Jordan Wu

U0900517

Homework Assignment #6

1. **For this problem, the task is to design control logic to control an ALU. Consider the following control inputs to the entire ALU: (You may want to refer to the ALU design in appendix B, pages B-33 and B-34.)**

    - **a_{invert}**
    - **b_{invert}**
    - **c_{in}, carry in**
    - **OP_0, the low bit of the operation input**
    - **OP_1, the high bit of the operation input**

    Assume you only need to design ALU control circuitry for the following six R-type instructions:

    - **add**
    - **and**
    - **nor**
    - **or**
    - **sub**
    - **slt**

Part 1: The MIPS encoding for each of these six instructions has an OP code of 0 and different function codes. Look up the function codes for each of the instructions above. Write the function codes for each instruction using six bits of binary.

Remember that R-type instructions are instructions for registers. The MIPS Reference Data Sheet (Green Reference Card), has the following format for an R-type instructions.

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

All R-type instructions have the same opcode of 000000. The only way to interpret the instruction will be to look at the funct bits which are the last 6 bits of the instruction. Using the MIPS Reference Data Sheet again to find the individual funct code of each of the following instruction.

instruction	$funct_{hex}$	$funct_2$
add	20_{hex}	10-0000
and	24_{hex}	10-0100
nor	27_{hex}	10-0111
or	25_{hex}	10-0101
sub	22_{hex}	10-0010
slt	$2a_{hex}$	10-1010

Part 2: Draw a truth table with exactly six inputs and five outputs. Label the inputs F5 ...F0. Label the outputs with the names of the five ALU control signals (from above).

For the following truth table, the six inputs will be the following F5 F4 F3 F2 F1 F0. This input sequence will represent the funct codes, where the MSB will be F5. The outputs will be a_{invert}, b_{invert}, c_{in}, OP_0, and OP_1. Here is the truth table below.

Instruction name	inputs						outputs				
	F5	F4	F3	F2	F1	F0	a_{invert}	b_{invert}	c_{in}	OP_1	OP_0
add	1	0	0	0	0	0	0	0	0	1	0
and	1	0	0	1	0	0	0	0	0	0	0
nor	1	0	0	1	1	1	1	1	0	0	0
or	1	0	0	1	0	1	0	0	0	0	1
sub	1	0	0	0	1	0	0	1	1	1	0
slt	1	0	1	0	1	0	0	1	1	1	1

Part 3: The next task is to write Boolean equations for each output. Write one Boolean equation for each output.

The Boolean equation for each of the outputs are:

$$a_{invert} = F5 \cdot \overline{F4} \cdot \overline{F3} \cdot F2 \cdot F1 \cdot F0$$

$$\begin{aligned} b_{invert} &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot F2 \cdot F1 \cdot F0 + F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot F1 \cdot \overline{F0} \\ &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot F1 \cdot (F2 \cdot F0 + \overline{F2} \cdot \overline{F0}) \\ &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot F1 \end{aligned}$$

$$C_{in} = F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot F1 \cdot \overline{F0}$$

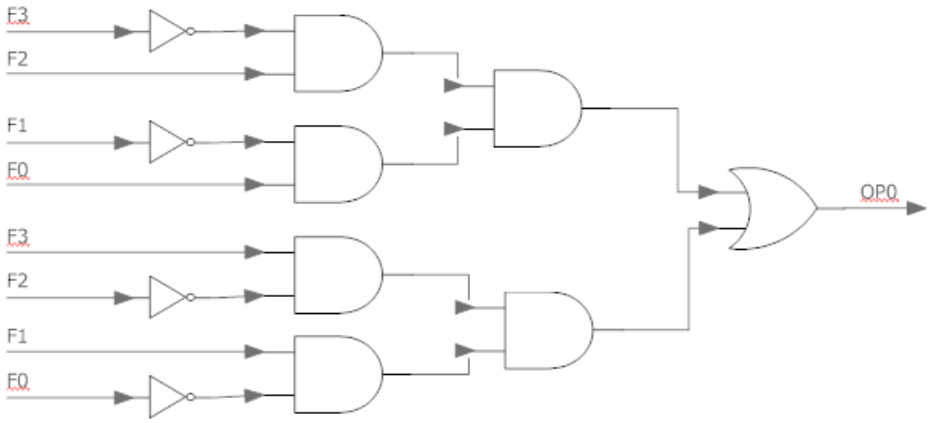
$$\begin{aligned} OP_1 &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot \overline{F1} \cdot \overline{F0} + F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot F1 \cdot \overline{F0} + F5 \cdot \overline{F4} \cdot F3 \cdot \overline{F2} \cdot F1 \cdot \overline{F0} \\ &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot \overline{F0} \cdot (\overline{F1} + F1) + F5 \cdot \overline{F4} \cdot \overline{F2} \cdot F1 \cdot \overline{F0} \cdot (\overline{F3} \cdot F3) \\ &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot \overline{F2} \cdot \overline{F0} + F5 \cdot \overline{F4} \cdot \overline{F2} \cdot F1 \cdot \overline{F0} \\ &= F5 \cdot \overline{F4} \cdot \overline{F2} \cdot \overline{F0} \cdot (\overline{F3} + F1) \end{aligned}$$

$$\begin{aligned} OP_0 &= F5 \cdot \overline{F4} \cdot \overline{F3} \cdot F2 \cdot \overline{F1} \cdot F0 + F5 \cdot \overline{F4} \cdot F3 \cdot \overline{F2} \cdot F1 \cdot \overline{F0} \\ &= F5 \cdot \overline{F4} \cdot (\overline{F3} \cdot F2 \cdot \overline{F1} \cdot F0 + F3 \cdot \overline{F2} \cdot F1 \cdot \overline{F0}) \end{aligned}$$

Part 4: Draw a logic gate diagram that computes OP_0.

We know that F5 and $\overline{F4}$ will always be satisfied since all the instructions we are considering uses 2_{hex} for its first two MSB bits. This means the equation will be reduced to the following.

$$OP_0 = \overline{F3} \cdot F2 \cdot \overline{F1} \cdot F0 + F3 \cdot \overline{F2} \cdot F1 \cdot \overline{F0}$$



2. The ALU can do addition and subtraction on 32-bit signed and unsigned numbers. Unfortunately, certain numbers can be added or subtracted that 'overflow' a 32-bit representation. This occurs when the result bits cannot accurately represent the actual sum or difference. Overflow includes results that are too large (too positive), or too small (too negative). We can easily detect overflow by looking at numbers being added, the results, and the operation. This problem asks you to define an overflow detecting circuit.

Consider a small part of this ALU; consider the one-bit ALU for the *most significant* bit of a 32-bit ALU. (See the figures on pages B-33 and B-34 of appendix B.) Based on these ALU designs, you should design the overflow detection unit for the 32-bit ALU. Your design should include:

- A discussion of what is required to detect overflow conditions for addition and subtraction of signed and unsigned numbers.
- A single truth table that indicates the overflow output for various inputs to the overflow detection unit
- A single Boolean equation (in sum-of-products form) that describes when the overflow output is true.

Refer to page B-33, and find the overflow detection unit. You must assume the overflow detection unit has the following **six** inputs.

a_{31}

The highest bit of the first 32-bit input addend.

b_{31}

The highest bit of the second 32-bit input addend. Note that for subtraction, this input will be inverted *before* it is passed to your overflow detection unit.

B_{invert}

The input to the ALU (and to your overflow detector) that indicates if the high-level operation is subtraction. (Remember than subtraction is just addition of a negated value, plus one.)

Sum

The highest bit of the sum of the two 32-bit addends (as an output from the ALU after the addition circuit). It should be noted that for subtraction, this bit is still the result of addition, but the second 32-bit *addend* 'b' is modified to be the two's complement negation of the subtrahend.

Carry

The carry out of the 32-bit addition. (See the above note about subtraction.)

Signed

An input (not shown in the diagram) that indicates if the operation is signed or unsigned. This input is true if the addends and result are signed numbers, and false if the addends and result are unsigned numbers. This input is not shown on the diagram, but we'll assume it exists.

Part 1 – Detecting overflow

I will examine each of the four cases separately before making a conclusion statement on how to detect overflow

- Addition of signed numbers

  NO OVERFLOW can occurs when adding operands with different signs. The reason is that when you add two operands with different signs the sum is never larger than one of the operands. Since the sum is never larger than one of the operands it can always be represented by 31-bits. Overflow occurs when adding two positive numbers and the sum is negative or when adding two negative numbers and the sum is positive (this mean we can use the sign bit to check for overflow).

- Subtraction of signed numbers

  A similar restriction for NO OVERFLOW applies to subtraction of signed numbers. When subtracting two operands, NO OVERFLOW can occurs when both the operands have the same signs. Remember that c – a = c + (-a), looking at the right hand side of the equation. We will get the rule for addition of signed numbers, for this case NO OVERFLOW occurs when adding operands with different signs. Overflow occurs when subtracting a negative number from a positive number and getting a negative result or when we subtract a positive number from a negative number and getting a positive result (this mean we can use the sign bit to check for overflow).

Here is a table of overflow conditions for addition and subtraction of signed numbers (Figure 3.2 in the textbook showing 4 cases to detect overflow for signed numbers)

Operation	Operand A	Operand B	Result indicating overflow
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

- Addition of unsigned numbers

  When adding unsigned number, this number can be from 0 to $2^{32} - 1$ (4294967295_{ten}). Which means that adding two operands that will produce a result that is over this boundary will cause overflow. We can check by looking at the CarryOut and the two MSB of both operands.

- Subtraction of unsigned numbers

When subtracting unsigned numbers, two operands that will produce a result that is over this boundary will cause overflow. We can check by looking at the CarryOut and the two MSB of both operands.

Conclusion:

Case 1: (when adding two positive numbers and the result is negative)

For signed numbers, when a_{31} and b_{31} are both 0 meaning that they are both positive numbers and their sum is equal to one. That means that there was a CarryIn from the previous 1 bit ALU, this will change our sign bit to 1 (negative result). This means that when we are adding two positive signed numbers and their MSB are both 0, overflow occurs when their sum is 1.

Case 2: (when adding two negative numbers and the result is positive)

For signed numbers, when a_{31} and b_{31} are both 1 meaning that they are both negative numbers and their sum is equal to 0. That mean that there was no CarryIn from the previous 1 bit ALU, this will change our sign bit to 0 (positive result). This means that when we are adding two negative signed numbers and their MSB are both 1, overflow occurs when their sum is 0.

Case 3: (subtracting a negative number from a positive number and get a negative result)

For signed numbers, when a_{31} is 0 (positive number) and b_{31} is 1 (negative number) and B_{invert} is 1 (we are subtracting these two operands), there is overflow if the sum is 1 (result is negative) and there is no CarryOut.

Case 4: (subtracting a positive number from a negative number and get a positive result)

For signed number, when a_{31} is 1 (negative number) and b_{31} is 0 (positive number) and B_{invert} is 1 (we are subtracting these two operands), there is overflow if the sum is 0.

Case 5: (adding unsigned numbers)

For unsigned number, when a_{31} is 0 and b_{31} is 1, there will be overflow if there is a CarryOut of 1.

Case 6: (subtracting unsigned numbers)

For unsigned number, a_{31} is 1 and b_{31} is 0, there will be overflow if there is a CarryOut of 1 and B_{invert} is 1.

Part 2 – Drawing the truth tables

Case	inputs						outputs
	a_{31}	b_{31}	B_{invert}	Sum	Carry	Signed	Overflow
1	0	0	0	1	Don't Care	1	1
2	1	1	0	0	Don't Care	1	1
3	0	1	1	1	Don't Care	1	1
4	1	0	1	0	Don't Care	1	1
5	0	1	0	Don't Care	1	0	1
6	1	0	1	Don't Care	1	0	1

Part 3 – Writing the Boolean equation

$$Overflow = \overline{a_{31}} \cdot \overline{b_{31}} \cdot \overline{B_{invert}} \cdot Sum \cdot Signed + a_{31} \cdot b_{31} \cdot \overline{B_{invert}} \cdot \overline{Sum} \cdot Signed + \overline{a_{31}} \cdot b_{31} \cdot B_{invert} \cdot Sum \cdot Signed + a_{31} \cdot \overline{b_{31}} \cdot B_{invert} \cdot \overline{Sum} \cdot Signed + \overline{a_{31}} \cdot b_{31} \cdot \overline{B_{invert}} \cdot Carry \cdot \overline{Signed} + a_{31} \cdot \overline{b_{31}} \cdot B_{invert} \cdot Carry \cdot \overline{Signed}$$

http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/overflow.html