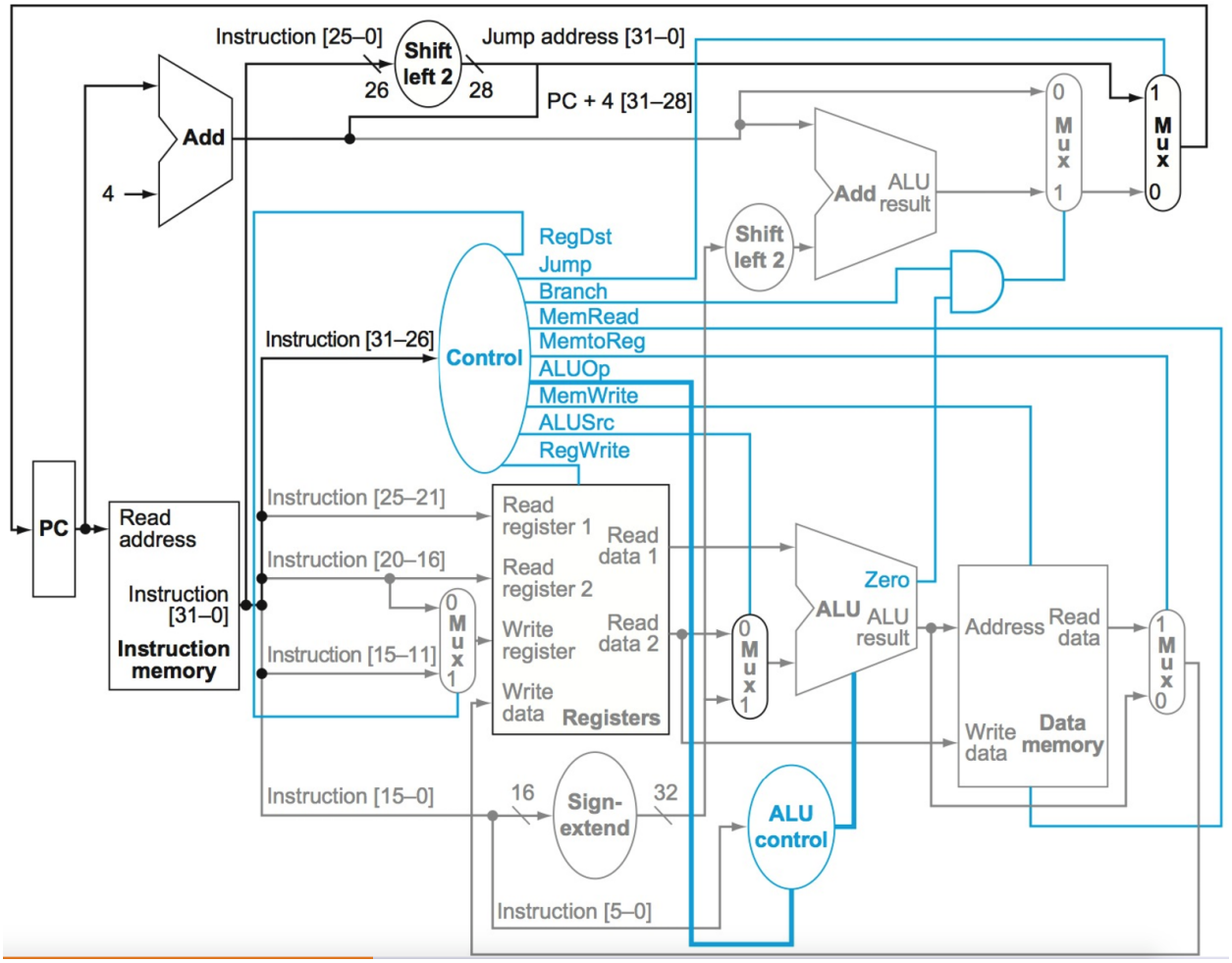
Homework 4

ECE2500 CRN:82943

Jacob Abel

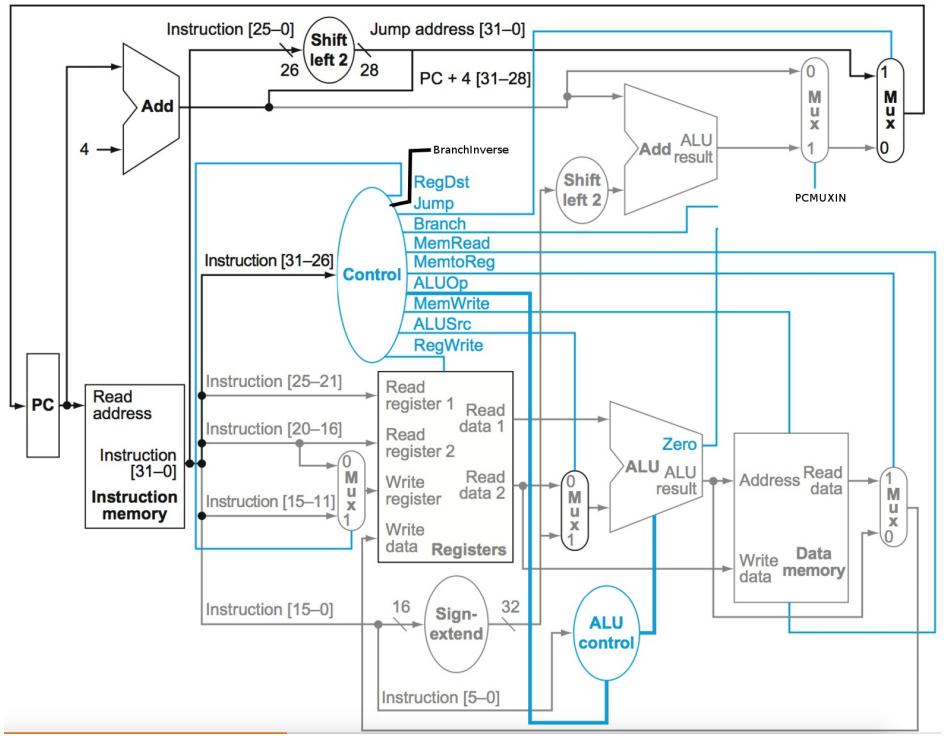
October 17, 2018

Single Cycle Data path

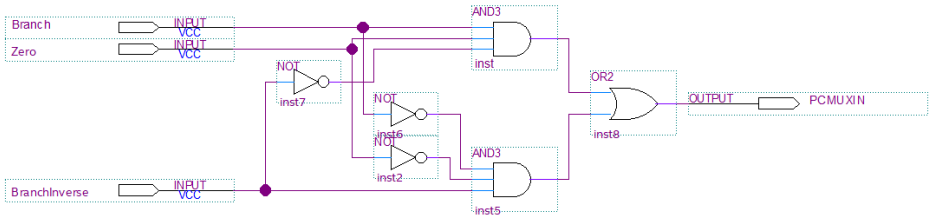


Implements `add`, `sub`, `beq`, `lw`, `sw`, `j` instructions

Problem 1: Consider the single-cycle data path given on the first page which implements the add, sub, `beq, lw, sw, j` instructions. Modify this data path to support the execution of a `bne` instruction, in addition to the other instructions already implemented. You are free to add/remove/modify multiplexers, ALU, adders, logic gates, shift units, and control signals. Your answer should consist of a concise description of the changes you will make to the data path as well as a figure that reflects the modified data path. You may draw only the part of the figure that is actually changed by your design.



Modified Data Path



Added Component

The change is just replacing the and gate with a slightly more complex boolean logic circuit and adding a control out for when the branch comparison needs to be negated. The logic block guarantees that if there is a branch, the primary branch and inverse branch gates will never override each other.

Problem 2: Consider the execution of the following instructions in the modified data path solved in Problem 1. For each instruction, write down the correct values for all the control signals `RegDst, Jump, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite`, and any control signal(s) added in Problem 1.

1. `sub $s0, $s1, $s2`
2. `j 0xF43C`
3. `lw $t0, 4($s1)`
4. `bne $s0, $t0, LABEL`

All table entries in binary

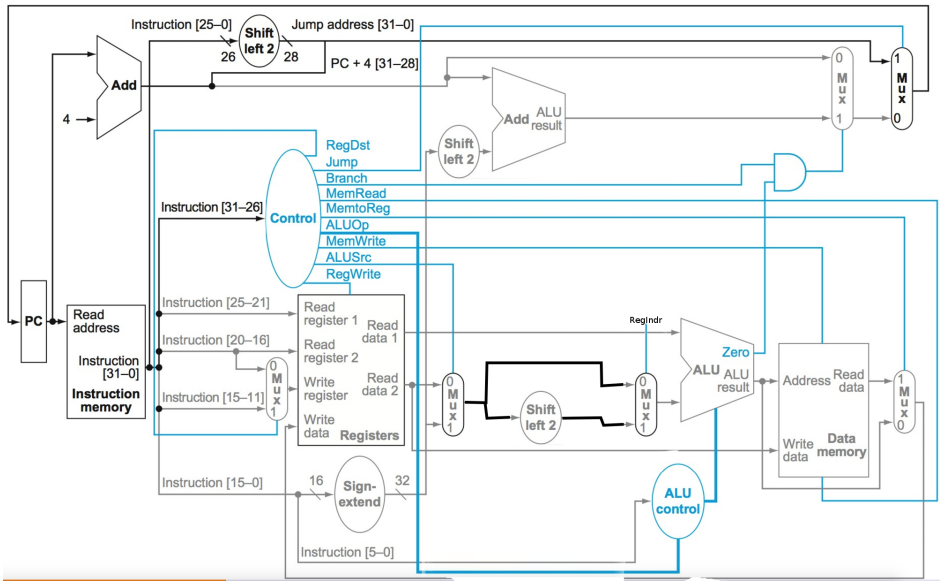
Instruction	RegDst	Jump	Branch	MemRead	MemtoReg
`sub $s0, $s1, $s2`	1	0	0	0	0
`j 0xF43C`	X	1	X	0	X
`lw $t0, 4($s1)`	X	0	0	1	1
`bne $s0, $t0, LABEL`	0	0	1	0	X

Instruction	ALUOp	MemWrite	ALUSrc	RegWrite	BranchInverse
`sub $s0, $s1, $s2`	10	0	0	1	0
`j 0xF43C`	XX	0	X	0	X
`lw $t0, 4($s1)`	00	0	1	1	0
`bne $s0, $t0, LABEL`	01	0	0	0	1

Problem 3: Imagine that you are a hardware designer who wants to add a new instruction to the MIPS ISA. Our new instruction has the following syntax: `lwr $rd, $rt($rs)`, the load word register instruction, which loads a word from memory into a destination register using a register as an offset (convenient!). Assume that we have found a way of encoding this instruction into 32-bit machine code using the R-type encoding format. We now want to modify the single-cycle data path in Figure 1 to support the execution of this instruction.

1. Which new blocks (if any) do we need to add to support execution of this instruction? Please describe briefly and draw a figure to reflect the new design. Your goal must be to make minimum modifications to existing hardware.

   A mux is added in between the mux and ALU input and a shift left 2 is added to this mux.



Modified Data Path

2. What new control signals (if any) do we need from the control unit to support this instruction?

   RegIndr: Signifies a indirect + register offset

3. Write the values for following control signals as well as any new control signals added: `RegDst, Jump, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite`.

   All values listed in binary

   RegDst 1

   Jump 0

   Branch X

   MemRead 1

   MemtoReg 1

   ALUOp 10

   MemWrite 0

   ALUSrc 0

   RegWrite 1

   RegIndr 1

Problem 4: Describe the advantages and disadvantages of single cycle versus pipelined data paths. Describe how changes (increases/decreases) in memory performance would affect each data path type.

Single cycle is much simpler and easy to design/debug. Additionally, as all instructions complete in one cycle, it is easy to program for as well. Unfortunately this also means that every instruction takes as long as the slowest instruction and you can't have multiple instructions computing at once (pipelining).

As memory is typically one of the slowest instructions, if you increase or decrease the memory speed, single cycle more or less directly increases/decreases in speed. In pipelined data paths, increasing/decreasing the memory speed will still increase/decrease the overall performance of the path but to a significantly smaller degree as in pipelined data paths, memory is not as much of a bottleneck.