Middle East Technical University | Department of Computer Engineering

CENG 232

Logic Design

Spring '2016-2017

Lab 3

Due date: 23 April 2017, 23:59
No late submissions!

1 Introduction

We are done with the combinatorial part of the lesson, from now on you will be dealing with sequential circuits. Which means that, we will introduce the notion of memory to our hardware implementations. With use of small memory units, it is possible to build large projects which can store state information in them. With this capabilities it is possible to store, read or modify large data.

The last 3 labs will serve as building blocks to a big project. In each lab, you will implement a small part of a special version of a basic computer (microcomputer). You will achieve this by turning descriptions into working circuits. Your implementations in this basic computer implementation will include, registers, counters, memory implementations, arithmetic logic unit (ALU), control logic and so on. At the end of these 3 labs, you will produce a nearly complete microcomputer which is capable of reading some simple instructions from it's memory and perform them in it's own processor.

These 3 labs aim to make you familiar with Verilog language, related software tools, and the FPGA boards as well as microcomputer architectures.

Below a general microcomputer architecture is described. This will serve as a guideline for you in the 3 lab period. *It is only the general structure of the overall project.* In each lab, the part you will be working on will be specifically defined at **Lab Work** part. .

2 General Microcomputer Architecture

2.1 Workflow

A microcomputer is a general purpose system capable of performing a large variety of computations. Notion of time is represented as **cycle** in these architectures. In each cycle the microcomputer reads an instruction from it's memory (fetch). Decodes the instruction into logical blocks and understands it's task (decode). And then makes necessary computations to accomplish it's task and evaluates the result (execute). Cycle is the necessary time to make an operation. After execute state, microcomputer is ready to read the new operations. Each computation (**program**) is represented as a sequence of instructions stored in the memory. After microcomputer finishes a computation, it reads new instructions and starts it's new cycle (Figure 1).

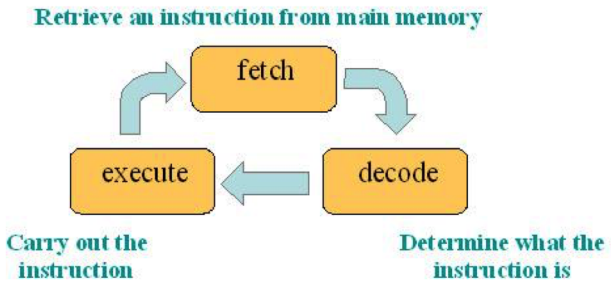

Figure 1: Microcomputer workflow.

2.2 Architecture

As shown in Figure 2, a microcomputer system consists of 3 subsystems:

- a **processor**;
- a **memory subsystem**; and
- **input/output(IO)** subsystem.

The **processor** is the computing engine of the microcomputer: it executes the sequence of the instructions, that is, the program. Registers are used to store temporary data and transfer information within microcomputer. It has some specific registers called **instruction registers (IR)**. These registers are used to read one instruction and store it temporarily. Each byte of this register has a different meaning. These parts are used to determine which operation this is (what is the task). Another important register is the **program counter (PC)**. Program counter holds the address of the next instruction to be read, it holds the address information because each instruction is stored in the memory. Apart from these specific registers, it also has some **general purpose registers** to make computation and store temporary data. Another important part of the processor is the **arithmetic and logic unit (ALU)**. ALU is the part that is responsible for making arithmetic calculations like addition, subtraction, bitwise operations etc. According to results of these operations it sets some specific bits of registers to emphasize the operation result. (Like setting carry bit when a summation operation causes overflow).

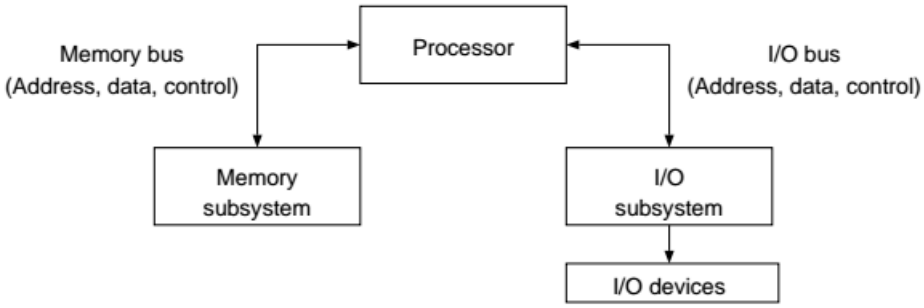

Figure 2: Microcomputer general structure.

The **memory subsystem** stores the program as well as the data used by the program. The memory is divided into **locations**, such as bytes and words; these locations are identified by their **address**. Memory operations are defined as **read** and **write** to memory locations.

The **I/O subsystem** contains the interfaces between the microcomputer systems and the external devices, such as keyboards, monitors etc.

In our implementation we won't be implementing a I/O subsystem. At the end of these 3 labs we aim to implement processor, memory subsystem and their connections.

3 Processor

The processor is a complex sequential system consisting of two subsystems: the **data subsystem** and **control subsystem**. The data subsystem contains the registers for the processor state and the operators for data modification, whereas the control subsystem controls the operation of the data subsystem.

4 Data Subsystem

This subsystem consists of a register file that contains the general-purpose registers, dedicated register for `PC`, `CR`, and `IR`, an arithmetic-logic unit (ALU), as well as other supporting modules, each of these has control signals that determine its operation. For example, signal `ALUop` determines the function performed by the ALU on the current input values, and signal `WrCR` produces the update of register `CR` with the value at its input. All register write operations are synchronized with signal `Clk`.

4.1 Dedicated Registers

The data subsystem contains dedicated registers for `IR`, `PC`, and `CR` (32 bits, 24 bits, and 4 bits respectively). To simplify the implementation, register `PC` is extended at the left with 8 bits that are set to zero; consequently, within the processor `PC` appears as a 32-bit register. The dedicated registers are loaded whenever the corresponding control input is set to 1 (`WrIR`, `WrPC`, and `WrCR`, respectively), synchronized with signal `Clk`. The registers are set to zero whenever signal `Reset` is active.

4.2 Multiplexer

Data Subsystem includes 32-bit 2-to-1 multiplexers in order to redirect data in the Processor architecture as needed. The inputs are selected via `Sel` signal.

4.3 Arithmetic-logic Unit

The arithmetic-logic unit (ALU) is a combinational module that performs the arithmetic and logic operations; it has two data inputs, one data output, one control input, and one condition output. The data inputs are labeled A and B, whereas the data output labeled C. The control input `ALUop` specifies which operation is performed according to the encoding given in Table 1. Some of the operators are explained as the following:

- **Zero_32** operation sets all the bits to zero and unary - is two's complement operator.
- **shiftl** shifts all the bits to the left by one and sets the rightmost bit to zero whereas **shiftr** shifts all the bits to the right by one and sets the leftmost bit to zero.
- **rotl** (rotate left) shifts all the bits to the left by one similar to **shiftl** except it sets the rightmost bit to the previous value of the leftmost bit as well as **rotr** (rotate right) shifts all the bits to the right by one similar to **shiftr** except it sets the leftmost bit to the previous value of the rightmost bit.

Table 1: ALU Operation table

ALUop	Operation
0000	Zero_32
0001	A + B
0010	A - B
0011	-B
0100	A and B
0101	A or B
0110	A xor B
0111	not(B)
1000	unused
1001	B
1010	shiftl(A)
1011	shiftr(A)
1100	rotl(A)
1101	rotr(A)
1110	A + 4
1111	unused

5 Lab Work

In this lab, you are expected to implement some of the modules of Data subsystem presented in the previous section namely Dedicated Register, Multiplexer, and Arithmetic-logic Unit. The specifications and implementation details are provided in the following sections.

5.1 Register32

This module is used to define Dedicated Registers in the Data Subsystem. The signature of the module is given below:

```
module Register32(
input [31:0] Data_in,
output reg[31:0] Data_out,
input Wr,
input Reset,
input Clk
    );
```

It has a 32-bit data input `Data_in`, a 32-bit data output `Data_out`, a write enable input `Wr`, a reset input `Reset`, and a clock input `Clk`. `Data_in` value is stored into `Data_out` if `Wr` is set by positive edge of `Clk` signal. `Data_out` is set to zero if `Reset` is set. Note that `Reset` is an asynchronous reset bit, that means it is not invoked by any clock pulse.

5.2 Mux32

This module is used to define Multiplexers in the Data Subsystem. The signature of the module is given below:

```
module Mux32(
input [31:0] A_in,
input [31:0] B_in,
input Sel,
output reg [31:0] Data_out
    );
```

It has two 32-bit data inputs `A_in` and `B_in`, a selection bit `Sel`, and a 32-bit data output `Data_out`. If `Sel` bit is 0 then `Data_out` is `A_in` value, otherwise it is `B_in`.

5.3 Arithmetic-logic Unit

This module is used to define Arithmetic-logic Unit (ALU) in the Data Subsystem. The signature of the module is given below:

```
module ALU(
input [3:0] A,
input [3:0] B,
input [3:0] ALUop,
output reg [3:0] C,
output reg [3:0] Cond
    );
```

It has two 4-bit data input `A` and `B`, a 4-bit operation code (opcode) input `ALUop`, a 4-bit data output `C`, and 4-bit condition output `Cond`. The module conducts the operation specified in Table 1. It basically reads the opcode from `ALUop` input, conducts the corresponding operation and stores the result to output register `C`.

The execution of some operations set `Cond`. The bits of this register are set to reflect the result from the execution of arithmetic and logic operations in Table 2. `Cond` output consists of these bits which are Z, N, C, and V in the most significant to least significant order.

Z and N are used for every operations as C and V bits are used only in binary - and + operations which have the `opCode` value `0001` and `0010`, respectively and in the other operations, C and V are set to zero.

C is used for detecting unsigned arithmetic error and V is used for detecting signed arithmetic ones. More explicitly, if C is set, the addition or subtraction result is not correct if it is to be interpreted as unsigned value. Similarly, if V is set that means the addition or subtraction result is not correct if it is to be interpreted as signed value. Some C and V bit cases are listed below:

- **Carry bit (C):**
  - 1111 + 0001 = 0000 (C is set to 1 since the most significant (leftmost) bits causes a carry out, yielding to a wrong result in unsigned arithmetic which is 0.)
  - 0000 - 0001 = 1111 (C is set to 1 since the most significant bits requires borrow, yielding to a wrong result in unsigned arithmetic which is 15.)
  - 0111 + 0001 = 1000 (C is set to 0 and the result is 8, correct in unsigned arithmetic.)

Table 2: Condition bits.

Zero	**Z**	set to 1 if the result is 0000, set to 0 otherwise.
Negative	**N**	set to 1 if the leftmost bit of the result is 1 (a negative value in two's complement representation), set to 0 otherwise.
Carry	**C**	set to 1 if the addition of two numbers causes a carry out of the most significant (leftmost) bits added or the subtraction of two numbers requires a borrow into the most significant (leftmost) bits subtracted, set to 0 otherwise.
Overflow	**V**	set to 1 if the addition of two numbers with the same sign (the most significant) bits gives a result with a different sign bit added or subtraction of two numbers with different sign bit gives a result with a sign bit different than that of A data input (the first operand of subtraction), set to 0 otherwise.

  - 1000 - 0001 = 0111 (C is set to 0 and the result is 7, correct in unsigned arithmetic.)

- **Overflow bit (V):**
  - 0100 + 0100 = 1000 (V is set to 1 since the sum of two numbers with the sign bits off (positive) results to a number with the sign bit is on (negative), yielding a wrong result in signed arithmetic.)
  - 1000 + 1000 = 0000 (V is set to 1 since the sum of two numbers with the sign bits on (negative) results to a number with the sign bit is off (positive), yielding a wrong result in signed arithmetic.)
  - 0100 + 0001 = 0101 (V is set to 0 and the result is 5, correct in signed arithmetic.)
  - 0110 + 1001 = 1111 (V is set to 0 and the result is -1, correct in signed arithmetic.)
  - 1000 + 0001 = 1001 (V is set to 0 and the result is -7, correct in signed arithmetic.)
  - 11000 + 110 = 1000 (V is set to 0 and the result is -8, correct in signed arithmetic.)

Further reading about carry (C) and overflow (V) bit can be accessed in the following web address: http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

6 FPGA Demonstration

You will make a demo of the **ALU** module using the FPGA board. `Board232.v` file (and a ready-to-use Xilinx project), which binds inputs and outputs of the FPGA board with your Verilog module, is provided on COW course page. FPGA inputs and outputs are given in Table 3 with their related `ALU` module inputs and outputs and also illustrated in Figure 3. You are required to test your Verilog module on the FPGA boards. After the submission date, you will make a demo to course assistants.

Table 3: FPGA inputs and outputs.

Name	FPGA Board	Description
`A`	SW7, SW6, SW5, SW4	Leftmost 4 switches (A)
`B`	—	Provided in `Board232.v` file
`ALUop`	SW3, SW2, SW1, SW0	Rightmost 4 switches (Op)
`C`	LD7, LD6, LD5, LD4	Leftmost 4 leds (C)
`Cond`	7-segment display	Cond in Z, N, C, V order

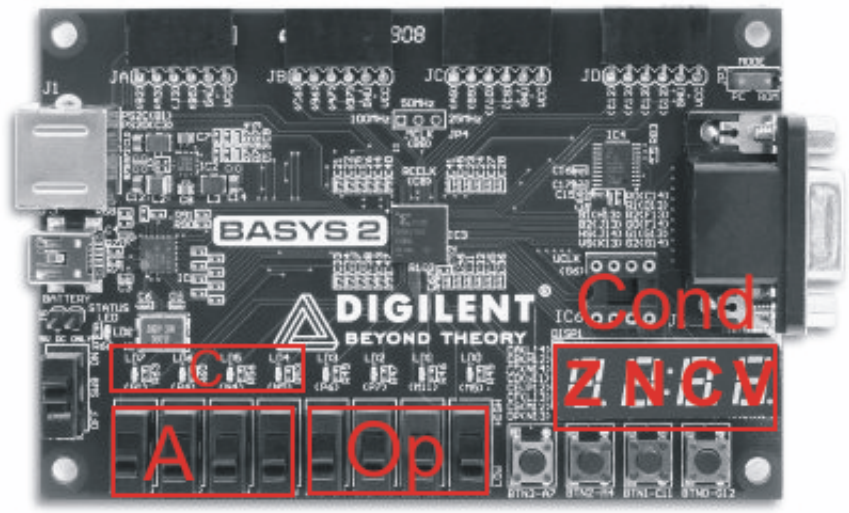

Figure 3: Inputs and output on FPGA board.

7 Deliverables

- Implement the modules in 3 files (`Mux32.v` ,`Register32.v`, and `ALU.v`) and compress them into a single file named `lab3.zip`. **Do not submit your testbenches or bit files.**
- A sample testbench for Register32 module will be provided to you. It is your responsibility to extend the testbench, and also to write a testbench for the other modules. You may share your testbenches on the newsgroup.
- Submit the zipped file through the COW system before the given deadline. The homework is supposed to be done with your group partner. Make sure both of you take roles in implementation of the project. Any kind of inter-group cheating is not allowed.
- Use the newsgroup metu.ceng.course.232 for any questions regarding the homework.
- You will make a demo of ALU module with the FPGA board the next week after the submissions (in your lab session hours). The exact dates and place will be announced later.