOR B	F=A+1	F=A (FS=00000)	F=A+B

[34] CPU_Processor_Test02_XXXXXXXX

Demonstrate the correct function of your Status Register by:

- Loading C, V, N, and Z flags into the Status Register
- Resetting the C-flag
- Resetting the V-flag
- Resetting the N-flag
- Resetting the Z-flag

The C, V, N, and Z flags must be generated in the Datapath by choosing suitable data and micro-operation. The data must be loaded from your RAM.

[35] CPU_Processor_Test03_XXXXXXXX

Initialise the RAM with the following five machine code instructions:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.NUMERIC_STD.ALL;
25
26  entity CPU_RAM_XXXXXXXX is
27      Port ( Clock : in STD_LOGIC;
28             Address : in STD_LOGIC_VECTOR (31 downto 0);
29             DataIn : in STD_LOGIC_VECTOR (31 downto 0);
30             WriteEnable : in STD_LOGIC;
31             DataOut : out STD_LOGIC_VECTOR (31 downto 0));
32  end CPU_RAM_XXXXXXXX;
33
34  architecture Behavioral of CPU_RAM_XXXXXXXX is
35
36  -- we use the least significant 7 bit of the address
37  type RAM_array is array(0 to 127) of STD_LOGIC_VECTOR (31 downto 0);
38
39  signal RAM : RAM_array:=(
40  X"00000000",-- 00
41  X"00000001",-- 01
42  X"00000002",-- 02
43  X"00000003",-- 03
44  -- Machine code
45  -- example studentID 87654321
46  -- your machine code starts at digit 3 of your ID = 4
47  -- Opcode = digit 3 = 4
48  -- DR = digit 2 = 3
49  -- SA = digit 1 = 2
50  -- SB = digit 0 = 1
51  --|     Opcode       |  DR   |  SA   |  SB   |
52    "0000000000000100"&"00011"&"00010"&"00001",-- 04
53    "0000000000000101"&"00100"&"00011"&"00010",-- 05
54    "0000000000000110"&"00101"&"00100"&"00011",-- 06
55    "0000000000000111"&"00110"&"00101"&"00100",-- 07
56    "0000000000001000"&"00111"&"00110"&"00101",-- 08
57  X"00000009",-- 09
58  X"0000000A",-- 0A
59  X"0000000B",-- 0B
60  X"0000000C",-- 0C
61  X"0000000D",-- 0D
62  X"0000000E",-- 0E
63  X"0000000F",-- 0F
64
65  X"00000010",-- 10
```

The reset signal of your program counter (PC) must reset the counter to the first instruction in your memory. Demonstrate that you can fetch all 5 instructions by in incrementing the PC.

Furthermore, you must change the DR and SB value of your last instruction with the correct 2's complement number to jump back two your first address. Demonstrate that this works.

[36] CPU_Processor_Test04_XXXXXXXX

Initialise the Control Memory with all our 15 micro-operations according to the following table. Your student ID determines the order of these micro-operations. The control memory address for the first micro-operations are the last two digits of your Student ID.

The reset signal of your CAR should set the register to your first micro-operation. Demonstrate that you can go through all 15 operations by incrementing the CAR. At the last operation you should use the Next Address (NA) to return to the first operation.

		Last Digit of your Student Number (ID)									
		0	1	2	3	4	5	6	7	8	9
Order of micro-operations	1st	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B	F=1's c A
	2nd	F=sIB	F=A OR B	F=A XOR B	F=A OR B	F=1's c A	F=A (FS=00111)	F=A OR B	F=A XOR B	F=1's c A	F=A+1
	3rd	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B	F=A XOR B
	4th	F=A AND B	F=srB	F=A AND B	F=A XOR B	F=A (FS=00111)	F=A XOR B	F=srB	F=B	F=A+1	F=A (FS=00000)
	5th	F=A+B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B	F=A OR B
	6th	F=B	F=1's c A	F=B	F=sIB	F=srB	F=B	F=1's c A	F=sIB	F=A (FS=00111)	F=sIB
	7th	F=A+B+1	F=A+1's c B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1	F=A AND B
	8th	F=srB	F=sIB	F=1's c A	F=B	F=sIB	F=srB	F=B	F=A (FS=00111)	F=srB	F=B
	9th	F=A+1's c B	F=A+B+1	F=A+1's c B+1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1	F=A-1
	10th	F=1's c A	F=B	F=srB	F=A+1's c B+1	F=B	F=1's c A	F=sIB	F=srB	F=B	F=A (FS=00111)
	11th	F=A (FS=00111)	F=A+1's c B+1	F=A+1's c B	F=A-1	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B	F=A+1's c B+1
	12th	F=A OR B	F=A XOR B	F=sIB	F=1's c A	F=A XOR B	F=sIB	F=A (FS=00111)	F=1's c A	F=sIB	F=srB
	13th	F=A-1	F=A (FS=00111)	F=A-1	F=A AND B	F=A OR B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1	F=A+1's c B
	14th	F=A+1's c B+1	F=A AND B	F=A (FS=00111)	F=srB	F=A-1	F=A AND B	F=A+1	F=A (FS=00000)	F=A+B	F=A+B+1
	15th	F=A XOR B	F=A-1	F=A OR B	F=A (FS=00111)	F=A AND B	F=A OR B	F=A XOR B	F=A+1	F=A (FS=00000)	F=A+B

Please see a copy this table on previous pages for more detail.

[37] CPU_Processor_XXXXXXXX

Design an instruction set that allows for the execution of all 15 micro-operations. Furthermore, a load instruction, a store instruction and one branch instruction. You must provide an algorithmic state machine chart for this implementation.

Demonstrate that your processor can execute all these 18 instructions.

Also, write and execute a program that uses all these instructions.