Assignment 2: CPU design

Set by: Dr Xiaojun Zhai (xzhai@essex.ac.uk)
Distributed to students: week 20
Submission deadline: see FASer
Feedback: three weeks from submission deadline
Submission mode: electronic only via FASer

Assignment objectives

This document specifies the second coursework assignment to be submitted by students taking CE869. This assignment is more challenging than the first one and it is meant to provide an opportunity to improve the knowledge of the VHDL language and, more importantly, to design a digital "system". You will be expected to learn to: a) implement digital system design in VHDL code; b) synthesise and download it to the target hardware; c) test, debug, and verify that the design meets the specifications; d) report about your design.

You are required to design code for your target hardware (a Digilent Basys3 board with a Xilinx Artix 7 FPGA) in order to implement a design that meets the specifications (below). You are required to submit working and correct code and you are strongly encouraged to use a modular coding style (allowing for greater flexibility, maintainability, modularity, and reusability). To show that you master all aspects of the language, your code should prevalently use concurrent statements for combinatorial circuits and sequential code for sequential circuits. Additionally, the use of non-standard packages (e.g. `STD_LOGIC_ARITH`, `STD_LOGIC_UNSIGNED`, `STD_LOGIC_SIGNED`) and `BUFFER` ports is forbidden, while the use of `INOUT` ports is accepted only when strictly necessary.

You are supposed to gain familiarity with VHDL coding during the supporting CE869 lectures and through self-study hours, also with the help of the recommended textbooks or any other book about VHDL. You are expected to work on this assignment mostly during lab hours. Your design project should be stored under the revision control repository that was assigned to you at the beginning of the course. You are supposed to commit often and describe your progress in the commit messages. In order to promote a learning scheme that values the learning process in addition to the submitted final design, your weekly progress (as traced back by the commit logs) will contribute to your assignment mark.

Design specifications

Your task for this assignment is to implement a 16 bit CPU. To make the assignment feasible within the time frame available for this module, the type of CPU will be fairly simple. In particular, the "program sequencing/control flow instruction" datapath can be modelled after the one on the left of Figure 1, while the "arithmetic/logic instruction" datapath can follow a structure like the one on the right in the same figure. Please notice that when `RAE` and/or `RBE` are low, the corresponding output(s) will simply match the input "`I`" to the register file. The opcodes for the instructions that the CPU is required to implement are given in Table 1.

To test your CPU, you will design a main entity that instantiates the CPU and connects it to the Basys3 peripherals. The sixteen switches of the Basys3 board will represent the input to the CPU while its output will be shown as hexadecimal number in the four digits of the 7-segment display. The central button will be used as reset signal to the CPU.

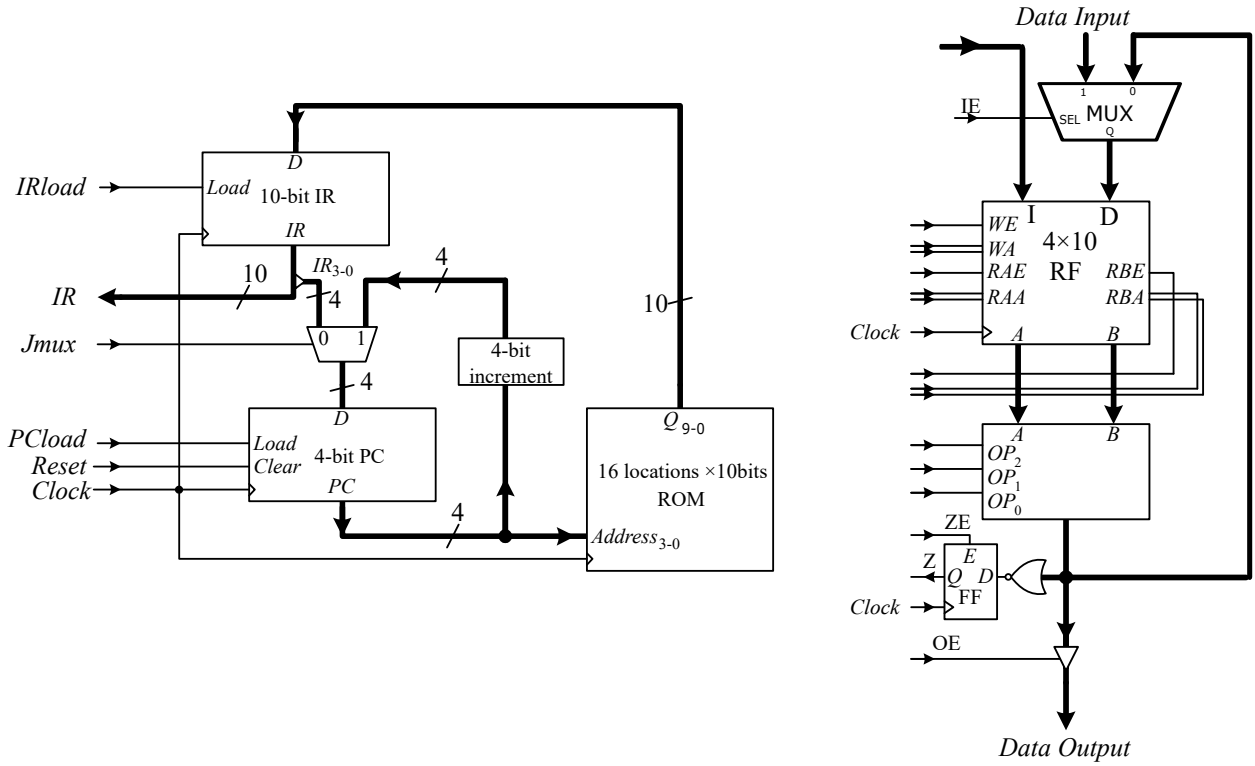

Figure 1: The figure shows the "program sequencing/control flow instruction" datapath (left) and the "arithmetic/logic instruction" datapath (right).

Instruction	Encoding	Affects	Operation
HALT	`0000000000`		Halt
MOV R*dd*, R*ss*	`000001ddss`		R*dd* ← R*ss*
IN R*dd*	`00001000dd`		R*dd* ← Input
OUT R*ss*	`00001001ss`		Output ← R*ss*
NOT R*dd*, R*ss*	`000011ddss`	Z	R*dd* ← NOT R*ss*
JMP *aaaa*	`000100aaaa`		Jump to *aaaa*
JNZ *aaaa*	`000101aaaa`		Jump to *aaaa* if Z status flag is unset (i.e. is 0)
JN *aaaa*	`000110aaaa`		Jump to *aaaa* if Z status flag is set (i.e. is 1)
LT R*rr*, R*qq*	`000111rrqq`	Z	Set Z to 0 if R*rr* < R*qq*, to 1 otherwise
INC R*rr*, #*nnnn*	`0010rrnnnn`	Z	R*rr* ← R*rr* + *nnnn*
DEC R*rr*, #*nnnn*	`0011rrnnnn`	Z	R*rr* ← R*rr* - *nnnn*
ADD R*dd*, R*rr*, R*qq*	`0100ddrrqq`	Z	R*dd* ← R*rr* + R*qq*
SUB R*dd*, R*rr*, R*qq*	`0101ddrrqq`	Z	R*dd* ← R*rr* - R*qq*
AND R*dd*, R*rr*, R*qq*	`0110ddrrqq`	Z	R*dd* ← R*rr* AND R*qq*
OR R*dd*, R*rr*, R*qq*	`0111ddrrqq`	Z	R*dd* ← R*rr* OR R*qq*
MOV R*dd*, #*nnnnnnn*	`1ddnnnnnnn`		R*dd* ← *nnnnnnn*

Table 1: Table of opcodes.

To test the CPU you will be asked to code three programs in the assembly and machine languages of the CPU, implementing the following tasks:

A. Given a nonzero number N as input, output the sum of the natural numbers less than N;

B. Given a number N as input, output "N div 11" (i.e. the integer quotient of the division between N and 11, "TRUNC(N/11)");

C. Given a number N as input, output N's largest divisor (excluding N itself) in hex. For example, if $N = 20$, the output should be A (the hex representation of 10); if $N = 35$, it should be 7; if N is prime, it should be 1. (A simple but not so efficient algorithm is to start from $N - 1$ and test if this number divides N, repeat for all numbers in decreasing order, and stop the first time the remainder of the division is zero.)

These design specifications should be interpreted as guidelines and should not constrain you from improving the CPU by doing modifications that you think would result in a better "product". The test programs above, though, should be implemented using only the instructions in Table 1. You are welcome to implement more elaborated programs to test the capabilities and the limitations of the CPU.

Report

You should write a report introducing the project and then describing your codings and designs for the assignment. It should first describe the design at a higher level and then detail the implementation of each module in a top-down fashion (rather than in chronological order). Also report if your code worked at the first attempt, what was wrong and how you fixed it. The repository log (if you used it properly) should help you considerably in this task. Your report should also include a discussion of design alternatives that you considered and the motivations for your final choice.

Your document should have a title page and be sub-divided into appropriately headed sections. It MUST contain references to material used in your work (e.g., VHDL code or information available in books or on the Internet). All of the VHDL code submitted should be included in the report in the form of code appendices and be typeset in Courier font (or a suitable fixed-width alternative font of your choice). Your report should consist of approximately 1500–3000 words of narrative (i.e. excluding references, code fragments, pictures, diagrams and schematics). Please write registration number and word count on the title page of your report.

Submission

Your work must be submitted to the university's online FASer submission system at the address https://faser.essex.ac.uk/ by the deadline given on the system. No other mode of submission is acceptable. You are strongly advised to upload a draft submission before the last lab hours prior to the deadline, and then update it up to the deadline.

You are required to submit one ZIP archive (not RAR or other formats) containing the following files:

1. All source files needed to synthesize your project (but not the temporary files created by Xilinx);

2. A report document submitted in PDF format. DO NOT SUBMIT THIS FILE IN .DOC OR .DOCX OR SIMILAR FORMATS - SUBMIT PDF.

3. Your repository log in .TXT format.

4. If you want (in your own interest, see next section), submit a .TXT or .PDF file with the transcripts of relevant forum discussions you contributed to.

DO NOT WAIT UNTIL CLOSE TO THE DEADLINE TO MAKE YOUR FIRST SUBMISSION. Difficulties with the submission system will not be accepted as an excuse for a missing submission.

Marking criteria

This assignment is worth 40% of the module mark. Marks will be awarded for the VHDL code, including coding style and quality, and for the descriptive document, including content, presentation, and discussions. In addition to the report, each students will be expected to demonstrate and explain her/his design with confidence and competence during a demo lab session. Marks will be assessed based on:

- Implementation
  - Quality of the implementation 30%
  - Modular design 6%
  - Generic and re-usable code 6%
  - Proper use of comments 6%
  - Steady progress, adequate use of GIT 6%
- Report
  - Correctness and completeness of report 12%
  - Clarity of presentation 12%
  - Organisation of report 12%
  - Quality of diagrams and schematics 8%
- Others
  - Compliance with submission instructions 2%

Marks below 100% can earn additional credits if the student actively engaged in forum discussions asking pertinent questions and giving competent answers to questions raised by classmates.

Late Submission and Plagiarism

Please refer to the Students' Handbook for details of the Departmental policy regarding submission and University regulations regarding plagiarism.

Revision 1.0
10/12/2020
Xiaojun Zhai