

# CDA 3101: Introduction to Computer Hardware and Organization

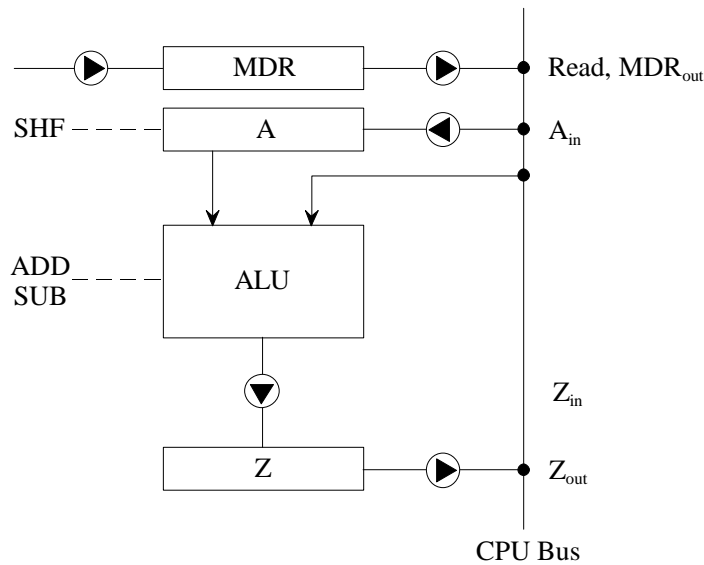
## Fall Term, 2003

### Lab 5: CPU/ALU

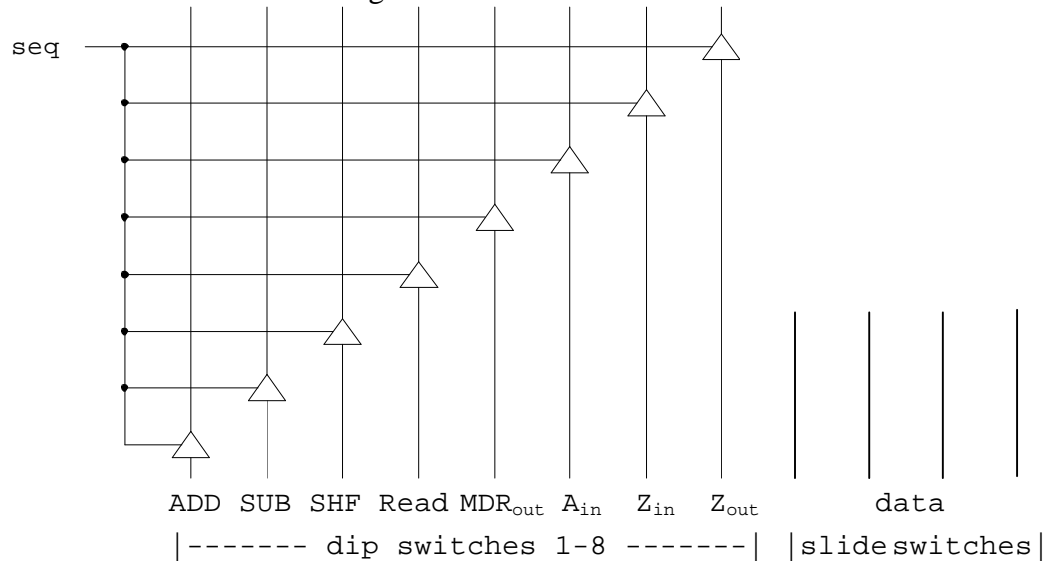
Due Date:

Monday, December 10, 2003 (with the final exam)

- Use three 7495 chips to represent 4-bit registers (call them registers MDR, A, Z). Register MDR should be loadable from the slide switches. The contents of registers A and Z should be constantly viewable on the light bank of the trainer.
- Implement a 4-bit full adder using a 7483. Don't forget to provide for the carry-in.
- Implement the simple CPU described below:



8 bit microcode control using a DIP switch bank:



Implement "seq" with a debounced switch. "seq" should also control your registers. You will probably want to use three-state logic to implement the two "out" transfers. You will probably want to use gate logic to hold the "load" line 0 on each register except when doing an "in" transfer. Set up a dip switch bank in the same manner as in lab 4 to provide 0/1 signals for your microcode. Be very careful in using three-state logic. For the disconnect state of a tri-state buffer, if the output side is connected to a TTL input, the signal on this side will usually rise to the TTL default of high, potentially generating undesirable side effects. You can, of course, avoid this situation in your design by using intervening gates (e.g., a NOT gate) where there is potentially a problem. If you are sparing in your use of 3-state logic you are unlikely to encounter such problems.

By manually setting the microcode, you should be able to execute any one of the commands

LOAD A	(Read)
	(MDR <sub>out</sub> , A <sub>in</sub> )
ADDtoA	(Read)
	(MDR <sub>out</sub> , ADD, Z <sub>in</sub> )
	(Z <sub>out</sub> , A <sub>in</sub> )
SUBfromA	(Read)
	(MDR <sub>out</sub> , SUB, Z <sub>in</sub> )
	(Z <sub>out</sub> , A <sub>in</sub> )
SHIFT A (SHF)	

Note SUB may be accomplished simply by inverting the signals from the CPU bus and setting the carry in to your adder to 1 (for ADD the carry-in should be 0). Be sure to leave your lights hooked up on A and Z so you can determine if your CPU is working. You do not need to worry about overflow conditions.

Write a report ordered as follows:

- \* cover page with group number and team member signatures
- \* completed [certification page](#) for each team member
- \* concise problem statement stating the purpose of the exercise and how it is to be done
- \* equipment explanations: equipment required and staged, carefully explained schematic diagrams
- \* technical explanations: for this lab you will need to explain your CPU design and its microcoding. What other instructions could be added (e.g., increment by 1, or decrement by 1) and how would you accomplish them? In your last lab, you constructed a memory unit. How would you take data from your memory bus (bus sense register) and get it to your CPU bus and vice-versa? also, how would you synchronize the transfer of data between these two units (assume independent clocks govern each)? Additionally, for this lab, you are the instruction interpreter and microprogram store. What else is needed in the CPU design to automate these functions?
- \* technical observations: recapitulation - please do not recite your implementation problems except as they lead to discovery; you are experimenting; in this lab, you have fairly straight forward circuitry that must be controlled in a precise manner. What happens if you are careless in setting your microcode controls? Where is 3-state logic necessary and where can combinational logic suffice in providing "in-out" controls? can you make your CPU do things not directly reflected in the instruction sequences given above? what were your design alternatives? (and why did you go the way you did), etc.
- \* technical conclusions: observations which allow you to reach definitive conclusions should be concisely summed up; for this lab, the theory predicts you should be able to employ register-transfer principles to provide a controlled environment which enables sequences of actions corresponding to ordinary computation (add, shift, subtract, etc). Is the theory supported by the lab? Can you make any statement regarding the relative simplicity of this CPU compared to one which has the added complexity of instruction interpretation, control store management, I/O, etc? Have you discerned enough regarding CPU internals to build one to perform simple arithmetic with even less human intervention?