BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE , PILANI K. K. BIRLA Goa Campus
First Semester 2020 - 2021
CS F342 Computer Architecture
Lab - 5

Prerequisite - Parameterised Modules in Verilog

Verilog supports the use of parameters when instantiating modules. Parameters make it possible for modules to be instantiated with different specifications. They remove redundancies in code and make it more readable. For example, a 2:1 mux can be parameterised, so that the same module can be used to instantiate a 32 bit or a 5 bit 2:1 mux.

A. Making a module parameterised -

Syntax : In order to parameterise a module, the `parameter` keyword is used. The parameter is assigned a default value. It is then used in the port list for instance, to specify the (input and output) port width.

```
// DFF that triggers on the positive clock edge
module dff_p (input clk, input reset, input write, input [bwidth-1:0] in, output reg [bwidth-1:0] out);

    // default bit width of the DFF is 1
    parameter bwidth = 1;

    always @(posedge clk)
        if (reset == 1'b0)
            if (write == 1'b1)
                out <= in;

    always @(reset)
        if (reset == 1'b1)
            out <= 0;

endmodule
```

In the above example, if the default value was 8 instead of 1, it would correspond to 8 (1bit DFFs) in parallel. The ports would be elaborated as -

```
(input clk, input reset, input write, input [7:0] in, output reg [7:0] out);
```

B. Using a parameterised module (required for this lab) -

To call a parameterised module, the desired value of the parameter can be specified before the instance name, within a `#(n)` declaration, as shown below. If this declaration is missing, the module assumes the default value of its parameter (bwidth = 1 for dff_n in this example).

```
dff_n #(32) a(clk, reset, 1'b1, regRs, outa);

dff_n #(32) b(clk, reset, 1'b1, regRt, outb);

sign_ext #(16,32) signext(irOut[15:0], signExtOut);
```

* In case multiple parameters need to be defined, they can be specified one after another (comma separated), in the same order as they were declared. For example, looking at the IM module from memory.v -

```
IM #(8, 7) instr_memory (clk, reset, … );
```

… ^would instantiate an 8 bit wide (i.e. byte organised) RAM with 7 address lines, or 2^7 = 128 locations.
For more information about syntax and usage of parameters , check out the link or the files provided. (dff.v)

Additional Info - Include Statements in Verilog

In order to make projects more organised, like any other conventional programming language, Verilog supports the use of include statements. When a file is included in another, its contents are copied into the top level file at compile time. The syntax for including files is shown in the images that follow.

In the example shown below, the file “mux.v” is included in the file “memory.v”. With this declaration, the module “mux2to1” can be called and instantiated normally in the memory file, even though its definition lies in an external file. Note that “mux.v” must exist in the same directory as “memory.v”

memory.v

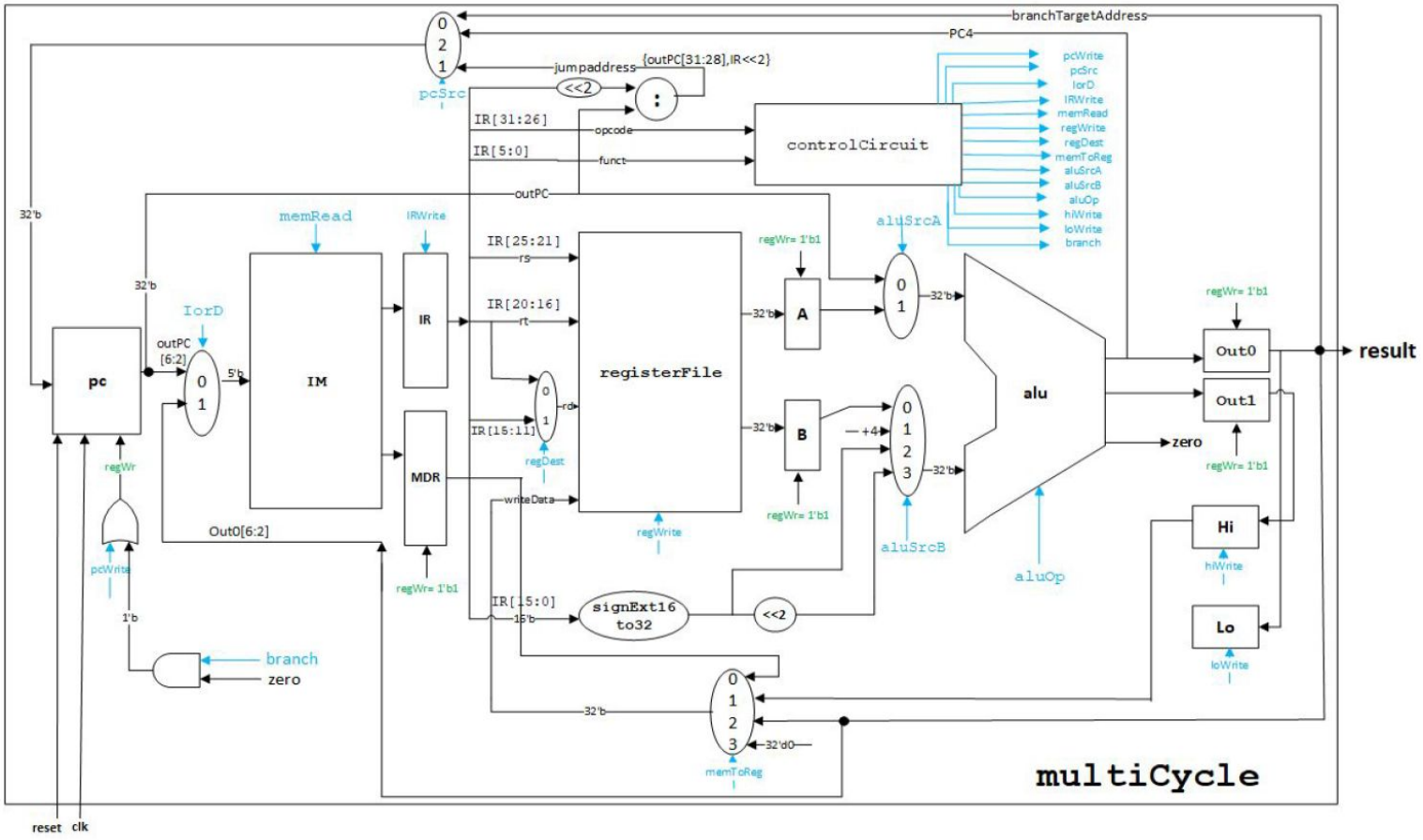
```
`include "mux.v"

module IM (
    input clk,
    input reset,
    input [ad_lines-1:0] address,
    input memRd,
    output [bwidth-1:0] data_out
);

    wire [31:0] data;

    parameter bwidth = 32;
    parameter ad_lines = 6;

    reg [bwidth-1:0] block [0 : (2 ** ad_lines) - 1];

    assign data = block[address];

    mux2to1 #(32) mux_final (32'h0, data, memRd, data_out);
```

mux.v

```
module mux2to1 (
    input [bwidth-1:0] in0,
    input [bwidth-1:0] in1,
    input sel,
    output [bwidth-1:0] out
);

    parameter bwidth = 1;

    wire [bwidth-1:0] select_bus;

    assign select_bus = {bwidth{sel}};
    assign out = (in0 & (~select_bus)) | (

endmodule
```

Lab Task

The following top level schematic (for multi_cycle.v) needs to be written to implement a multicycle processor.



Note that PC, IR, MDR, A, B, Out0, Out1, Hi and Lo are all 32 bit “intermediate_registers”, instantiated from the module in the file - “registers.v”.

* The IM block used in the processor must have 5 address lines, and a bit width of 32.
*** It should also be instantiated with the name “instruction_memory”.**

Specifications and Diagrams

Additionally, the alu and control circuit modules need to be implemented too, and their codes must be written in the files “alu.v” and “control.v” respectively. The templates for these files have been provided in the given .zip file.

The processor supports the following instructions (shown along with their formats).

Instruction Format

TYPE	INSTRUCTION	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	CYCES
		OPCODE	RS	RT	RD	SHAMT	FUNCT	
R	MULT	011000	RS	RT	00000	00000	000000	IF, ID, EX, WB
R	MFHI	010000	00000	00000	RD	00000	000000	IF, ID, WB
		OPCODE	RS	RT		IMM		
I	BEQ	000100	RS	RT		IMM		IF, ID, WB
I	ADDI	001000	RS	RT		IMM		IF, ID, EX, WB
I	LW	100011	RS	RT		IMM		IF, ID, EX, MEM, WB
		OPCODE			IMM			
J	J	000010			IMM			IF, ID, WB

Different operations of each instruction are done on successive clock cycles (a single instruction takes multiple cycles to execute). Which stages are involved for each instruction has been shown in the following table.

Instruction	**Instruction fetch**	**Instruction decode/ register fetch**	**Execution**	**Memory access/ Write Back**	**Memory read**
mult rs rt	IR ← Memory [PC] PC ← PC + 4	A ← Reg [IR [25:21]] B ← Reg [IR [20:16]] ALUOut ← PC + signextend (IR[15:0]) <<2)	ALUOut ← RegRs X RegRt	As the result is more than 32 bits, store the first (MSB) 32 bits in Register Hi and last (LSB) 32 bits in Register Lo.	-
mfhi rd			-	RegRd← Hi	-

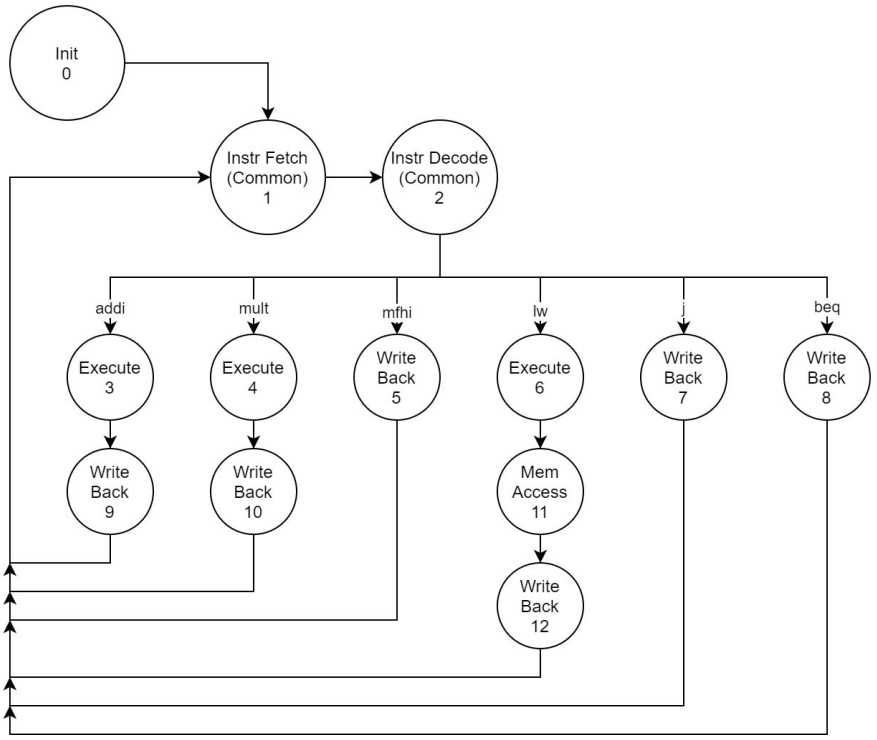
beq rs rt immediate			-	If (A == B)
				zero ← 1
				Mux will choose (Sel = 1)
				PC ← ALUOut
				
				Else
				zero ← 0
					-

addi rs rt immediate			ALUOut ← RegRs + imme	RegRt ← ALUOut	-
lw rs rt immediate			ALUOut← A + signextend (IR[15:0])	MDR ← Memory [Aluout]	Reg [IR[20:16]] ← MDR
j immediate			-	PC ← {PC[31:28], (IR[25:0]<<2}	-

Control Circuit

The control circuit consists of an FSM made of a total of 13 states. At each negative clock edge, the state gets updated according to the following state diagram. The next state would depend upon the current state as well as the opcode.

* A 4 bit internal `reg` can be declared to keep track of the 13 ($< 2^4$) states.



The values of control signals depend on the current state of the control circuit. Their values for each state have been shown in the table below.

State	*IorD*	*memRead*	*IRWrite*	*RegDst*	*regWrite*	*aluSrcA*	*aluSrcB*	*aluOp*	*hiWrite*	*loWrite*	*memToReg*	*pcSrc*	*pcWrite*	*branch*
1	1'b0	1'b1	1'b1	1'b0	1'b0	1'b0	2'b01	2'b00	1'b0	1'b0	2'b00	2'b10	1'b1	1'b0
2	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	2'b11	2'b00	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0
3	1'b0	1'b0	1'b0	1'b0	1'b0	1'b1	2'b10	2'b00	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0
4	1'b0	1'b0	1'b0	1'b0	1'b0	1'b1	2'b00	2'b10	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0
5	1'b0	1'b0	1'b0	1'b1	1'b1	1'b0	2'b00	2'b00	1'b0	1'b0	2'b01	2'b00	1'b0	1'b0
6	1'b0	1'b0	1'b0	1'b0	1'b0	1'b1	2'b10	2'b00	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0
7	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0	2'b00	2'b01	1'b1	1'b0
8	1'b0	1'b0	1'b0	1'b0	1'b0	1'b1	2'b00	2'b01	1'b0	1'b0	2'b00	2'b00	1'b0	1'b1
9	1'b0	1'b0	1'b0	1'b0	1'b1	1'b0	2'b00	2'b00	1'b0	1'b0	2'b10	2'b00	1'b0	1'b0
10	1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	2'b00	2'b00	1'b1	1'b1	2'b00	2'b00	1'b0	1'b0
11	1'b1	1'b1	1'b0	1'b0	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0
12	1'b0	1'b0	1'b0	1'b0	1'b1	1'b0	2'b00	2'b00	1'b0	1'b0	2'b00	2'b00	1'b0	1'b0

ALU

The ALU for this question supports only three operations, i.e ADD, SUB and MUL. For this task, the ALU would need to have a 64 bit output to support 32x32 multiplication.

aluOp	Operation	alu_out[63:32]	alu_out[31:0]
00	Add	32'h0	in1 + in2
01	Sub	32'h0	in1 - in2
10	Multiply	(in1 * in2)[63:32]	(in1 * in2)[31:0]

List of Files

Filename	Modules Inside + Explanation
alu.v	alu - Main alu of the multicycle processor. AluOut0 contains lower 32 bits of result when multiply operation is done ,and AluOut1 contains upper 32 bits. For other operations AluOut0 will contain the 32 bit result and AluOut 1 will be 0.
control.v	control_circuit - Finite State Machine based on the given state diagram and output control signals depends on current signals as given in table above
dff.v	dff_n - parameterised dff module activated at negedge dff_p - parameterised dff module activated at posedge
memory.v	im - instruction memory for the processor
mux.v	mux2to1 - 2 to 1 parameterised input length mux mux4to1 - 4 to 1 parameterised input length mux
register_file.v	register_file - register file containing 32 registers 32 bit each. 2 read ports and 1 write port. R0 is always 0
registers.v	intermediate_reg - parameterised length register used for state registers and to store values during the multicycle datapath, negedge triggered
sign_ext.v	sign_ext - parameterised input and output length sign extension. #(input length, output length)
multi_cycle.v	multi_cycle - your Multi Cycle implementation using modules defined in other files
testbench.v	testbench - testbench implementation to get testing signals for waveform. Change yourID_Lab5.vcd to your credentials inside this module

The files that need to be modified have been highlighted in yellow. The remaining files need not be changed.

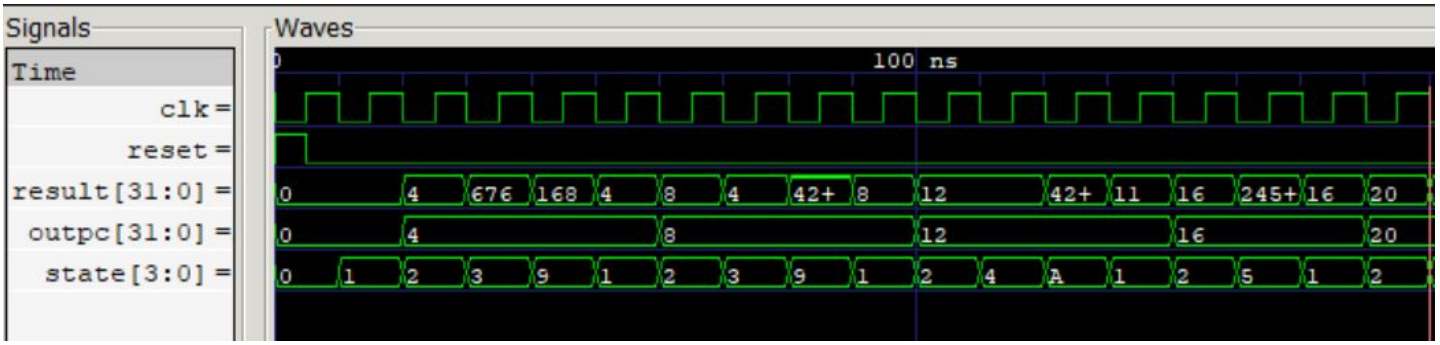
The following command should be used while compiling the testbench (the .vvp filename may vary) :

```
> iverilog -s testbench -o YourID_Lab5.vvp testbench.v
```

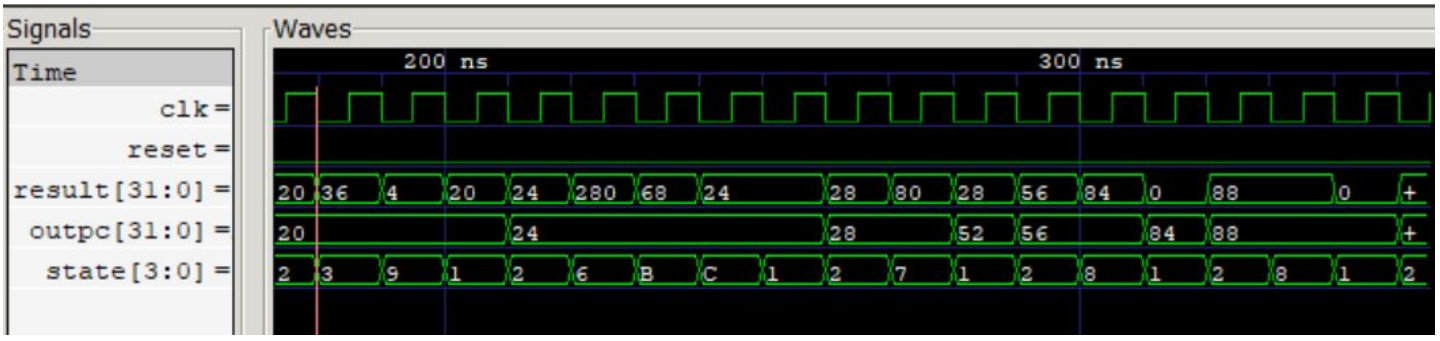
DO NOT CHANGE ANY OTHER FILE NAMES

Expected waveform

(The First 180ns)



(From 180ns - 355ns)



Marking Scheme:-

1 mark : result (testbench → result [31:0])
1 mark : outpc (testbench → mc → outpc [31:0])
2 marks : state (4 bit signal taken from controlCircuit module)

Submission Method

1. Save the vcd dump file generated as **YourID_Lab5.vcd** (this will already be called **YourID_Lab5.vcd** since you have changed it in the testbench).
2. Save your GTKWave output as **YourID_Lab5.gtkw** using the 'Save As' option in File->Write.

All files (**YourID_Lab5.vcd**, **YourID_Lab5.gtkw** AND **all the verilog files**) must be compressed into a .zip file which is to be submitted on Quanta.

* Do not create archives in other formats (rar, tar.gz etc).

* Once uploaded on Quanta, remember to submit for grading. Do not leave it as a draft.