Unpipelined Beta

Control logic

<table>
<thead>
<tr>
<th>ALUFN[5:0]</th>
<th>RESET</th>
<th>IRQ</th>
<th>OP</th>
<th>OPC</th>
<th>LD</th>
<th>LDR</th>
<th>ST</th>
<th>JMP</th>
<th>BEQ</th>
<th>BNE</th>
<th>ILLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>000011</td>
<td>--</td>
<td>--</td>
<td>F(op)</td>
<td>(op)</td>
<td>&quot;+&quot;</td>
<td>&quot;A&quot;</td>
<td>&quot;+&quot;</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>000101</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>000111</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>--</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>010000</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010001</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101000</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101110</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100110</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101011</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101001</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Problem 1.

For this problem assume that each register has been initialized to the value $0x0000??00$ where "??" is the register number as a two-digit hex number. So R0 is initialized to $0x00000000$, R1 to $0x00000100$, ..., and R30 to $0x00001E00$. R31 of course always reads as 0.

For each instruction below, please indicate the values that will be found in the unpipelined Beta datapath just before the end of the clock cycle in which the instruction is executed. If the value doesn't matter since it's not used during the execution of the instruction or can't be determined, write "-".

. = $0x100$

SUB(R5,R3,R7)
. = 0x100
LD(R3, -0x200, R7)

// hex for instruction
0x60E3FE00
\[ . = 0x100 \]

\[ \text{JMP(LP)} \]

\[ . = 0x100 \]

\[ \text{BEQ(R31, .+0x80, LP)} \]

\[ \text{Literal: } (0x180 - 0x100) \]

\[ = 0x1f \]

\[ 4 \times \text{Literal} = 0x7c \]
Problem 2.

Consider adding the following instructions to the Beta instruction set, for implementation on the Beta hardware shown in lecture (see diagram included in the reference material at the end of this quiz). You’re allowed to change how the control signals are generated but no modifications to the datapath are permitted.

For each instruction either fill in the appropriate values for the control signals in the table below or put a line through the whole row if the instruction cannot be implemented using the existing Beta datapath. Use “—” to indicate a “don’t care” value for a control signal. The values can be a function of Z (which is 1 when Reg[Ra] is zero).

LDX(Ra, Rb, Rc) // Load indexed
   EA ← Reg[Ra] + Reg[Rb]
   Reg[Rc] ← Mem[EA]
   PC ← PC + 4

STX(Ra, Rb, Rc) // Store indexed
   EA ← Reg[Ra] + Reg[Rb]
   Mem[EA] ← Reg[Rc]
   PC ← PC + 4

MVZC(Ra, literal, Rc) // Move constant if zero
   If Reg[Ra] == 0 then Reg[Rc] ← SXT(literal)
   PC ← PC + 4

SOB(Ra, literal, Rc) // Subtract one and branch
   PC ← PC + 4
   EA ← PC + 4*SEXT(literal)
   tmp ← Reg[Ra]
   Reg[Rc] ← Reg[Ra] - 1
   if tmp != 0 then PC ← EA

ARA(Ra, literal, Rc) // Add Relative Address
   Reg[Rc] ← Reg[Rc] + PC + 4 + 4*SEXT(literal)
   PC ← PC + 4

(FILL IN TABLE BELOW)

<table>
<thead>
<tr>
<th>Instr</th>
<th>ALUFN</th>
<th>WERF</th>
<th>BSEL</th>
<th>WDSEL</th>
<th>MOE</th>
<th>MWR</th>
<th>RA2SEL</th>
<th>PCSEL</th>
<th>ASEL</th>
<th>WASEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX</td>
<td>&quot;4&quot;</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVZC</td>
<td>&quot;B&quot;</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>o</td>
<td>?</td>
<td>o</td>
<td>?</td>
<td>o</td>
</tr>
<tr>
<td>SOB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARA</td>
<td>&quot;4&quot;</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>?</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

6.004 Worksheet - 5 of 8 - L10 – Beta Implementation
Problem 3.

Ben Bitdiddle is proposing the short assembly language program shown to the right as a manufacturing test to ensure the correct operation of the Control ROM. He is assuming -- and you may too -- that the Beta datapath components (e.g., Memories, ALU, muxes, register file, adders) are working correctly and that any errors in execution are due to faulty signals from the Control ROM. Ben's plan is to run the program then look at the value stored in the memory location labeled ANS. If the value is 0x6004, the test passes, otherwise the Beta being tested is declared faulty and discarded.

For each of the following faults, indicate the value that the faulty Beta will store into ANS.

(A) RA2SEL is stuck at the value 0.

Value stored in ANS by faulty Beta: 0x6003

(B) WDSEL[1:0] is stuck at the binary value 00.

Value stored in ANS by faulty Beta: 0x8

(C) PSEL[2:0] is stuck at the binary value 000.

Value stored in ANS by faulty Beta: 0x6005

Problem 4. Beta Implementation

Consider the assembly language program shown to the right. Assume that all register values are initialized to 0, execution starts at PC=0 and halts when HALT() is executed.

This program is run on 4 different broken Betas, where each Beta has a specified control signal stuck at the specified value, i.e., the control signal value is fixed and is not affected by the value produced by the Beta's CTL module. For each broken Beta, please give the value in registers R1, R2, R3, and the location X: after the programs halts. Assume that any don't care control signal values are 0.

<table>
<thead>
<tr>
<th>Broken control signal</th>
<th>Final value in</th>
<th>Location X:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA2SEL stuck at 0</td>
<td>42 1 0xc</td>
<td>0x contents of R0</td>
</tr>
<tr>
<td>WDSEL stuck at 0b00</td>
<td>0x10 8 0xc</td>
<td>0x10</td>
</tr>
<tr>
<td>WASEL stuck at 1</td>
<td>0 0 0</td>
<td>-42</td>
</tr>
<tr>
<td>WERF stuck at 1</td>
<td>0x14 1 0xc</td>
<td>42</td>
</tr>
</tbody>
</table>

SELECT RB, note RC=PC+4 only write to R3 CHANGE RC during ST
Problem 5.

In this problem, you will consider a number of plausible hardware faults in an otherwise working Beta processor; you may want to consult the diagram and documentation on the backs of pages of this quiz. Each of the faults involves changing a particular output of the control logic to some new (incorrect) constant value. In each case, you are to evaluate the impact of the fault on each of the following Beta instructions:

\[
\begin{align*}
I_1: & \quad ST(R0, 0x100, R1) \\
I_2: & \quad JMP(LP, R31) \\
I_3: & \quad BEQ(R31, .+4, R0) \\
I_4: & \quad SUB(R1, R0, R0)
\end{align*}
\]

For each of the following faults, identify which (if any) of the above instructions will fail to work properly - that is, if the fault might effect the processor state (register and FC values) after the execution of the instruction. Be careful: some of these are tricky!

(A) ALUUFN stuck at code for "." (32-bit SUBTRACT)

Which instruction(s) fail? Circle all applicable, or NONE: \( I_1 \) \( I_2 \) \( I_3 \) \( I_4 \) NO

(B) RA2SEL stuck at 1

\textit{Note for I4: R5 = Rc, so RA2SEL doesn't matter}

Which instruction(s) fail? Circle all applicable, or NONE: \( I_1 \) \( I_2 \) \( I_3 \) \( I_4 \) NO

(C) WERF stuck at 0

Which instruction(s) fail? Circle all applicable, or NONE: \( I_1 \) \( I_2 \) \( I_3 \) \( I_4 \) NO

(D) BSEL stuck at 0

Which instruction(s) fail? Circle all applicable, or NONE: \( I_1 \) \( I_2 \) \( I_3 \) \( I_4 \) NO

Problem 6.

(A) The Beta executes the assembly program below starting at location 0 and stopping when it reaches the HALT() instruction. Please give the values in the indicated registers after the Beta stops. Write the values in hex or write "CAN'T TELL" if the values cannot be determined.

\[
\begin{align*}
\text{Value left in R0 or "CAN'T TELL": } & \quad 0x87654321 \\
\text{Value left in R1 or "CAN'T TELL": } & \quad 0x1 \\
\text{Value left in R2 or "CAN'T TELL": } & \quad 0xC
\end{align*}
\]

\[\begin{array}{ll}
0 & LD(r31, x, r0) \\
2 & CMPLE(r0, r31, r1) \\
3 & BNE(r1, L1, r2) \\
4 & ADDC(r31, 1, r0) \\
6 & L1: \quad HALT() \\
8 & x: \quad LONG(0x87654321)
\end{array}\]
(B) Redo part (A) but this time assume that all the control signals going to the datapath from the control logic are stuck at logic 0, except for \texttt{WERF} which operates as expected. Note that when \texttt{ALUFN}[4:0] = 0b00000, the \texttt{ALU} computes \texttt{A+B}.

\begin{verbatim}
PCSELc = 0 \Rightarrow \text{Branch not taken} \\
\texttt{WSEL} = 0 \Rightarrow \texttt{PC}+4 \text{ always written into \texttt{Rc}}
\end{verbatim}

\begin{itemize}
\item Value left in \texttt{R0} or "CAN'T TELL": 0x\hspace{1cm}10
\item Value left in \texttt{R1} or "CAN'T TELL": 0x\hspace{1cm}8
\item Value left in \texttt{R2} or "CAN'T TELL": 0x\hspace{1cm}C
\end{itemize}

(C) Bettah Beta Inc. (you can tell they’re based in Boston!) is proposing a new Beta instruction \texttt{TCLR} that sets \texttt{Rc} to the current value of a memory location whose address is in \texttt{Ra} and writes a zero to that location, all in a single cycle. They are assuming that main memory works as it does in JSim: its read ports are combinational and the write port takes a \texttt{CLK} signal and performs the write at the end of the current cycle – so the same memory location can be read and written in the same clock cycle.

Here’s their draft entry for the Beta reference manual:

\begin{verbatim}
Usage: TCLR(Ra,Rc)
Opcode: 011010 \\
Operation: PC \leftarrow PC + 4 \\
\quad EA \leftarrow \text{Reg}[Ra] \\
\quad \text{Reg}[Rc] \leftarrow \text{Mem}[EA] \\
\quad \text{Mem}[EA] \leftarrow 0
\end{verbatim}

The contents of register \texttt{Rc} are set to the contents of the memory location whose address is in \texttt{Ra}. Then, at the end of the cycle, that memory location is set to 0.

Please fill in the appropriate values for the control signals that will cause the datapath to implement the correct operations OR briefly explain why \texttt{TCLR} cannot be implemented with the existing Beta datapath in a single cycle.

\begin{center}
\textbf{Fill in table:}
\end{center}

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Instr} & \textbf{ALUFN} & \textbf{WERF} & \textbf{DSEL} & \textbf{WDSEL} & \textbf{AWR} & \textbf{RA2SEL} & \textbf{PCSEL} & \textbf{ASEL} & \textbf{WASEL} \\
\hline
TCLR & "A" & 1 & - & 2 & 1 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}

- \texttt{ASEL} = 0, \texttt{ALUFN} = "A" \Rightarrow \text{memory address in Reg}[Ra] \\
- \texttt{WDSEL} = 2, \texttt{WERF} = 1, \texttt{WASEL} = 0 \Rightarrow \text{register written with Mem}[Reg[\texttt{Ra}]] \\
- \texttt{RA2SEL} = 0, \texttt{WR} = 1 \Rightarrow \text{memory write data is Reg}[R6] = \text{Reg}[3] = 0 \\
and main memory will write

6.004 Worksheet - 8 of 8 - L10 – Beta Implementation