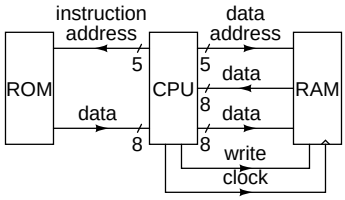
Assignment 2
RTL Design

Part 1 due Monday, November 6, 2000
Part 2 due Monday, November 13, 2000

Introduction

In this assignment you will design and test a simple computer. The computer is composed of a CPU, a ROM and a RAM:

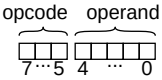


The ROM and RAM each store 32 words of 8 bits (1 byte).

The CPU has two registers: a 5-bit program counter (PC) and an 8-bit accumulator (A). The CPU design therefore contains:

- the PC datapath
- the A datapath
- the controller (called the "instruction decoder")

Note that this CPU has independent data and instruction memories (this is called a "Harvard" architecture). The ROM is the instruction memory and is addressed by the PC. The RAM is the data memory and is addressed by the operand field of the current instruction. The output of the instruction memory is an instruction encoded as a 3-bit operation (opcode) in the most significant (MS) 3 bits and a 5-bit operand field in the LS 5 bits:



The CPU can execute eight different instructions, described below.

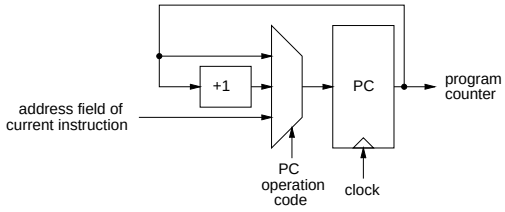
The design of this CPU allows one instruction to be executed per clock cycle. This means that the instruction decoder is a combinational logic circuit, not a state machine.

CPU Description

PC Datapath

The program counter (PC) datapath updates PC as instructed by the controller. PC can be loaded with: (1) PC, (2) PC+1 (to point to the next instruction), (3) the current instruction's operand field (to branch to another instruction), or (4) zero (to reset the computer).

The structure of the PC datapath is :



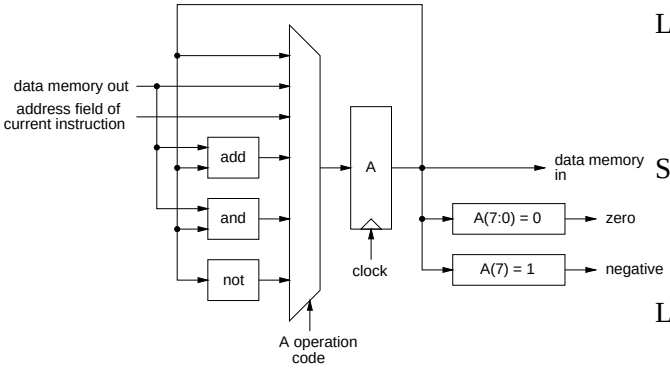
The PC datapath entity inputs are: the operand field of the current instruction, the PC datapath control signal, a reset signal, and the clock.

The PC datapath entity output is: the PC register value (the current value of the program counter).

A Datapath

The accumulator (A) datapath updates the accumulator as instructed by the controller. A can be loaded with: (1) A, (2) the data memory output, (3) the operand field of the current instruction, (4) the result of an 'add', 'not', or 'and' operation on the data memory output and A.

The structure of the ALU datapath is:



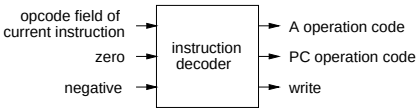
The ALU datapath entity inputs are: the data memory output, the operand field of the current instruction, the A datapath control signal, and the clock.

The ALU datapath entity outputs are: the A register value, a signal that is asserted when A is zero, and a signal that is asserted when the MS bit of A is set (A is negative).

Instruction Decoder (Controller)

The instruction decoder uses the current instruction (the instruction memory location addressed by PC) to control the A and PC datapaths.

The structure of the controller is:



It's inputs are: the MS 3 bits of the instruction memory output (the current instruction), and the zero and negative status signals from the ALU.

It's outputs are: the A datapath control signal, the PC datapath control signal, and the RAM read/write control signal.

Design the controller so that the CPU can execute the following instructions:

	code	description
LOAD	000	load - load A from the data memory location addressed by the instruction operand field
STORE	001	store - store A in the data memory location addressed by the instruction operand field
LOADI	010	load immediate - load A with the current instruction's operand field (the MS 3 bits are cleared)
ADD	011	add - load A with the sum of A and the data memory location addressed by the instruction operand field
NOT	100	not - load A with the one's complement of A
AND	101	and - load A with the AND of A and the data memory location addressed by the instruction operand field
JZ	110	jump on zero - if A is zero load PC with the current instruction's operand field
JN	111	jump on negative - if A is negative load PC with the current instruction's operand field

Note that an instruction's 5-bit operand field can be used for different purposes depending on the instruction: to load A with immediate data (LOADI), to load PC with an instruction memory address (JZ and JN instructions), ignored (NOT), or to address data memory (other instructions).

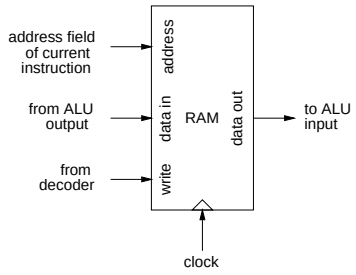
Do not use the opcode itself as the PC and A control signals.

Memory Description

RAM

The data memory is a RAM with the following inputs: "data memory in", address, read/write control signal, and clock. The only output is a "data memory out" output. Use an array of data words and the VHDL array indexing operator (( )) to implement the RAM.

The structure of the RAM is:



ROM

The instruction memory is a ROM with a 5-bit address input and an 8-bit output (the instruction). Use a selected assignment statement to implement the ROM. Note that you will not have to define the contents of all locations (see below).

The structure of the ROM is:

The following example program loads the accumulator with -3 (0FDH), increments it three times until it becomes non-negative and then loops forever on the last line. The code fragment shown below has been assembled into a format suitable for including in a selected assignment:

```
-- Instruction   Address   Opcode  Operand
   "01000000" when 0,  --  LOADI   0
   "00100000" when 1,  --  STORE   0
   "01000001" when 2,  --  LOADI   1
   "00100001" when 3,  --  STORE   1
   "01000010" when 4,  --  LOADI   2
   "10000000" when 5,  --  NOT
   "01100001" when 6,  --  ADD     1
   "11100110" when 7,  --  JN      6
   "11001000" when 8,  --  JZ      8
```

Use the above program it to design the instruction memory. Unused locations should be set to zero.

Assignment

Data Types

Create a package that defines types and constants for your design. Define data types for instruction and data words. Each should be 8 bits wide. The data words should be subtypes of `unsigned`, the instructions should be subtypes of `std_logic_vector`.

Define a constant for each of the opcodes.

Define types and constant for the PC and A datapath control signals. Enumerated types could be used but they are not supported by the MaxPlusII waveform editor.

Design

Write VHDL entities and architectures for each of the five parts (A and PC datapaths, controller, RAM and ROM). Each entity should `use` the above package.

Testing

Test each part of your design *separately*. As a minimum, ensure the following:

- the instruction memory gives the correct output for each address input
- if you write the values 1, `0FFH` `0EEH` to memory locations 0, 1 and 31 you can read back the same values from these memory location.
- the PC can be reset, loaded, and incremented
- the ALU performs each of its six operations correctly (show at least one example of each operation)
- the controller generates the correct outputs for each instruction, including the two possible outputs for each conditional jump instruction

Write a top-level entity that combines the five components. This entity should have two inputs: reset and clock, and three outputs that allow you to monitor the operation of the computer: PC, the instruction memory output, and A. Simulate the operation of your computer from reset until it executes two iterations of the infinite loop (approximately 11 instructions).

For Part 1, submit the VHDL source code listings and simulation outputs for the ROM, RAM and PC datapath. For Part 2 submit listings and simulation outputs for all parts (the five individual tests and the test of the complete computer).