Columbia University
Department of Electrical Engineering
EECS E4340. Laboratory #1.
PDP-8 Dataflow Design.
Due: March 3, 2004

This is the first assignment on the road to a complete PDP-8 implementation in hardware. This assignment should be completed in groups of two. Only one assignment should be handed in per group.

In this lab, you will finish the register-level transfer description of the PDP-8 dataflow that we went over in class. As you progress in the design of your machine, you will be completely free to modify this datapath to implement design improvements. The Cadence library `pdp8` in
`/tools2/courses/ee4340/cdslibs/pdp8`
contains the schematics and components for this design. You will want to copy over this library as the starting point for this lab.

1 Specifications

Given the following additional specifications, you are to complete the VHDL architectures of each of the components to produce a functional PDP8 datapath design. The schematic for the datapath is given in Figure 1. We will assume that all the flip-flops in the design are positive-edge triggered.

- `ac_link`. This is a 13-bit rotator in which the link (as the most-significant bit position) and accumulator are rotated together. When `cll` is high, the link is cleared When `cml` is high, the link is complemented. When `right` is high and `left` is low, the accumulator-link rotates once right. When `right` is low and `left` is high, the accumulator-link rotates once left. When `right` and `left` are both low, the accumulator-link holds its current value. When `right` and `left` are both high, the accumulator takes in a new value on the next positive edge of the clock.

- `twelve_bit_reg_hold`. These are all 12-bit registers that take in new data on a rising clock edge when the `load` signal is high, else they hold their current value.

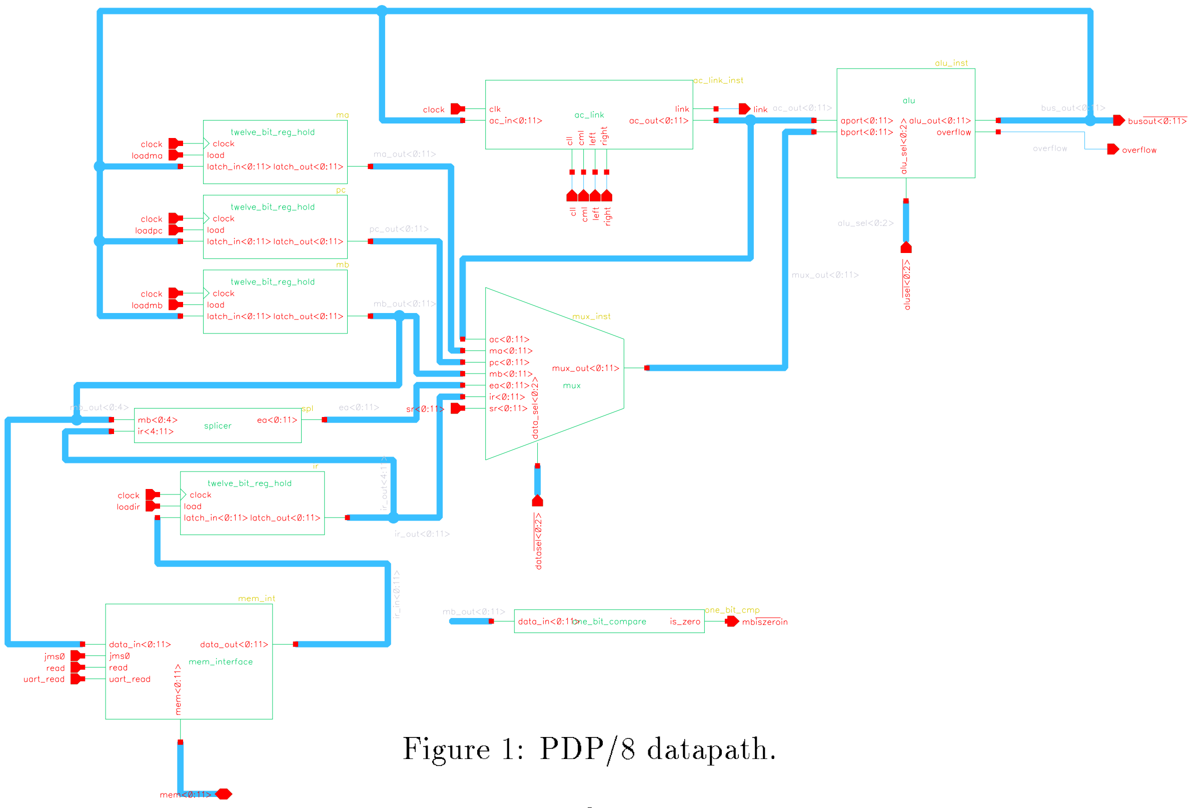
Figure 1: PDP/8 datapath.

- `alu`. This macro performs all of the main logical and arithmetic operations of the PDP8. `data_sel<0:2>` is encoded as follows:
  - `alu_ADD`. "000". Performs a two's-complement addition of `aport` and `bport`, indicating an overflow with `overflow`.
  - `alu_INC`. "001". Increments `bport`.
  - `alu_AND`. "010". Performs a logical AND of `aport` and `bport`.
  - `alu_ZERO`. "011". Puts all zeros on the output.
  - `alu_OR`. "100". Performs a logical OR of `aport` and `bport`.
  - `alu_MUX`. "101". Passes the value of `bport` to the output.
  - `alu_NOTMUX`. "110". Passes the logical complement of `bport` to the output.
- `mux`. This is a simple multiplexer with the following encoding for the select lines:
  - `sel_AC`. "000"
  - `sel_MA`. "001"
  - `sel_IR`. "010"
  - `sel_MB`. "011"
  - `sel_PC`. "100"
  - `sel_EA`. "101"
  - `sel_SR`. "110"
  - `sel_NULL`. "111". Don't care. Mux output not used.
- `splicer`. This macro is used to from the effective address by combining the page offset in the instruction with the page address (either zero or the address of the current page). If `ir<4>` is high, then `mb<0:4>` is concatenated with `ir<5:11>` to form the effective address. If `ir<4>` is low, then `"00000"` is concatenated with `ir<5:11>` to form the effective address.
- `one_bit_compare`. The output is zero is the inputs are all zero. This is used for the `ISZ` instruction.

- `mem_interface`. If `jms0` is high, then the `mem` drivers are tristated and `data_out` is driven to `"100000000000"`. This is used to force a subroutine call to `"0000"` on an external interrupt. If `read` is high, then the `mem` drivers are tristated and `data_out` is driven to the value on the `mem` bus, otherwise, `mem` is driven to the value on `data_in`.

2 Functional verification

The bulk of the work in this assignment is actually in *functional testing* of your design. In this assignment, I want you to put a lot of effort into designing a good testbench for your datapath. While exactly how you do this is up to you, I offer a few guidelines:

- You want to make sure you exercise all of the ALU functions and all of the MUX paths and perhaps simulate the execute cycles of several instruction.
- We sure to exercise the `mem_interface` component with read and writes to memory.
- Create a test sequence using only the switch register as input, one that exercises as much of the function of the datapath as possible.

These last two items are most important, because this will allow you to use much of your testbench in the debug of the real hardware (that is, when you have programmed the dataflow, but have not yet programmed the controls, but want to make sure your datapath is functioning correctly.)

3 What to turn in

Turn in the VHDL for each of your macros. Also turn in your testbench, waveform plots, or other outputs that substantiate your testing. Include a brief descrition of your testing methodology.