CSE352 Spring 2013 Homework Assignment #7 - Solutions (updated on 6/12/13)

Q1)

(a) sllv

Similarly to sll, first we need to modify the ALU and the ALU control logic. Instead of shifting by shamt[4:0], we shift by A[4:0]:

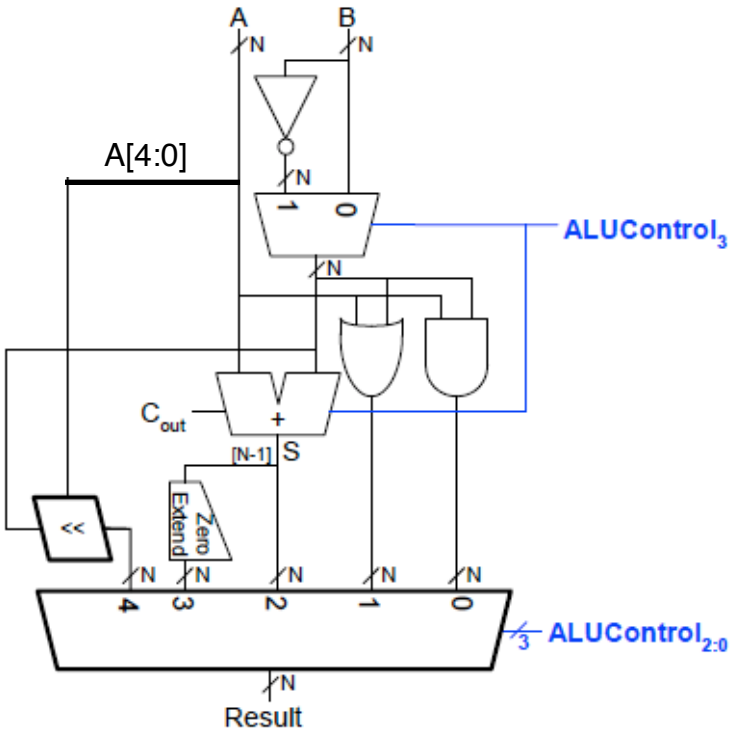


FIGURE 7.1 Modified ALU to support `sll`

$ALUControl_{3:0}$	Function
0000	A AND B
0001	A OR B
0010	A + B
0011	not used
1000	A AND $\overline{B}$
1001	A OR $\overline{B}$
1010	A - B
1011	SLT
0100	SLL

TABLE 7.1 Modified ALU operations to support `sll`

ALUOp	Funct	ALUControl
00	X	0010 (add)
X1	X	1010 (subtract)
1X	100000 (add)	0010 (add)
1X	100010 (sub)	1010 (subtract)
1X	100100 (and)	0000 (and)
1X	100101 (or)	0001 (or)
1X	101010 (slt)	1011 (set less than)
1X	000110 (sllv)	0100 (shift left logical)

TABLE 7.2 ALU decoder truth table

No need to add control signals as this instruction is an R-type instruction.

(b) sltiu (not accounted for)

As in slti, the datapath doesn't change.

As in slti, we add a new entry in the ALU decoder table - [ALUOp : 11] - in order to tell the ALU to perform a "set less than" operation: Out = (A-B) < 0 ? zeroext(1) : zeroext(0)

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
10	100000 (add)	010 (add)
10	100010 (sub)	110 (subtract)
10	100100 (and)	000 (and)
10	100101 (or)	001 (or)
10	101010 (slt)	111 (set less than)
11	**X**	**111 (set less than)**

TABLE 7.4 ALU decoder truth table

The Main Decoder table entry for sltiu is the same as in slti.

Instruction	opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
sltiu	**001010**	**1**	**0**	**1**	**0**	**0**	**0**	**11**

TABLE 7.5 Main decoder truth table enhanced to support slti

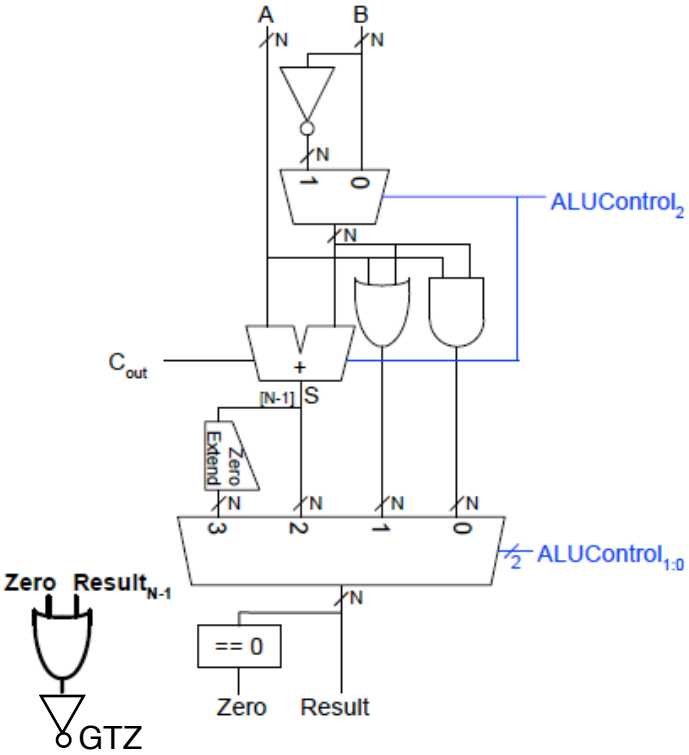
Note on the difference between slti and slitu:

> *The MIPS32 Instruction Set states that the word unsigned as part of Add and Subtract instructions, is a misnomer. The difference between signed and unsigned versions of commands is not a sign extension (or lack thereof) of the operands, but controls whether a trap is executed on overflow (e.g. Add) or an overflow is ignored (Add unsigned). An immediate operand CONST to these instructions is always sign-extended.*

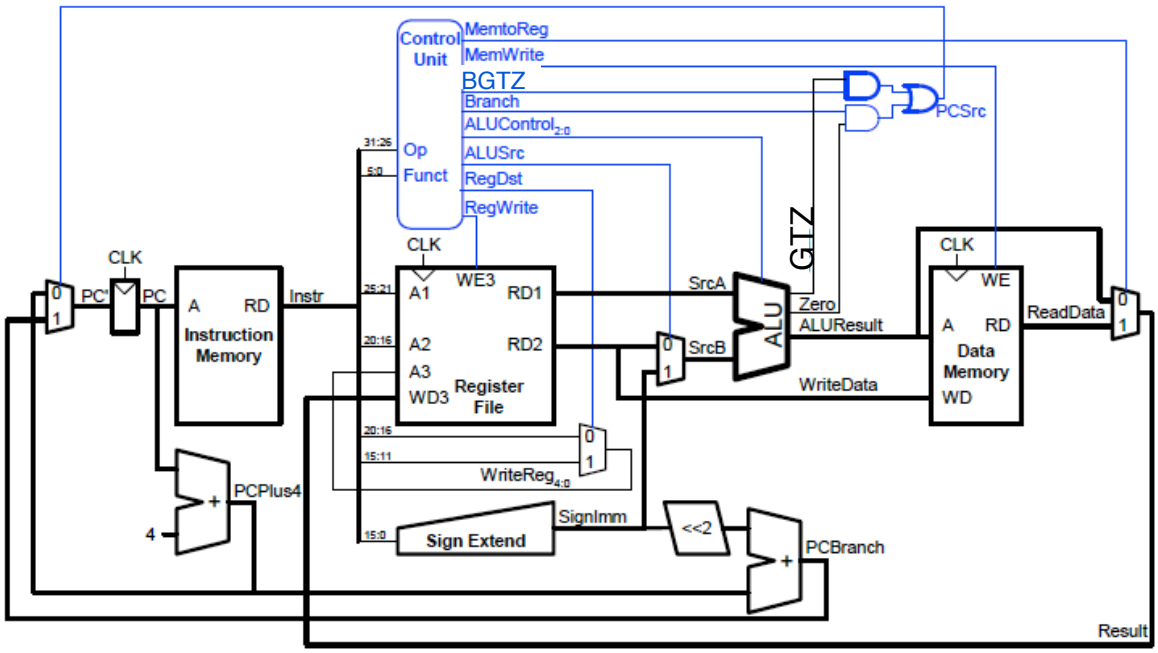
Therefore, the only difference between slti and slitu is that if an overflow occurs, it will be ignored in the case of an unsigned comparison.

(c) bgtz

First, we modify the ALU - we introduce a GTZ flag:



We modify the datapath: PCSrc = (BGTZ & GTZ) | (Branch & Zero)



We add an output to the main decoder: BGTZ which is 1 when the instruction is a bgtz instruction.

Instruction	opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	BGTZ
R-type	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
BGTZ	000111	0	X	0	0	0	X	01	1

TABLE 7.6 Main decoder truth table enhanced to support blez

(d) lb

Add a new main decoder output LB, where LB is 1 if the instruction is an lb instruction.

Take the least significant 8 bits from the Result wire, and sign extend it to a 32 bit value, Result_b. A multiplexer with Result and Result_b as inputs feeds its output into the WD3 from the register file. The “select” input to that multiplexer is LB such that when LB is 1, the output of that multiplexer is Result_b, and when LB is 0, the output of that multiplexer is Result.

Here we assume that reads on the register file are not word-aligned.

Q2)

(a) Given the alternative five-stage pipeline diagram below, discuss the advantages and disadvantages of this design for each of the 4 points discussed above.

1. We still avoid RAW dependencies between arithmetic-logical instructions, but this is at the cost of one extra ALU. On the other hand, the forwarding unit is greatly simplified.
2. The 1 cycle load latency is avoided, but we created a 1 cycle address generation latency penalty in the case where an arithmetic instructions generates a result for a register used in an address computation, as in:
   i: R5 <- R6 + R7
   i+1: R9 <- Mem[R5]
3. The branch resolution that used to be done in the EX stage needs to be done now in the AG stage, but this should not be a problem, for we need an adder (ALU) for the address computation in that stage.
4. Now it is the AG stage that is unused for arithmetic-logical operations.

(b) If the two ALUs in the alternative 5-stage pipeline are used respectively for branch target computation and register comparison, what would be the penalties for branch instructions:

- In the case where stalling occurs as soon as a branch is recognized?

Assuming the branch target computation is done in the AG stage and the branch resolution computation, the penalty for stalling every time a branch is encountered would be three cycles regardless of the branch outcome.

- In the case where a branch-not-taken prediction policy is used?

With the same assumptions, if we mis-predict (branch is taken), we have to nullify the actions of the three instructions following the branch. Since at this point these three instructions following the branch have not yet reached a stage where either memory or registers can be modified, we can again no-op them.

If we correctly predict the outcome of the branch, then there is no penalty as a result.