

Laboratory 6

(Due date: **005**: April 10th, **006**: April 11th)

OBJECTIVES

- ✓ Design a 16-bit microprocessor with Single-Cycle Hardwired Control.
- ✓ Implement an Instruction Set Architecture (ISA).

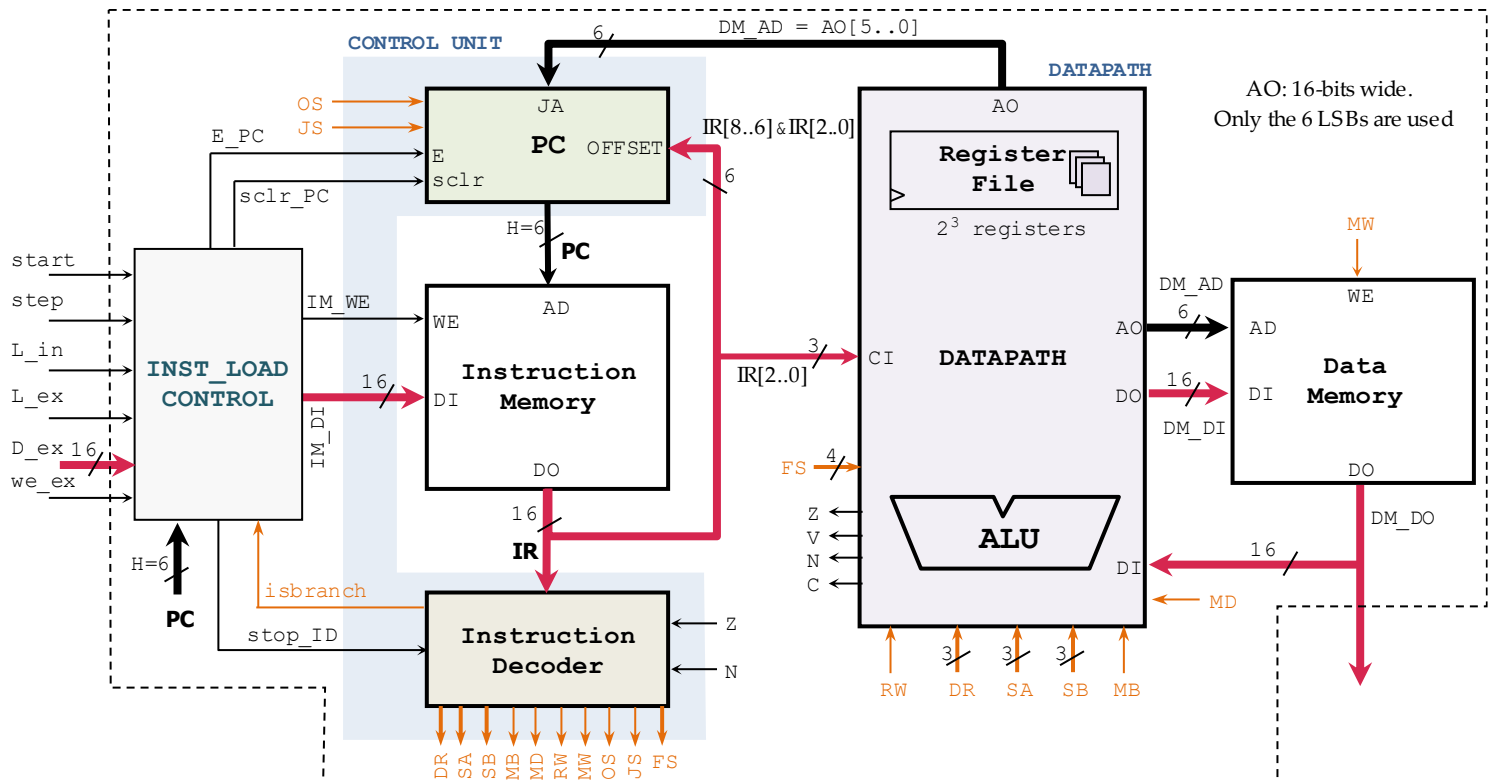
VHDL CODING

- ✓ Refer to the [Tutorial: VHDL for FPGAs](#) for parametric code for: Register, adder/subtractor.

FIRST ACTIVITY: 16-BIT MICROPROCESSOR – DESIGN AND SIMULATION (70/100)

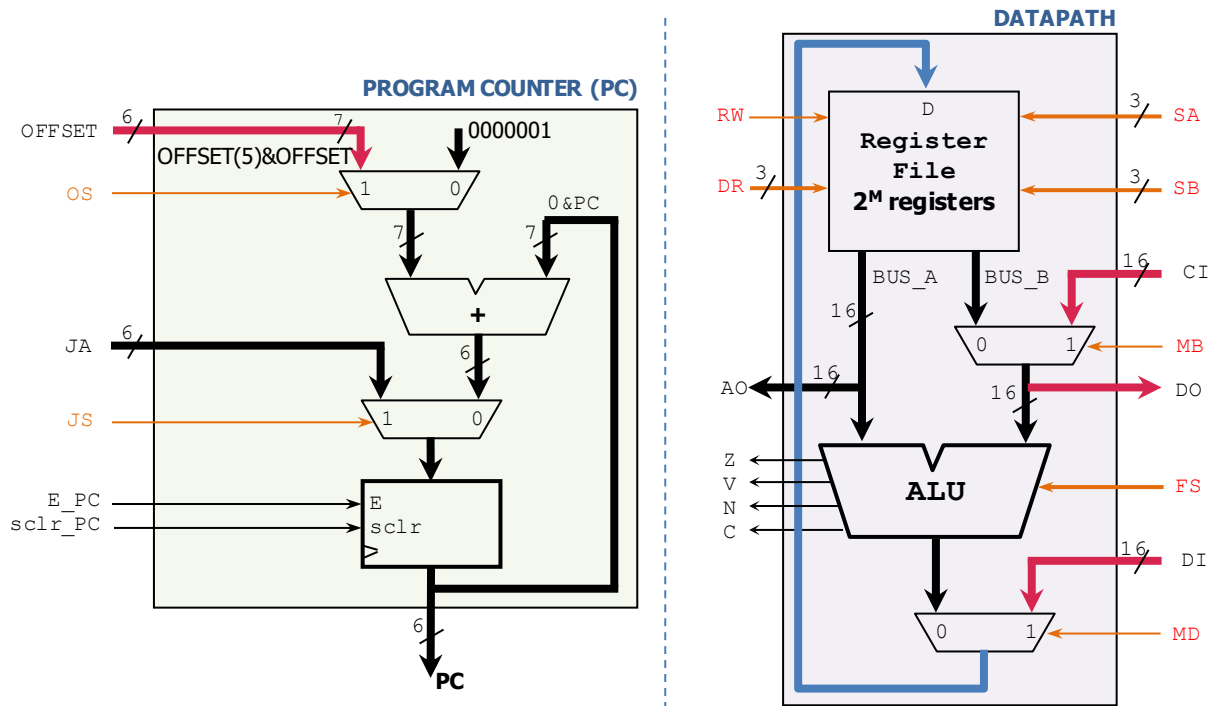
DESIGN PROBLEM

- Implement the **Simple Computer** (see Notes – Unit 6): uP with 6-bit IM/DM address, 16-bit instructions, and 16-bit data.



COMPONENTS:

- **DM, IM:** 64 words, 16 bits per word. Use the files `RAM_emul.vhd`, `my_rege.vhd`. (set the proper parameters).
- **Datapath:** (note that $CI[2..0] = IR[2..0]$, $CI[15..3] = "00...0"$)
 - ✓ Register File: 8 registers ($R_0 - R_7$) are included. See *Notes – Unit 6* for an example with 4 registers.
 - ✓ ALU: Use the files: `alu.vhd`, `alu_arith.vhd`, `alu_logic.vhd`, `super_addsub.vhd`, `fulladd.vhd`.
- **PC:** Note that **OFFSET** is a 6-bit signed number. The adder uses 7 bits, from which we only retrieve the 6 LSBs.
- **Instruction Decoder (ID):** This is a large combinational circuit. The outputs depend directly on the inputs.
 - ✓ The outputs are generated based on the instructions on **IR** (Instruction Register).
 - ✓ Instruction Set: For the list of instructions, refer to *Notes – Unit 6*. The Instruction Set does not include instructions that read the **V** and **C** bits. Thus, the **ID** does not consider these two bits.
 - ✓ **stop_ID**: If **stop_ID**=1, it forces the signals **RW**, **MW**, **OS**, **JS** to be '0'.
 - ✓ **isbranch**: If the instruction in **IR** is a branch or jump instruction, this signal is set to '1'.
- **Instruction Load Control:** This block is required in order to write instructions on the **IM**, and then to trigger program execution. Use the file `instload_ctrl.vhd` (use parameters $H=6$, $N=16$) This circuit is a FSM that works as follows:
 - ✓ To store instructions on **IM** from an external port: assert **L_ex** and then use the inputs **D_ex** and **we_ex**.
 - ✓ To store instructions on **IM** using pre-stored hardwired data: assert **L_in**.
 - ✓ Once instructions are written on the **IM**, program execution is started by asserting **start** for a clock cycle. The **step** signal controls whether to enable program execution (**step**=1) or disable it (**step**=0).

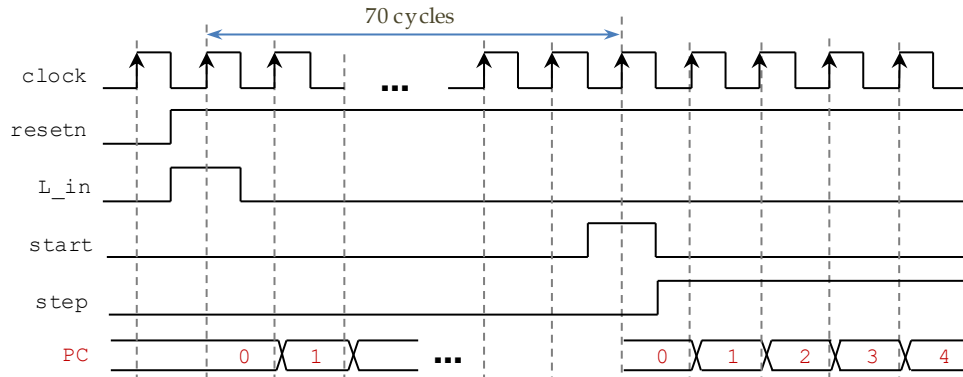


PROCEDURE

- Create a new Vivado project. Select the corresponding Artix-7 FPGA (e.g.: XC7A50T-1CSG324 for the Nexys A7-50T).
- Write the VHDL code for the given circuit. Synthesize your circuit to clear syntax errors and most warnings.
 - Note: The code for the ALU, IM, DM, and Instruction Load Control blocks is already provided. You need to instantiate these components and set up the corresponding generic parameters.
- Write the VHDL testbench to simulate your circuit.
 - Your testbench must test the following Assembly program (use a 50 MHz input clock with 50% duty cycle).
 - Assembly program (pre-stored in instload_ctrl.vhd). It stores numbers from 43 down to 29 in Data Memory (DM) on addresses 0 to 14. The number to be stored appears in R6. The program completes when BRZ R4, -7 makes PC=0.

Address	VHDL code snippet	Assembly Program		address	DM
000000	CD(0) <= "1001100010---101"	start: LDI R2,5	R2 ← 5	000000	2B
000001	CD(1) <= "1001100110---111"	LDI R6,7	R6 ← 7	000001	2A
000010	CD(2) <= "1000010110110111"	ADI R6,R6, 7	R6 ← 14	000010	29
000011	CD(3) <= "0000000100110---"	MOVA R4,R6	R4 ← 14	000011	28
000100	CD(4) <= "0000010110100110"	ADD R6,R4,R6	R6 ← 28	000100	27
000101	CD(5) <= "0000001110110---"	loop: INC R6,R6	R6 ← R6+1	000101	26
000110	CD(6) <= "0100000---100110"	ST R4,R6	M[R4] ← R6	000110	25
000111	CD(7) <= "1100000111100001"	BRZ R4, -7	If R4=0 ⇒ PC ← PC-7=0	000111	24
001000	CD(8) <= "0000110100100---"	DEC R4,R4	R4 ← R4-1	001000	23
001001	CD(9) <= "1110000---010---"	JMP R2	PC ← R2=5	001001	22
001010	...		(NOP operation)	001010	21
...				001011	20
				001100	1F
				001101	1E
				001110	1D

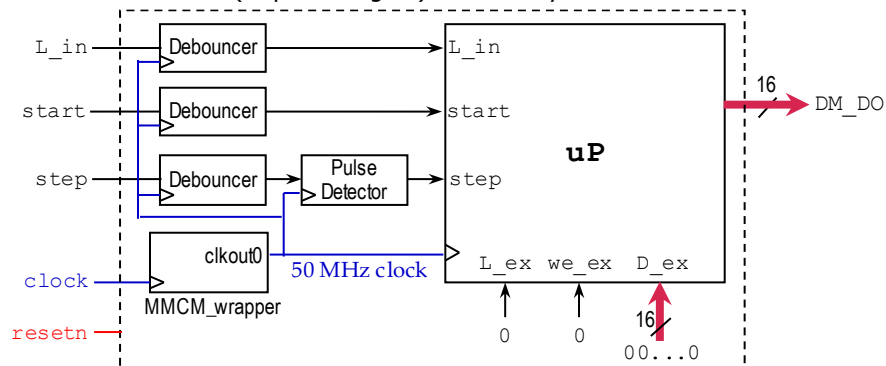
- Set L_in=1 for a clock cycle. Then wait 70 cycles for the program to be written on the Instruction Memory.
 - Since they are not being used, set the inputs L_ex, we_ex, and D_ex to 0's.
- Set start=1 for a clock cycle. Make sure that step = 1 during the execution of the program (for as many cycles as needed)



- Perform Behavioral Simulation of your design. **Demonstrate this to the TA.**
 - Add internal signals to the waveform view. In particular: PC, IR, R0-R7, ID outputs, DM registers.
 - To verify the correct processing of instructions, look at PC and IR. Then, observe the R0-R7 values as well as other signals (e.g.: ID outputs). To verify that the correct data was stored on DM (Data Memory), you can add the Individual Registers (from 0 to 14) of DM to the waveform view.

SECOND ACTIVITY: TESTING (30/100)

- In order to properly test this microprocessor, we need to:
 - Set the inputs L_ex, we_ex, and D_ex to 0's.
 - Avoid mechanical bouncing on the push buttons for L_in, start, and step. Connect the debouncer circuit (mydebouncer.vhd, my_genpulse_sclr.vhd) on these inputs.
 - Ensure that each pressing of step is converted to a one-cycle pulse. Connect a pulse detector (mypulse_det.vhd) to the debounced step signal. This way one instruction is executed each time step is pressed.
 - When step=0, the instload_control block issues stop_ID=1. This causes the program execution to pause.
 - Reduce the frequency of operation to 50 MHz. Add a MMCM block with a 50 MHz output clock (MMCM_wrapper.vhd, use O_0=2 for clockout0 = 50MHz). Then use the 50 MHz clock as the system clock.
 - Due to the large combinational delay, the design cannot meet the timing constraint of the input clock (100 MHz). As a result, we use a Digital Clock Manager (MMCM) that generates a 50 MHz clock (this timing constraint can be met).
- Create a top file with the modifications (as per the figure). Note that you do not need to simulate this circuit.



- I/O Assignment: Create the XDC file associated with your board.
 - Suggestion (Nexys A7-50T/A7-100T, Nexys 4/DDR):

Board pin names	CLK100MHZ	CPU_RESET	BTNU	BTNL	BTNC	LED15-LED0
Signal names in code	clock	resetn	L_in	start	step	DM_DO
 - Note: synchronous circuits always require a clock and reset signal.
 - Reset signal:** As a convention in this class, we use active-low reset (resetn). Thus, we tie resetn to the active-low push button CPU_RESET of the Nexys A7-50T/A7-100T, Nexys 4/DDR board.
 - Clock signal:** Like other signals in the XDC file, uncomment the lines associated with the clock signal and replace the signal label with the name used in your code. In addition, there is parameter -period that is set by default to 10.00. This is the period (in ns) that your circuit should support.
 - Nexys A7-50T: In these lines, replace the label CLK100MHZ with the signal name you use in your code (clock):

```
set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
```
- Generate and download the bitstream on the FPGA and test the Assembly Program. **Demonstrate this to your TA.**
 - To test the Assembly Program, follow these steps:
 - Push and release L_in.
 - Push and release start.
 - Push and release step. For every stroke, an instruction is executed. Do this repeatedly until the program completes its task (this happens when the instruction BRZ R4,-7 branches back to instruction at 000000).
 - The first time you execute ST R4,R6 (i.e., after 7 strokes of step) you should see 0x001D on the output DM_DO.
 - The second time you execute ST R4,R6, you should see 0x001E on the output DM_DO.
 - ...
 - The last time you execute ST R4,R6, you should see 0x002B on the output DM_DO.
 - Note:** after ST R4,R6 (or BRZ R4,-7) is executed, the R6 value appears on DM_DO. This is because these two instructions cause SA=4, which results in AO=R4[5..0]. R4[5..0]: DM address where the value of R6 is stored.

SUBMISSION

- Submit to Moodle (an assignment will be created):
 - ✓ This lab sheet (as a .pdf) completed (if applicable) and signed off by the TA (or instructor).
 - Note: *The lab assignment has two activities. You get full points of the 1st activity if you demo it by the due date. You can demo the 2nd activity by the due date or late (here, we apply a penalty towards the points of the 2nd activity).*
 - ✓ (As a .zip file) All the generated files: VHDL code, VHDL testbench, and XDC file. **DO NOT submit the whole Vivado Project.**
 - Your .zip file should only include one folder. Do not include subdirectories.
 - It is strongly recommended that all your design files, testbench, and constraints file be located in a single directory. This will allow for a smooth experience with Vivado.
 - You should only submit your source files AFTER you have demoed your work. Submission of work files without demoing will be assigned NO CREDIT.

TA signature: _____

Date: _____