Experiment 7: Digital System Design

Due date: 06.05.2021 @04.16
Res. Asst. Kadir Özlem
kadir.ozlem@itu.edu.tr

1 Introduction

In this experiment, you will implement a single cycle CPU with Verilog HDL. Please read the introduction slide again carefully.
You can use any operator/code block in this experiment. You must simulate all parts and modules.

2 Part 1: Register Line and Decoder

In this part you will design:

- 4:16 decoder module with enable input. If enable signal is logical low, all the outputs of decoder will be logical low.
- 16-bit register line module. This module should take **lineselect**, **clock**, **reset**, and **16-bit dataIn** as input and give 16-bit **stored data** as output. At the falling edge of the reset signal data will be cleared. At the rising edge of the **clock** signal, the module should store the data which is given as input when the **lineselect** is high.

3 Part 2: Register File Design

In this part you should implement a 16 line 16-bit register file module (32 byte) using 4:16 decoder module and 16-bit register line module. This module should take 4-bit **selA**, 4-bit **selB**, 4-bit **selWrite**, 16-bit **dataIn**, **reset**, **writeEnable**, and **clock** as input and give 16-bit **dataA** and 16-bit **dataB**. The operations of the register file module are given below.

- At the falling edge of the **reset**, all lines of the register file will be cleared.
- **selWrite** input selects the line to be updated. The selected line should store the **dataIn** input at the rising edge of the **clock** when the **writeEnable** is high.

Table 1: ALU Instructions

OPCODE	Operation Name	Operation	Update Flag
0	AND	dst = srcA & srcB	No
1	OR	dst = srcA | srcB	No
2	Addition	dst = srcA + srcB	Yes
3	Subtraction	dst = srcA - srcB	Yes
4	XOR	dst = srcA ˆ srcB	No
5	Logical Shift Right	dst = srcA >> srcB	No
6	Logical Shift Left	dst = srcA << srcB	No
7	LOAD	dst = srcB	No

- Register outputs are exported via **dataA** and **dataB**. **selA** selects the line of the register file for **dataA** output. Similarly, **selB** selects the line of the registerfile for **dataB** output. These two operations do not require the clock signal.

4 Part 3: ALU Design

In this part you will design an Arithmetic Logic Units (ALU) that does some operations in the Table 1. In addition to these operations, the ALU will produce zero flag. Meaning that, if the result of the operation is zero, a special flip-flop (called **zeroFlag**) will be set to logical high. The inputs of the ALU will be called **srcA**, **srcB**. The output will be called **dst**. The ALU will decide its operations with the help of 3-bit **Op** input. Also, the module has **clock** and **reset** inputs and **zeroFlag** output. At the falling edge of the **reset** zeroFlag will be cleared. At the rising edge of the **clock** zeroFlag will be updated according to output of the ALU when the operation updates the flags. For example, AND operation does not update the flag.

5 Part 4: Instruction Decoder

In this part, you will create the control signals of your digital system according to **instruction** input. Table 2 shows the instruction set of the digital system. According to instruction set there are 4 type of instruction in this system. Table 3, 4, 5, and 6 shows the instructions bits separation for register, immediate, load, and branch respectively. Inputs and outputs ports information are given below.

- **instruction** (15-bit Input): comes from the ROM and takes the current instruction as input.
- **opcode** (4-bit Ouput): contains the operation information of the instruction.
- **selWrite** (4-bit Ouput): contains the destination register address for writing operation.

Table 2: Digital System Instruction Set

OPCODE	**Operation Name**	**Operation**	**Type**	**Update Flag**
0	AND	dst = srcA & srcB	Register	No
1	OR	dst = srcA | srcB	Register	No
2	Addition	dst = srcA + srcB	Register	Yes
3	Subtraction	dst = srcA - srcB	Register	Yes
4	XOR	dst = srcA ˆ srcB	Register	No
5	Logical Shift Right	dst = srcA >>srcB	Register	No
6	Logical Shift Left	dst = srcA <<srcB	Register	No
7	LOAD	dst = Immediate	Load	No
8	ANDI	dst = srcA & Immediate	Immediate	No
9	ORI	dst = srcA | Immediate	Immediate	No
10	ADD_Immediate	dst = srcA + Immediate	Immediate	Yes
11	Compare	Compare srcA, srcB	Register	Yes
12	NOOP	-	-	No
13	B	PC = Immediate	Branch	No
14	BNE	PC = PC + Immediate if not equal	Branch	No
15	BEQ	PC = PC + Immediate if equal	Branch	No

- **selA** (4-bit Ouput): contains the first operand address information of the instruction in the registerFile.

- **selB** (4-bit Ouput): contains the second operand address information of the instruction in the registerFile.

- **fourBitImmediate** (16-bit Ouput): contains the 4-bit immediate value information for immediate instructions. Zero extension will be used to extend 4-bit to 16-bit.

- **eightBitImmediate** (16-bit Ouput): contains the 8-bit immediate value information for load and branch instructions. Zero extension will be used to extend 4-bit to 16-bit.

- **writeEnable** (1-bit Ouput): If the instruction needs to write the results to register file, it must be 1. Otherwise, it will be 0.

- **isLoad** (1-bit Ouput): The flag will 1 if the instruction is the load instruction.

- **isImmediate** (1-bit Ouput): The flag will 1 if the instruction is the immediate instruction.

- **isBranch** (1-bit Ouput): The flag will 1 if the instruction is the branch instruction (Opcode 13).

Table 3: Register Type Instructions

Opcode	dst	srcA	srcB
4-bit	4-bit	4-bit	4-bit

Table 4: Immediate Type Instructions

Opcode	dst	srcA	Immediate
4-bit	4-bit	4-bit	4-bit

- **isBranchNotEqual** (1-bit Ouput): The flag will be 1 if the instruction is the branchNotEqual instruction.
- **isBranchEqual** (1-bit Ouput): The flag will be 1 if the instruction is the branchEqual instruction.

6 Part 5: Program Counter

In this part, program counter module will develop the program counter module for the digital design. The output of the program counter module (**PC**) stores the current program address of the system. The inputs of the module are **reset**, **clock**, **isBranch**, **isBranchNotEqual**, **isBranchEqual**, and **immediateAddress (8-bit)**. The instruction fetched from the program memory using this information. **PC** will be cleared at the falling edge of the reset signal. If the operation is branch (Opcode 13), **PC** value will be the **immediateAddress** (8-bit input) at the rising edge of the clock. PC value will be the **immediateAddress** (8-bit input) at the rising edge of the clock. PC value will be current **PC** value + **immediateAddress** at the rising edge of the clock if the operation is branch not equal and flag shows the result is not equal, or the operation is branch equal and flag shows the result is equal.You can access the result information using zeroFlag. Otherwise, PC value will be next address of the memory at the rising edge of the clock.

7 Part 6: Mini Computer

In this part, you will implement the mini computer using the modules you implemented in the previous parts. The module only needs **clock** and **reset** signals. Clock and reset signal will be connected to other modules which needs the this signals. You must give

Table 5: Load Type Instructions

Opcode	dst	Immediate
4-bit	4-bit	8-bit

Table 6: Branch Type Instructions

Opcode	Unused	Immediate
4-bit	4-bit	8-bit

instruction, **dataA**, **dataB**, **dst**, **PC**, **input value of ALU srcB** for debugging and test of the system. You will also needs the ROM module, but we have already developed it for you. You can directly add this module to your design files. You must also include the memory data to your project, the guideline have been added to homework files for this operation. The input connection information of the modules are given below.

- Program Memory
  - PC <– PC output of the Program Counter
- Instruction Decoder
  - instruction <– instruction output of the Program Memory
- Register File
  - selA <– selA output of the Instruction Decoder
  - selB <– selB output of the Instruction Decoder
  - selWrite <– selWrite output of the Instruction Decoder
  - dataIn <– dst output of the ALU
  - writeEnable <– writeEnable output of the Instruction Decoder
- ALU
  - srcA <– dataA output of the Register File
  - srcB <– It can be dataB, 4-bit immediate or 8-bit immediate according to instruction type. You must select it for each type of instruction.
  - Op <– Opcode output of the Instruction Decoder
- Program Counter
  - zeroFlag <– zeroFlag output of the ALU
  - isBranch <– isBranch output of the Instruction Decoder
  - isBranchNotEqual <– isBranchNotEqual output of the Instruction Decoder
  - isBranchEqual <– isBranchEqual output of the Instruction Decoder
  - immediateAddress <– One of the immediate value output of the Instruction Decoder.

8 Report

- You can use any **software tool** for your circuit designs. You may attach them to the report as figures by properly referencing them in the text.
- Please use the table attributes of Latex. You can check out online Latex table generators (for example: `https://www.tablesgenerator.com`).
- Your report should contain information about the results of your simulations. If your implementations are not fully correct, discuss what the source of the errors might be in your report.
- For further details about the report, please check Ninova e-learning system.

9 Submission

- Please add comments to the code to clarify your intends.
- Please don't send any document via e-mail to one of the assistants.
- Your reports must be written in Latex format. Latex report template is available on Ninova. You can use any Latex editor of your choice. If you upload your report without Latex file, you directly get 0 as your report grade. You should upload both .pdf and .tex files of your report.
- You should submit 2 separate ".v" files for your Verilog codes: one containing the modules, one containing the simulation codes.
- Please make sure that you have written your full name and student ID number to every document you are submitting.
- Please only write the names of members who have contributed to the experiment.
- Note that late submissions are not accepted, be aware of the deadline.
- Please do not hesitate to contact me (kadir.ozlem@itu.edu.tr) for any question. Don't forget to have fun and stay healthy.