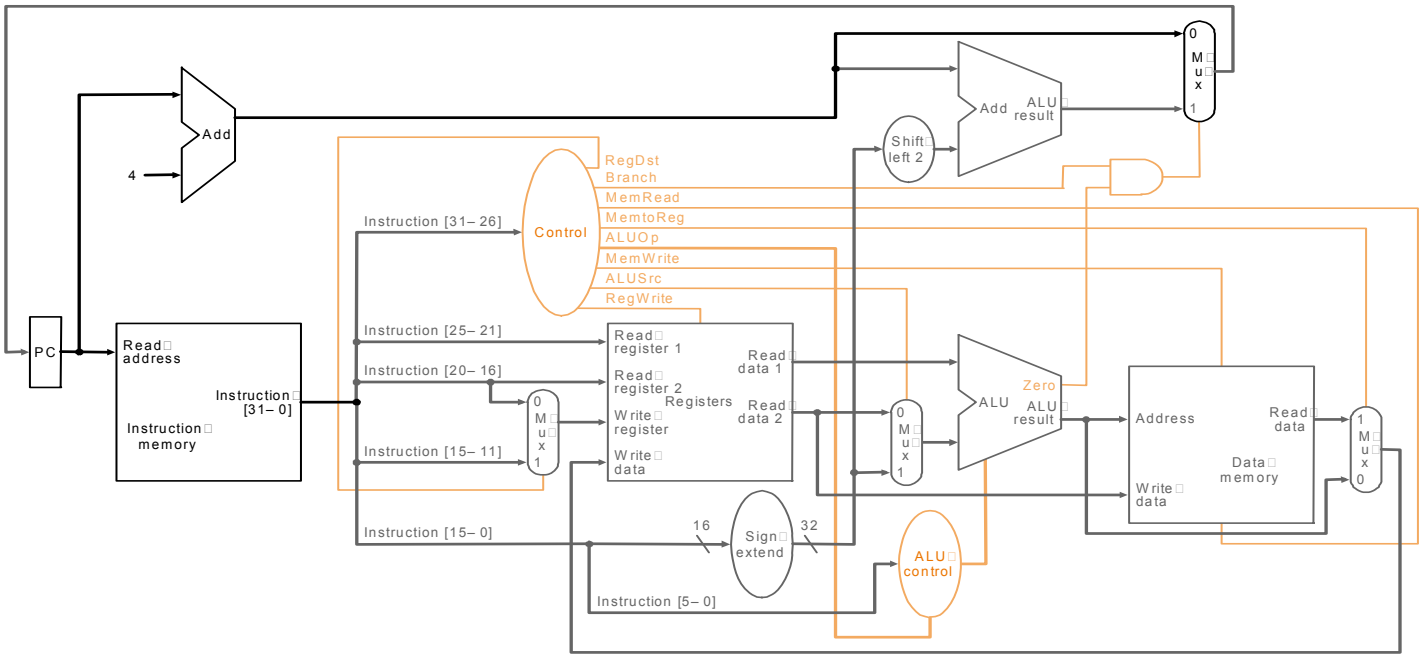
CPRE 381 Lab 7
Single-Cycle Datapath Implementation

In this lab you will construct a full single-cycle datapath based on components built in previous labs.

Part 1 – Control



The figure above shows the basic datapath you will implement. The control logic has been highlighted. Below is an abbreviated truth table for the Control unit. Assume the op codes are 000000, 100011, 101011, and 000100 for R-format, lw, sw and beq respectively.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Below is the truth table for the ALU Control unit including don't care conditions. It takes both the inputs from the Control unit as well as the function field for R-Type instructions.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

1. Design the Control and ALU Control units in verilog.
2. Implement your designs.
3. Test and verify each component to show that it matches the truth tables for all operation types.
4. Verify in the log files that your designs do not use any registers or latches. They should be combinational only.

Part 2 – Assembling the Datapath

Once you have tested and verified your control logic, you can assemble the full single-cycle datapath from the figure above. Below is an itemized list of components with hints and instructions for assembly.

1. Complete a design for the full datapath choosing your datapath width.
2. Implement the datapath design in verilog.
3. Create a list of the MIPS instructions that can be implemented in this datapath.
4. Write a small program that will demonstrate the functionality of all instructions.
5. Convert the program instructions to their binary equivalents and create a memory initialization file (MIF) used to initialize the instruction memory.
6. Create a MIF file for the data memory to contain any variables you need to initialize to evaluate your instructions.
7. Simulate and verify the functionality of the list of instructions on the datapath design.

Program counter
The program counter is the main sequential device that drives the entire datapath. It is simply a register that must address the instruction memory. For this implementation we will use a small instruction memory of 256 words. Size your program counter appropriately.

Adders
There are two adders shown in the datapath of the program counter. They are shown with ALU symbols, but always perform addition and therefore you can use an adder in place of a general purpose ALU.

Optimization notes: if your instruction memory is word addressable, you can increment by 1 instead of 4 and the shift-left-2 unit can be eliminated. Further optimization can be applied by combining the first adder, PC register and the branch multiplexor into a single unit (up-counter with a load input). This approach could be used even for the version where addresses are incremented by 4 if you recognize that the same logic applies, and the two least significant bits are always grounded (zero).

Shift Left 2
Note that you will not need to implement this device for your datapath because you are to make your memories word addressable instead of byte addressable. However, it is important to understand that this type of shift does not require any logic. The two

least significant bits are simply grounded (set to zero) and the higher order bits are simply routed from the inputs ignoring the two most significant bits on the input.

Instruction Memory
Use the memory wizard to instantiate a memory device to be used for instructions. Create a device with 256 words, each 32 bits wide. Make the device word addressable to match your program counter. You can select either a ROM or RAM device but assume that you will only ever load the contents from a memory initialization file (MIF) as in the memory lab, and the contents will not be changed within the datapath.

Sign Extend
A sign extension is also a very simple unit requiring no special logic. It simply repeats the MSB for all of the upper bits in the word (i.e. all 1's for a 1 MSB and all 0's for a 0 MSB). Connecting the MSB input to all of the upper bit outputs can create this functionality. For your datapath assume that only an 8-bit immediate is available (instruction[7:0]) and apply your sign extension appropriately.

Register File
Reuse the register file you designed in a previous lab. You may extend it to handle 32 registers and 32 bit words as MIPS supports, or you may organize your datapath to support 16-bit values and use only 16 registers. (Instruction length and data width need not match.) If you extend your register file, the address bits from the instructions will be the same as in MIPS. If you do not, you may simply ignore the MSB of each address field in the instruction.

General Purpose ALU
Reuse the ALU designed in a previous lab. Size the ALU to match the datapath width of your register file.

Data Memory
Instantiate a memory module as in a previous lab to store data. Make the width of the data match that of your ALU and register file. Make the memory word addressable with 256 words of memory.

Control and ALU Control
Use the components designed in Part 1 of this lab.

Various multiplexors
Size appropriately for the selected datapath width.