Homework 9
Assembly Language & Architecture

1. A. Add Hamming parity bits to the value 204, assuming it is stored in unsigned format using 8 bits.
   \[(110011010100)\]

   B. Describe the data-recovery process and the exact value recovered if bit 3 is somehow flipped. Do the same if bit 9 is flipped, and if both 3 and 9 are flipped simultaneously. \((\text{recovered to } 204; \text{ recovered to } 204; \text{ unrecoverable})\)

2. For each of the following sets of 8 data bits interleaved with 5 parity bits using the Hamming system, is the value valid, or was there an error? If the original unsigned number is recoverable, what is it? If not, why not?
   A. \(1101011101110\) \((\text{error; 255})\)
   B. \(100100011110\) \((\text{valid; 129})\)
   C. \(100001101110\) \((\text{error; unknown})\)

Programming

Your assignment is to use Logisim make a rudimentary Control unit for a MIPS CPU that parses a 32-bit instruction in machine language.

It will have a single 32-bit input, which will be the instruction to be parsed. Make this input on the west side of the control circuit you design. You need to implement the following instructions:

- \(\text{add}\)
- \(\text{sub}\)
- \(\text{addi}\)
- \(\text{ori}\)
- \(\text{lw}\)
- \(\text{sw}\)
- \(\text{beq}\)
- \(\text{j}\)

You will need to make the following outputs:

- **Three mutually-exclusive bits indicating if the instruction is R-, I-, or J-type.** These will be on the north side of the circuit, in this order.
- **The opcode, rs, rt, rd, shamt, funct, immediate, and address fields.** Output all of these for every instruction, regardless of its type. (So you will output \(rs\) for J-type instructions, using bits 21-25, even though it’s really meaningless.) These will be on the south side of the circuit (left to right), in the order I have indicated here.
- **The control outputs RegWrite, RegDst, ALUSrc, Branch, Jump, MemRead, MemWrite, MemtoReg, and ALUInput.** All of these are single bits except ALUInput, which is 4 bits long. \(\text{RegWrite}\) is true when the instruction writes to a register. \(\text{RegDst}\) is true when the register being written to is determined by bits 11-15 of the instruction, and false when it’s determined by bits 16-20. \(\text{ALUSrc}\) is true when the second operand sent to the ALU comes from an immediate value. \(\text{Branch}\) is true when the instruction is a branch instruction (but not a jump). \(\text{Jump}\) is true when the instruction is a \(j\) or \(jal\). \(\text{MemRead}\) is true when main memory must be read. \(\text{MemWrite}\) is true when main memory must be written. \(\text{MemtoReg}\) is true when the results of a memory read need to be directed back to a register. \(\text{ALUInput}\) defines the operation to be given to the ALU’s control; see the table below for values and their meanings. Note that the output in some cases won’t matter (e.g. when \(\text{RegWrite}\) is false, \(\text{RegDst}\)‘s value doesn’t matter.) These outputs will be on the east side of the circuit, in this order (top to bottom).

It may be easier to separate these three types of outputs into different subcircuits, to keep the design modular. Use the template that is provided, and call your file control.circ.

<table>
<thead>
<tr>
<th>ALU Control</th>
<th>Operation</th>
<th>ALU Control</th>
<th>Operation</th>
<th>ALU Control</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>and</td>
<td>0010</td>
<td>addition</td>
<td>0111</td>
<td>set-on-less-than</td>
</tr>
<tr>
<td>0001</td>
<td>or</td>
<td>0110</td>
<td>subtraction</td>
<td>1100</td>
<td>nor</td>
</tr>
</tbody>
</table>