CENG311 - Computer Architecture **Fall 2024**

PROGRAMMING ASSIGNMENT #3 **Due date:** 07.01.2025 08.30

In this assignment, you are required to create your own ISA (Instruction Set Architecture) and its implementation by customizing or updating the MIPS-lite single-cycle implementation (as discussed during your lab sessions). You will use the ModelSim simulator to develop and test your code.

Your processor will remain a 32-bit processor, but the register file will have 16 registers. The set of ISAs for all students is shown in Table 1. You can change the order of the instructions as desired; the table is provided only as an example. For R-type instructions, you have the flexibility to modify the function bits and ALU control lines as per your requirements (or keep them unchanged). If implementing specific instructions requires you to add new registers, control blocks, or multiplexers, you are allowed to do so. However, if you add any new blocks, you must also provide a figure showing the updated design of your processor. This will be presented during the assignment demonstration lab.

Additionally, note that the opcode for each instruction will vary based on your student ID. The least significant 3 bits of your ID determine the opcode for the first instruction. For example:

Student id: 230062100 -> 100 is the first instruction's opcode.

opcode (decimal)	opcode (binary)	function field (decimal)	function field (binary)	instruction
100	01100100	1	0001	sll (R-type)*
100	01100100	2	0010	srl (R-type)*
100	01100100	3	0011	nor (R-type)*
100	01100100	4	0100	sub (R-type)
100	01100100	5	0101	or (R-type)
101	01100101	x	xxxxx	lw (I-type)
102	01100110	x	xxxxx	sw (I-type)
103	01100111	x	xxxxx	beq (I-type)
104	01101000	x	xxxxx	bne (I-type)*
105	01101001	x	xxxxx	addi (I-type)*
106	01101010	x	xxxxx	j (J-type)*
107	01101011	x	xxxxx	jalfor (Jal-for type) *

Table 1. An example of instruction set architecture.

There are a total of 12 instructions in Table 1, some of which have been implemented during the lab sessions. Among these, the instructions marked with * are required to be implemented. Below, the instructions that need to be implemented are explained in detail.

1) R-type

8 bits	4 bits	4 bits	4 bits	6 bits	6 bits
Opcode[31:24]	rs[23:20]	rt[19:16]	rd[15:12]	shamt[11:6]	funct[5:0]

a) Shift Left Logical (sll): This instruction shifts the value in the rt register to the left by the number of positions specified in the shamt field of the R-type instruction. The shifted result is then written to the rd register.

b) Shift Right Logical (srl): The value in the rt register is shifted to the right by the number of positions defined in the shamt field of the R-type instruction. The outcome of the operation is stored in the rd register.

c) **NOR (nor):** This instruction performs a bitwise NOR operation on the values of the rs and rt registers. The result is then saved in the rd register.

2) I-type

8 bits	4 bits	4 bits	16 bits
Opcode[31:24]	rs[23:20]	rt[19:16]	address[15:0]

a) **Branch Not Equal (bne):** The instruction compares the contents of the rs and rt registers. If the values in the two registers are not equal, the program branches to the target address, which is the given 16-bit address plus the value of PC + 4.

b) **Add Immediate:** This instruction adds a constant (immediate value) to the value in the rs register and stores the result in the rt register. It is often used for updating values or computing addresses with offsets.

3) J-type

8 bits	24 bits
Opcode[31:24]	address[23:0]

a) **Jump (j):** The program directly jumps to the given address. For example, if the address is 0x4, the program counter (PC) is updated to PC = 0x4.

4) Jal-for type

8 bits	4 bits	4 bits	16 bits
Opcode[31:24]	nr[23:20]	necl[19:16]	address[15:0]

a) **Jump and Link For (jalfor):** In this instruction, we define a new type called Jump and Link For (jalfor). The for loop will be implemented using the jump instruction. The first eight bits of the instruction are allocated for the opcode, and the number of repetitions for the loop will be taken from the nr (number of repetitions) field. The necl (number of executed code lines) field, which is 4 bits long, specifies how many lines of code will be executed in each iteration of the loop. The remaining 16-bit field is the target address for the jump.

The jalfor instruction first performs a direct jump to the specified address. Then, the program will execute the number of lines indicated by the necl field from that address. After executing those lines, the program returns to the address specified in the instruction and repeats the execution for the number of times specified in the nr field. Once the loop is complete, the program returns to the address from where the jalfor instruction was originally called.

01101011	0010	0010	0000000000001100 (0x0C)
Opcode[31:24]	nr[23:20]	necl[19:16]	address[15:0]

For example, if the jalfor instruction has a PC (program counter) value of 0x04, a jump target address of 0x0C, nr = 0x02 (number of repetitions), and necl = 0x02 (number of executed code lines), the following happens:

- First, the program jumps to the address 0x0C.
- In the first iteration, the code at PC 0x0C and PC 0x10 is executed.
- In the second iteration, the same code at PC 0x0C and PC 0x10 is executed again.

• Finally, after the loop ends, the program moves to PC 0x08 (0x04+4) and continues executing from there.

You need to design revised single-cycle data path and control units that create a processor that executes all instructions as well as the instructions already implemented in the design. You must also implement the **nor, addi, sll, srl, bne, j, and jalfor** instructions. After designing the updated version of the MIPS processor, you implement it in Verilog hardware description language.

You must submit a report containing only your ISA and Verilog code. If any new registers, multiplexers, or control blocks are added, a diagram illustrating these additions must also be included in the report.

Additionally, on a date to be announced later, you will be required to show your homework to the TAs. During the demonstration, a total of four code blocks will be executed, three of which are listed below, and the fourth will be specified during the demonstration. Therefore, please ensure that all instructions are functioning correctly before the demonstration.

Furthermore, in the demonstration, the registers, data memory, and the executing instruction should be displayed as shown in the lab section. This includes printing the opcode, rs, rt, rd, address, function field, and other relevant fields separately to show to the TAs.

Finally, during the demonstration, you must use the **initDataMemory.dat** and **initRegisterMemory.dat** files provided. Also, the program counter should complete within **500** picoseconds during the demonstration (this timing needs to be adjusted in the code as discussed in the lab).

Code Blocks for Demonstration

1:

```
 0 - jalfor $2, $2, 0x0008
 4 - j 0x0c
 8 - add $r0, $r1, $r0
12 - addi $r0, $r0, 0x05
16 - addi $r0, $r0, 0x05
20 - addi $r0,$r0, 0x05
```

2:

```
 0 - or $r1,$r0,$r2
 4 - j 0x0c
 8 - nor $r0, $r1, $r2
12 - beq $r0,$r1, 0xfffe
16 - addi $r1, $r0, 0x00
20 - bne $r0,$r1, 0xfffe
```

3:

```
 0 - nor $r2,$r0,$r1
 4 - addi $r0, $r0, 0x01
 8 - j 0x10
12- sw $r0,$r0,0x00
16- beq $r0,$r1, 0xfffe
20- bne $r0,$r1, 0xfffb
```

Requirements:

•You must submit the source code and a report.

•You need to work individually, no group work is allowed.

•No late homework will be accepted.

Submission: You are required to submit all of your files to Teams. Please create a compressed file including all source files and report, and name it as yourstudentnumber_P3.zip (e.g. If your student number is 202112345678, the file name must be 202112345678_P3.zip). Your report should only include your processor design (if there are new blocks) and your ISA as provided in Table 1.