

Spring 2005
CSE 141L Projects in Computer Architecture
Lab 3: Construct & Test Data Path for 8-bit Processor

Reports and electronic submissions are due at the beginning of class on Tuesday, May 17th.

In this lab, you will design a data path for the 8-bit processor architecture you developed in Lab 1 and test its functionality.

You will implement all instructions in your architecture except for the control transfer instructions (branch, jump, etc.) and memory load/store instructions.

What you will turn in for this Lab

Written Report Only:

- Summary of your ISA from Lab 1.
- Printed schematics for the top level data path as well as all the lower level modules you designed in LogicWorks 5.
- Printout of waveforms that show example input data, ALU opcode and the result from the ALU for each instruction.
- Answers to the following questions.

Electronic Turn-in Only:

- All the LogicWorks 5 files you created (to be submitted using turnin script).
- Description of the procedure to follow in order to test the operation of the data path for each instruction for your processor. This is so your TA can test your design.

Working knowledge of LogicWorks 5 is assumed!

We will not review LogicWorks 5 in class. You should be familiar with this tool in general and following items in particular:

- Various components available in Standard Libraries such as: D Flip-flop, logic gates, multiplexors, registers, adders, clock, binary switch, binary display, hex keypad, hex display, etc.
- How to connect a bus to various components.
- How to define a sub-circuit bottom up, i.e. create a circuit and then use it to define the pins on the parent symbol.
- How to simulate a circuit and generate waveforms.
- How to print a circuit and waveforms.

Design of the data path

A data path is a collection of registers and logic elements through which the data flows during the operation of a processor. Various elements of the data path are generally controlled by logic in a separate control block.

You will design the data path for your 8-bit processor. This will include the register file, any special registers (except the PC, which will be implemented in Lab 4), ALU and basic control logic for the data path. You will take the opcodes for your instructions and generate appropriate ALU opcodes. For example, if you have an ADD instruction it will be implemented through the data path and will have a corresponding ALU opcode. Note that you **must** implement an instruction to load a register with a constant value and an instruction to move the contents of one register to another register through the data path.

The control logic for the data path will take the ALU opcode and generate signals necessary to control the function of the ALU. You will also appropriately connect hex keypads, binary switches, hex displays and binary displays to test your data path.

The figure on page 4 shows an example of 8-bit data path for a specific architecture. Your ISA will have different characteristics and hence your data path may differ significantly from the example shown in the figure on page 4.

Your data path may have internal storage consisting of a register file, stack, accumulator and/or special registers. You should set up a mechanism appropriate for your architecture to supply operands to the ALU from internal storage. You must also provide a mechanism to load immediate values (i.e. constants) to the internal storage.

An example data path is shown in the figure on page 4. Various components are:

- The register file has one write port (address $WA[1:0]$, write enable WE , data $D[7:0]$) and two read ports A (address $RA[1:0]$, data $A[7:0]$) and port B (address $RB[1:0]$, data $B[7:0]$).
- The inputs to the ALU are a source operand ($S[7:0]$), destination operand ($D[7:0]$) and 3-bit opcode ($OP[2:0]$).
- The output of the ALU is result ($R[7:0]$) and a status flag ($FLAG$) as well as a write enable for the flag ($WRFLAG$). The architecture for which the data path is depicted in the figure has two variants of compare instructions that set the $FLAG$ depending on the result of the comparison. Furthermore, the $FLAG$ is left unchanged if the instruction being executed is one other than a compare instruction. This is accomplished by providing a $WRFLAG$ signal, which is used as the write enable to the register associated with $FLAG$.
- A 2:1 multiplexor ($MUX8X2$) provides a means to supply immediate data to the ALU. A particular register can be set to a constant by using the immediate data, a MOV opcode, asserting write enable high and setting appropriate address on the write port of the register file.

You should design a data path that is capable of executing all the instructions in you ISA, except for the control transfer instructions and memory load and store instructions. Furthermore, it should have a provision for memory load and store instructions (you don't need an interface to the data memory).

You should connect binary switches and hex keypads to provide appropriate stimulus and also connect binary and hex displays to show the results at various points in your design.

In the figure on page 4, REG8CLKL is an 8-bit register that registers data on the negative edge of the clock and is used to display the output of the ALU when it becomes stable.

In the printout, you should show all the internal schematics for each one of the circuits in your design. Also, your waveforms should show following features of your data path:

- Ability to load a constant with the required number of bits for your ISA into any appropriate register (e.g. any general purpose register).
- Correct operation of the ALU for all the ALU opcodes supported.
- Ability to have the same register to be the source and destination of an instruction.

Show the ALU operation for interesting input data. For example, if the carry from the adder is saved in your architecture, use data that both does and does not set the carry bit.

In order to make it easy for the TAs to grade your report, organize and document your report well.

Questions

1. Which instruction is the most expensive in terms of number of gates it requires (no need to give the exact gate count, just give the reasoning).
2. What tricks did you use to decrease the logic in your data path by sharing the logic among more than one instruction?
3. How does your "move constant to register" and "move register to register" instruction work? Is it a special case of another ALU instruction, or does it use special data path elements?
4. Using your data path, explain how you will load a register with a value from a memory location and store contents of a register to a memory location.

CPU8_DP

8-bit CPU Datapath

ALU Operations

