PCIMControl

To simulate Program Counter (PC), Instruction Memory (IM) and Instruction Decoder (ID). A program counter (PC) is a CPU register in the computer processor which has the address of the next instruction to be executed from memory. Instruction Memory is read only memory which store instruction. The Instruction Decoder is responsible to convert the opcode 11 bits and give out the 9 control signals. Below are the steps involved:

a. Design Program counter unit using Program Counter and adder module.
b. Design Combine Program counter and Instruction Memory unit where PC will generate by Program counter Unit and instruction will be read from Instruction memory.
c. Design Combine Program Counter, Instruction Memory and Instruction Decoder where PC will generate by Program counter Unit and instruction will be read from Instruction memory and this instruction is decoded using Instruction decoder.

- Simulation of Program Counter Unit.

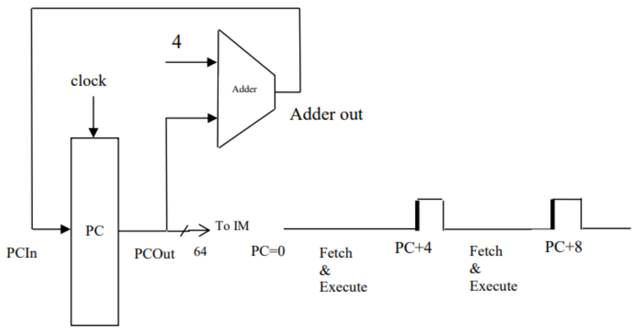


Figure 5.1 a) Program Counter and adder, b) Timing diagram indicating execution wrt the clock cycle

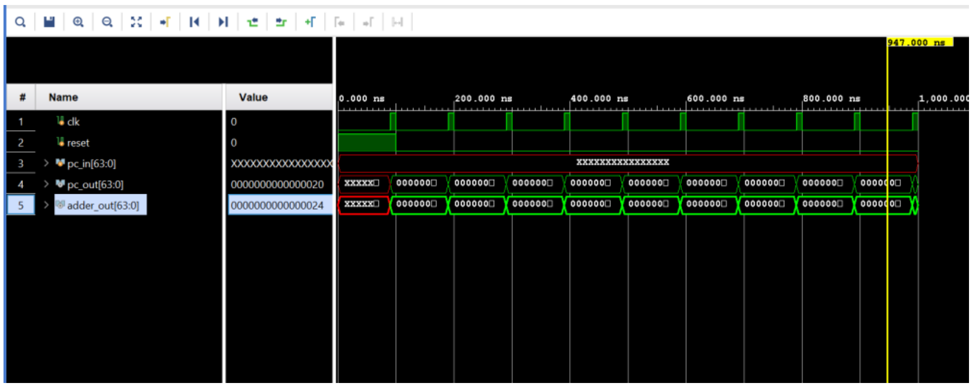
Program Counter Unit operation parameters:

- Input clk, reset 1 bit
- Input pc_in 64 bit
- Output pc_out 64 bit
- Wire adder_out 64 bit

Module Name: PC

Test Module Name: PC_tb

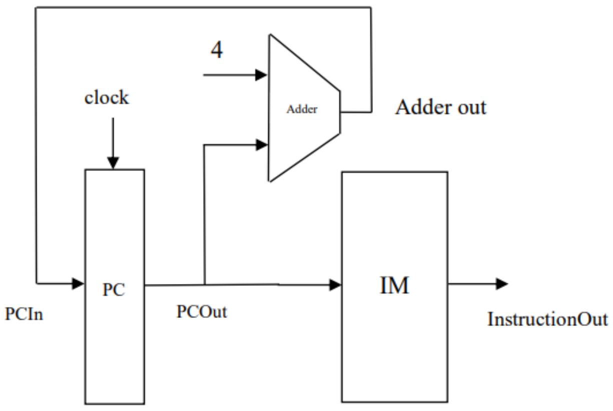
Below screen shot show the waveform for input and output parameters of Program Counter Unit:



#	Name	Value
1	clk	0
2	reset	0
3	pc_in[63:0]	XXXXXXXXXXXXXXXX
4	pc_out[63:0]	0000000000000020
5	adder_out[63:0]	0000000000000024

Here we are created two modules Adder and PC. In the PC module, the PC will be updating on the positive clock edge.

- Simulation of combine Program Counter and Instruction Memory Unit.



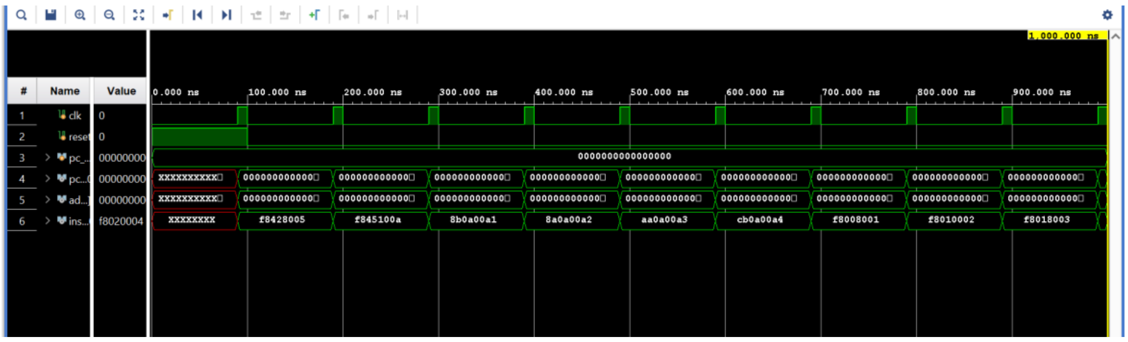
Combine Program Counter and Instruction Memory Unit operation parameters:

- Input clk, reset 1 bit
- Input pc_in 64 bit
- Output instruction_out 32 bit

Module Name: PC_IM

Test Module Name: PC_IM_tb

Below screen shot show the waveform for input and output parameters of Combine Program Counter and Instruction Memory Unit:



Here we have created IM module to load the instruction and then Combine IM with PC created in part A.

As IM is read only we will initialize the memory with instruction as follow:

- Load registers 5 and 10 from data memory locations 40-47 and 80-87;
- Perform the 4 ALU instructions on them (as in a previous Lab) and write back the results in Registers R1, R2, R3 and R4
- Store those results of the calculation in data memory locations 1 onward (8-15, 16-23 etc.)

The formats and opcodes for these instructions are required to fill the instruction memory, given in the figure bellow:

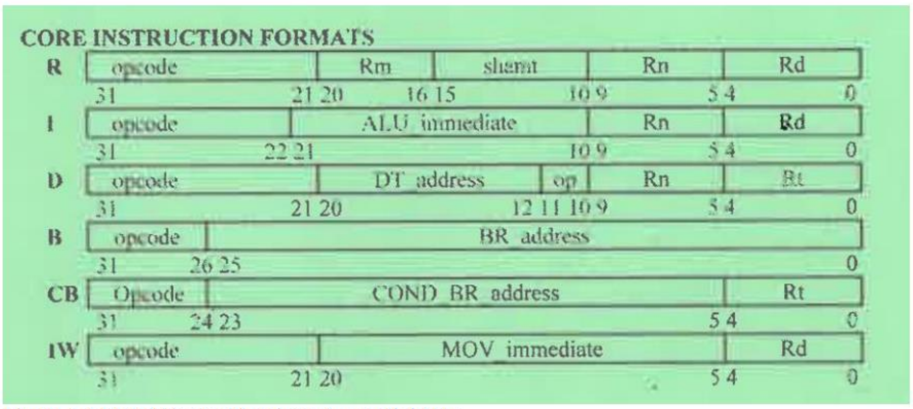
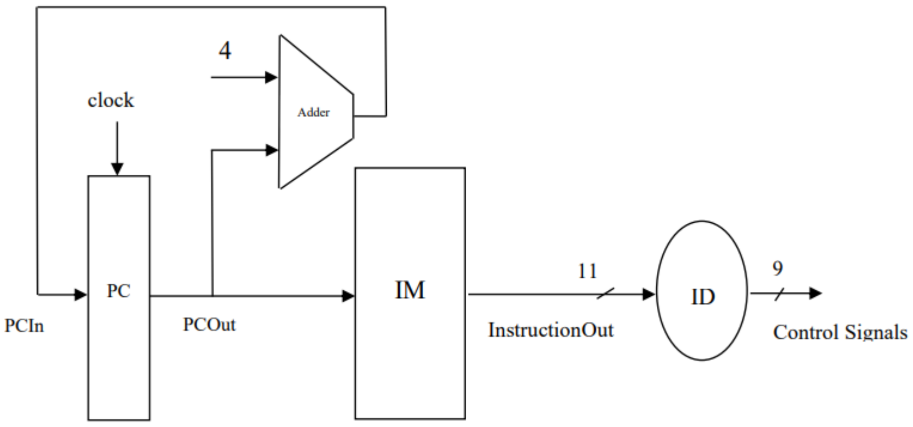

CORE INSTRUCTION FORMATS
R | opcode | Rm | shamt | Rn | Rd
31 21 20 16 15 10 9 5 4 0
I | opcode | ALU immediate | Rn | Rd
31 22 21 10 9 5 4 0
D | opcode | DT address | op | Rn | Rt
31 21 20 12 11 10 9 5 4 0
B | opcode | BR address
31 26 25 0
CB | Opcode | COND BR address | Rt
31 24 23 5 4 0
IW | opcode | MOV immediate | Rd
31 21 20 5 4 0

Figure 5.3 LEGv8 instruction format, copyright [1]

- Simulation of combine Program Counter, Instruction Memory and Instruction Decoder Unit.



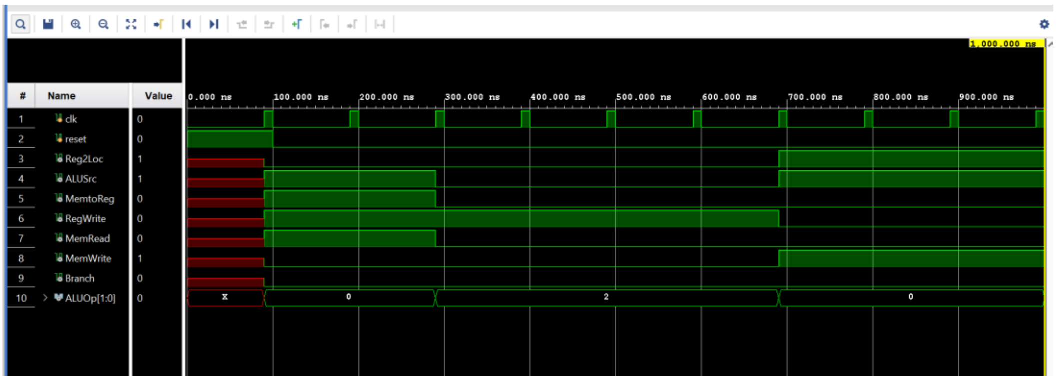
Combine Program Counter, Instruction Memory and Instruction Decoder Unit operation parameters:

- Input clk, reset 1 bit
- Output Reg2Loc, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch 1 bit
- Output ALUOp 2 bit

Module Name: PC_IM_ID

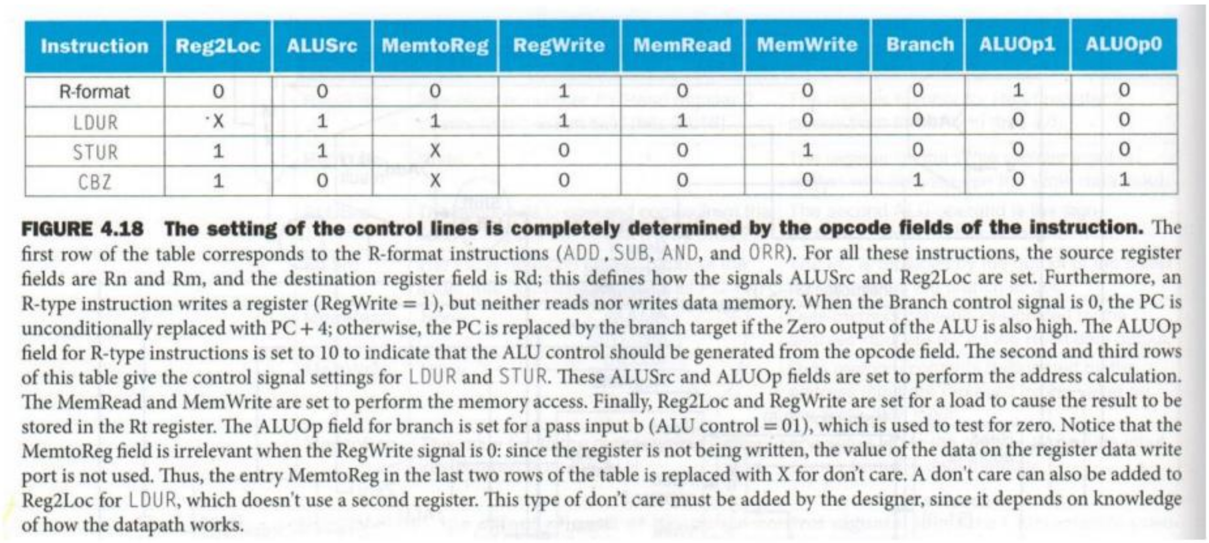
Test Module Name: PC_IM_ID_tb

Below screen shot show the waveform for input and output parameters of Combine Program Counter, Instruction Memory and Instruction Decoder Unit:



#	Name	Value
1	clk	0
2	reset	0
3	Reg2Loc	1
4	ALUSrc	1
5	MemtoReg	0
6	RegWrite	0
7	MemRead	0
8	MemWrite	1
9	Branch	0
10	ALUOp[1:0]	0

Here we created ID module to decode instruction read by IM and then combine ID with module created in part B. To decode Instruction below is the list of 9 control signals (7 control signals + the 2 bit ALUOp) which convert 11 bit Opcode to 9 bit control signal.



Instruction	Reg2Loc	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	0	0	0	1	0	0	0	1	0
LDUR	X	1	1	1	1	0	0	0	0
STUR	1	1	X	0	0	1	0	0	0
CBZ	1	0	X	0	0	0	1	0	1

FIGURE 4.18 The setting of the control lines is completely determined by the opcode fields of the instruction. The first row of the table corresponds to the R-format instructions (ADD, SUB, AND, and ORR). For all these instructions, the source register fields are Rn and Rm, and the destination register field is Rd; this defines how the signals ALUSrc and Reg2Loc are set. Furthermore, an R-type instruction writes a register (RegWrite = 1), but neither reads nor writes data memory. When the Branch control signal is 0, the PC is unconditionally replaced with PC + 4; otherwise, the PC is replaced by the branch target if the Zero output of the ALU is also high. The ALUOp field for R-type instructions is set to 10 to indicate that the ALU control should be generated from the opcode field. The second and third rows of this table give the control signal settings for LDUR and STUR. These ALUSrc and ALUOp fields are set to perform the address calculation. The MemRead and MemWrite are set to perform the memory access. Finally, Reg2Loc and RegWrite are set for a load to cause the result to be stored in the Rt register. The ALUOp field for branch is set for a pass input b (ALU control = 01), which is used to test for zero. Notice that the MemtoReg field is irrelevant when the RegWrite signal is 0: since the register is not being written, the value of the data on the register data write port is not used. Thus, the entry MemtoReg in the last two rows of the table is replaced with X for don't care. A don't care can also be added to Reg2Loc for LDUR, which doesn't use a second register. This type of don't care must be added by the designer, since it depends on knowledge of how the datapath works.

Figure 5.7 Control signals, LEG V8, copyright [1]