CDA 3101 Assignment 7
Due in class on Friday, Nov. 19

Turn hardcopy in class, stapled, with your name and “CDA 3101 Assignment 7” clearly written on it.

Assignments submitted after Nov. 19, 12:35 pm but before Nov. 22, 12:35 pm will be considered one day late.

IMPORTANT: Use a ruler and a RED pen to make any changes/additions to any of the figures provided below, so that the changes are clearly visible. In case, you make a mistake, I suggest you start with a clean figure. Also, clearly write (in a numbered list) what changes/additions you have made to the figures and why (Please feel free to use extra pages for this).

1. (12 points) Describe the effect of stuck-at-0 fault (i.e. regardless of what it should be, the signal is always 0) would have for signals shown below, in the single-cycle datapath provided as part of Q2. Which instruction, if any, will not work correctly? Explain why.

Consider each of the following faults separately:

   a. RegWrite = 0
   b. ALUop0 = 0
   c. ALUop1 = 0
   d. Branch = 0
   e. MemRead = 0
   f. MemWrite = 0

2. (16 points) Imagine I want to support the \texttt{bge $t0, $t1, label}, which compares $t0 and $t1 and branches only if $t0 \geq $t1 (without breaking the rest of the instructions like add, sub, and, or, beq, lw, sw etc.). The instruction uses I-format representation (6 bits opcode, 5 bit rs, 5 bit rt, 16 bits label). Assume that the ALU generates another output ‘sign’ which is 1 when the ALU output is negative. (Hint: $t0 \geq $t1 if $t0 - $t1 \geq 0).
a. Make changes to the single-cycle datapath and control shown below to support this instruction. You should only add wires, gates, muxes to the datapath; do not modify the main functional units (the memory, register file, and ALU) themselves. Try to keep your diagram neat!

b. Complete the table given below by writing the values (0/1/X) next to each control signal to show the values that would exist in the circuit after all the “work” for executing this instruction has completed, right before the moment that the rising edge of the next clock cycle occurs. X stands for don’t care.

If you need add a new control signal, please add it along with its value to the table below.

For each value that you either state how many bits the value takes up or make it implicit...don’t write 2 for a 2-bit signal ...you should instead write either “2 (2-bits)” or “10”, making it obvious that it is a 2-bit signal.

<table>
<thead>
<tr>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegDst</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALU Op</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. (10 points) Imagine I want to implement the "jal label" instruction (without breaking the rest of the instructions like add, sub, and, or, beq, lw, sw). This instruction stores the address of the next instruction (PC+4) in register $ra (i.e. $31) and then jumps to the target address given by the label. This instruction uses the J-Format representation (6 bit opcode, 26 bit address info). The target address is computation is same as in the jump instruction. All assumptions about the operation of the functional units are as discussed in class.

a. Make changes to the single-cycle datapath and control shown below to support this instruction.

b. Complete the table given below by writing the values next to each control signal to show the values that would exist in the circuit after all the “work” for executing this instruction has completed, right before the moment that the rising edge of the next clock cycle occurs.

Add any new control signal(s) with their values to the table below that may be required to implement the jal instruction.

For each value that you either state how many bits the value takes up or make it implicit...don’t write 2 for a 2-bit signal ...you should instead write either “2 (2-bits)” or “10”, making it obvious that it is a 2-bit signal.

For control signals that are don’t care, specify X.

<table>
<thead>
<tr>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegDst</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALU Op</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Datapath Diagram](image-url)
4. (12 points) Using the single-cycle datapath figure given above, simulate \texttt{addi \$5, \$2, 0xBA00} (which occurs at address \texttt{0x00400040}) by indicating the numeric value (in Hex or Binary) of each of the data and control signals given below. You should show the values that would exist in the circuit after all the “work” of the instruction has completed, right before the moment that the rising edge of the next clock cycle occurs.

Make sure for each value, you (i) indicate specifically whether the value is in Hex or Binary and (ii) either state how many bits the value takes up or make it implicit...thus don’t write $7_{16}$ for a 32-bit number, you should instead write either “$7_{16}$ (32-bits)” or “00000007_{16}”, making it obvious that it is a 32-bit number.

You should write “X” if the control signal doesn’t matter and you must give justification why the control signals should be set to a the value you have indicated.

The \texttt{addi} instruction uses the I-format representation. Assume the register contents shown below (in hexadecimal).

\begin{verbatim}
R00: 00000000  R08: 00021974  R10: 00000005  R18: 00007000
R01: 0000047c  R09: 00000fff  R11: 00041500  R19: 00000044
R02: 00010858  R0A: 0a0b0c0d  R12: 00085200  R1A: 00055555
R03: 00010824  R0B: 000000aa  R13: 00000000  R1B: 00000347
R04: 00022000  R0C: 00085200  R14: 00067100  R1D: ef7b1808
R05: 00010884  R0D: 0072a45c  R15: 47000000  R1E: effff920
R06: 00000000  R0E: effff880  R16: 0c483000  R1F: 0001085c
R07: 00055000  R0F: 00042828  R17: 00041500
PC = Input to the PC register =
INS[31..26] = Input 1 to ALU =
Write Register = “Write Data” input to register file =
“Address” input of data memory = “Write Data” input of data memory =
\end{verbatim}

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>Value (0/1/X)</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegWrite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemtoReg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUSrc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegDst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUOp 2 bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALU Ctrl unit output (4 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>