PROJECT FOR VLSI II STUDENTS
EEE 458 JULY 2018 SEMESTER

Objective: The objective of the project is to build team works with 6 students in a group and to design from concept to layout including the testbench to verify the operation of simplified version of a Microprocessor without Interlocked Pipeline Stages (MIPS) processor and create the gate level structure followed by physical design that covers floorplanning, power mesh, clock tree synthesis, nanorouting, post-routing timing optimization and design rule check (DRC).

Deliverables: The project has to be completed according to the following top-down hierarchy keeping structured design in mind. Every functional block of MIPS architecture has to be implemented module by module. The working functionality of every instruction has to be verified.

Below is a description of the work flow to aid the project completion:

1. System design with functional and architectural analysis. The specification of the chip in terms of functionality and I/O pins have to be documented at this stage.
2. Write a Verilog description of the sub-block and simulate it using Cadence IUS and verify the functionality.
3. In this stage you will synthesize your design with gpdk 45 standard cell library *GSCLIB045* and also verify your synthesized netlist using Cadence IUS. Use Cadence Genus for synthesis and IUS verification.
4. In this stage you will do physical design of your chip using Cadence Innovus. A lot of things are needed to care at this stage.
5. Complete chip layout with I/O pad placement, clearing all DRC errors and produce gds for tape out.
6. Reporting via Project report and power point presentation.

Project Details:

A 32-bit MIPS processor has to be designed keeping the following specifications in mind.

- The instruction set will be 32-bit and compatible with R-format, load operation, store operation, branch and unconditional jump.
- The six students will form 3 sub-group. Initially, One sub-group will work on ALU and Register, another group on Instruction memory and Data memory and the third group on the control circuit design. After finishing their respective job all the sub-group will work as one group and integrate the whole design.
- The MIPS module (Processor module) must be designed only using synthesizeable Verilog code. Proper commenting throughout the code will be given positive marks.

- Although traditional MIPS architecture has 32 registers reserved for proper functionality, for the design purpose only $t0 - $t8 registers are required with 5-bit addressing i.e 5’b00000 corresponding to $s0 and 5’b01000 corresponding to $t8.

- With 32-bit addressing, the data memory has a size of 4Gb. For project demonstration, only the first 32 word memories could be initialized to zero.
- The students have to write the machine code in the instruction register. Only the first 32 word memories could accommodate the entire machine code of the given program.
- The R-format includes ALU operation such as ADD, SUB, AND, OR, NOR, SLT.
- The following opcode, function, alu_op and alu_control bits have to be implemented according to their respective operation.
- Group who could implement the design in FPGA board will be awarded with bonus marks.

SET - 1

Operation	Opcode	Function	alu_op	alu_control
R-format (ADD)	000000	100000	10	0010
R-format (SUB)	000000	100010	10	0110
R-format (AND)	000000	100100	10	0000
R-format (OR)	000000	100101	10	0001
R-format (SLT)	000000	101010	10	0111
R-format (NOR)	000000	101111	10	0011
Beq	000100	xxxxxx	01	0110
Bne	000110	xxxxxx	01	0110
Lw	100011	xxxxxx	00	0010
Sw	101011	xxxxxx	00	0010
Jump	100110	xxxxxx	00	0010
Addi	101000	xxxxxx	00	0010

- Finally the students have to implement the following c-code whose equivalent assembly language is also given below.

```
// c code - 01
int a=0;          // temporary variable 1
int b=1;          // temporary variable 2
int c;            // variable three
for (int i=0; i<20; i=i+2)
{
        c=a+b;          // variable 3 = variable 1 + variable 2
        a=b;            // variable 1 gets the value of variable 2
        b=c;            // variable 2 gets the new value of variable 3
}
 // store value of "c" in data memory
```

```
// Assembly code
//initialize the first two variables in temp register
addi    $t1, $t1, 0          //                                          PC=04
addi    $t2, $t2, 1          //                                          PC=08
addi    $t3, $t3, 0          // $t3 holds the value of int i.            PC=12
addi    $t4, $t4, 20         // $t4 = 20.                                PC=16
add     $s1, $t1, $t2        // c=a+b.                                   PC=20
addi    $t1, $t2, 0          // a=b.                                     PC=24
addi    $t2, $s1, 0          // b=c.                                     PC=28
addi    $t3, $t3, 2          // i=i+2.                                   PC=32
beq     $t4, $t3, 1          // if branch taken then jump to PC+4+4
                             //                                          PC=36
j       5                    // else jump to PC=20. Loop continues       PC=40
sw      $s1, 5($t4)          // memory[5+$t4] = $s1                      PC=44
slt     $s2, $t4, $s1        // if $s2=1 then c>20                       PC=48
                             // Program ends here
```

SET - 2

Operation	Opcode	Function	alu_op	alu_control
R-format (ADD)	000000	100000	00	1010
R-format (SUB)	000000	100010	00	1110
R-format (AND)	000000	100100	00	0000
R-format (OR)	000000	100101	00	0001
R-format (SLT)	000000	101010	00	0101
R-format (NOR)	000000	101111	00	0011
Beq	000100	xxxxxx	01	1110
Bne	000110	xxxxxx	01	1110
Lw	100011	xxxxxx	10	1010
Sw	101011	xxxxxx	10	1010
Jump	100110	xxxxxx	10	1010
Addi	101000	xxxxxx	10	1010

- Finally the students have to implement the following c-code whose equivalent assembly language is also given below.

```
// c code - 02
int n = 15;                     // an integer n
int j = 2;                      // n will be divided by this variable j
int i = j+0;                    // temp variable to accumulate j at every loop
while(i<n)
{
        a = n – i;              // this keeps the difference between n and multiple of j
        if (a == 0)
        {
                break;
        }
        i = i+j;
}
// store value of “a” in data memory
```

```
//Assembly code
// initializing n, j, i
addi $s1, $s1, 17       // $s1=17                                   PC=4
addi $s2, $s2, 3        // $s2=3                                    PC=8
addi $t1, $s2, 0        // $t1=i                                    PC=12
sub $t2, $s1, $t1       // a = n – i                                PC=16
slt $t3, $s1, $t1       // checking if n<i then t3=1                PC=20
beq $s1, $t1, 4         // if n == i, then go to PC+4+ 16           PC=24
bne $t4, $t3, 2         // if t3=1 then n<i so break out of while loop  PC=28
add $t1, $t1, $s2       // i = i + j accumulate multiples of j in i PC=32
j 4                     // jump to PC=16                            PC=36
add $t2, $t2, $s2       // a = a + j                                PC=40
                        // can you guess what “a” contains now?
sw $t2, 8($s2)          // memory[8+$s2] = $t2                      PC=44
```