1. [Based on question #4.1 of Summer 95 Final] **Pipelined Ripple_Carry Adder:**

Given below is an arrangement where a 4-bit register file is to be used with the pipelined ripple carry adder discussed in your class notes. The register file has 8 registers and also two read ports and one write port. We need to be able to perform only two instructions using this setup, an ADD and a NOP (NO-Operation). In the ADD, you add two source registers and store the result in the destination register. In the NOP, it does not matter whether you add or not, you should NOT STORE any result. Here we are NOT designing any HDU (Hazard Detection Unit) or FU (Forwarding Unit) to deal with data dependencies. Let us assume that the compiler is responsible for inserting NOPs to take care of any dependencies.

The instructions are 10-bits long and the formats are given below.

The single-bit opcode is a "1" for **ADD** and a "0" for **NOP**.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>rs = Source Reg 1</th>
<th>rt = Source Reg 2</th>
<th>rd = Destination Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>size of the fields =&gt;</td>
<td>1 bit</td>
<td>3 bits</td>
<td>3 bits</td>
<td>3 bits</td>
</tr>
<tr>
<td><strong>add</strong> rd, rs, rt</td>
<td>1</td>
<td>rs2</td>
<td>rs1</td>
<td>rs0</td>
</tr>
<tr>
<td><strong>nop</strong></td>
<td>0</td>
<td>rs2</td>
<td>rs1</td>
<td>rs0</td>
</tr>
</tbody>
</table>

Instructions keep coming into the IF/ID register on every clock. You are not responsible for instruction fetching.

Complete the datapath and control on the next page. Mark the sizes of all the stage registers. Control bits can be carried along with data in the stage registers. Here we are ignoring the final carry C4 and storing the 4-bit result. **Do NOT be misled by Miss Bruin’s design below!**
List what major design errors you corrected in Miss Bruin’s design.

________________________________________
_________________________________________________________________________________
_________________________________________________________________________________
_________________________________________________________________________________
_________________________________________________________________________________
_________________________________________________________________________________

**IF**

- **opcode**: D D D D D D D D D D
- **rs**: Source Reg 1
- **rt**: Source Reg 2
- **rd**: Destination Reg

**Size = 10bit**

```
IF / ID
R1A2 R1A1 R1A0 R2A2 R2A1 R2A0 WA2 WA1 WA0
WD3 WD2 WD1 WD0
R1D3 R1D2 R1D1 R1D0 R2D3 R2D2 R2D1 R2D0
WRITE
CLKSYS_CLK
REGISTER FILE
```

**ID**

- **WRITE**: A B
- **CLK**: Co Ci S

**EX1**

- **Size =**
  - A B Co Ci S

**EX2**

- **Size =**
  - A B Co Ci S

**EX3**

- **Size =**
  - A B Co Ci S

**EX4/WB**

- **Size =**
  - A B Co Ci S
2. [Based on question 5 of Summer 2003 Midterm and question 8 of Spring 1994 Final]

**Pipeline Design (Stalling / Flushing / Forwarding):**

2.1 **Bubbles are produced**
   (in stalling only/in flushing only/in stalling as well as in flushing/in neither stalling nor flushing).

2.2 **In the early-branch design of the pipeline CPU (current lab6 based on 3rd ed.), flushing and stalling** ________ (never occur in the same clock cycle/may sometimes occur in the same clock cycle/always occur in the same clock cycle).

In a late-branch design (based on the first edition), if the branch below is successful, do flushing and stalling both occur **together** or one would **prevent** the other? Explain.

```
beq $1, $2, TARGET
lw $4, 40 ($5)
or $8, $4, $6
```

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

2.3 **There are 9 (1+4+2+2) control signals generated by the control unit. Eight of these (8 out of 9) are going from the ID stage to the EX stage. Do you need to convert all the 8 signals to zero when you stall an instruction in the ID stage? Please explain below.**

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

2.4 **To ________ (stall/flush) an instruction in ID stage, you inhibit (prevent) updating of the following register(s). (circle as many of the following as you wish)**

PC , IF/ID , ID/EX , EX/MEM , MEM/WB

You never inhibit (prevent) updating of a stage register if you are currently ______________

(flushing / stalling / can not fill this blank with either of the previous two choices).

2.5 **In this question we consider the late-branch design of the first edition with one HDU in ID stage and one FU in EX stage, and an internally forwarding register file.**
In the answers below, if there is a stalling, state the reason for stalling and which instruction(s) in which stage(s) are being stalled. If there is a forwarding, state the reason and also state which instruction from which stage is offering forwarding help to which instruction in which stage.

All the three streams use the same 3 instructions in different order.

Stream #1

\[
\text{add } \$3, \$3, \$1; \quad \text{lw } \$3, 40(\$5); \quad \text{or } \$6, \$5, \$4;
\]

Stream #2

\[
\text{lw } \$3, 40(\$5); \quad \text{or } \$6, \$5, \$4; \quad \text{add } \$3, \$3, \$1;
\]

Stream #3

\[
\text{lw } \$3, 40(\$5); \quad \text{add } \$3, \$3, \$1; \quad \text{or } \$6, \$5, \$4;
\]

For stream #1 above, the following occur(s): (circle all correct choices)

(i) hazard detection and stalling by HDU
(ii) forwarding by FU
(iii) internal forwarding in the reg. file
(iv) none of these

Remark:

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

For stream #2 above, the following occur(s): (circle all correct choices)

(i) hazard detection and stalling by HDU
(ii) forwarding by FU
(iii) internal forwarding in the reg. file
(iv) none of these

Remark:

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

For stream #3 above, the following occur(s): (circle all correct choices)

(i) hazard detection and stalling by HDU
(ii) forwarding by FU
(iii) internal forwarding in the reg. file
(iv) none of these

Remark:

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

2.5.1 Now reconsider the above three streams in the context of the early-branch design based on the current lab 6. Explain any differences or striking resemblances to your three answers above.

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

EE457 Lab #6 / part 4  4 / 24
© Copyright 2006 Gandhi Puvvada
2.6 In this question we consider the *early-branch* design of our current lab 6 with two HDUs (HDU and HDU_Br) and two FUs (FU and FU_Br). Of course the register file is an *internally forwarding* register file.

Identify the dependencies in the following instruction streams and how they should be resolved:

For the stream #1 above, the following occur(s): (circle all correct choices)

(i) HDU_Br initiated stalling (ii) HDU initiated stalling
(iii) forwarding by FU_Br  (iv) forwarding by FU
(v) internal forwarding in the reg. file  (vi) none of these

Remark:_____________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

For the stream #2 above, the following occur(s): (circle all correct choices)

(i) HDU_Br initiated stalling (ii) HDU initiated stalling
(iii) forwarding by FU_Br  (iv) forwarding by FU
(v) internal forwarding in the reg. file  (vi) none of these

Remark:_____________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

For the stream #3 above, the following occur(s): (circle all correct choices)

(i) HDU_Br initiated stalling (ii) HDU initiated stalling
(iii) forwarding by FU_Br  (iv) forwarding by FU
(v) internal forwarding in the reg. file  (vi) none of these

Remark:_____________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

For the stream #4 above, the following occur(s): (circle all correct choices)

(i) HDU_Br initiated stalling (ii) HDU initiated stalling
(iii) forwarding by FU_Br  (iv) forwarding by FU
(v) internal forwarding in the reg. file  (vi) none of these

Remark:_____________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
Summary: In the lab #6 design for the early-branch, we stall the branch instruction for _________ (0/1/2/3/arbitrary) clock cycles if it is dependent on an **R-type** instruction __________________ in EX stage / in MEM stage. We stall the branch instruction for _________ (0/1/2/3/arbitrary) clock cycles if it is dependent on an **LW** instruction in EX stage (i.e. *beq* is dependent on *lw* immediately ahead of it). We stall the branch instruction for _________ (0/1/2/3/arbitrary) clock cycles if it is dependent on an **LW** instruction in MEM stage.

The result of an **R-type** instruction is available at the end of _______ (EX/MEM/WB) stage, and the result of an **LW** instruction is available at the end of _______ (EX/MEM/WB) stage. However, we choose not to forward these results (to *beq*) from the same stage where they are generated because _____________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

2.7 Whenever a load word (lw) instruction is followed by a dependent instruction (dependent on the word being loaded), the HDU detects the hazard and inserts a bubble. This being the case, to reduce the hardware, the compiler (a simple-minded design of a compiler) can be asked to put a NOP (no operation instruction) between such instructions *without losing any additional performance*. **TRUE / FALSE**

In the case of an early-branch design, can we use the same principle in the case of control hazards with conditional branch instructions by asking compiler to put one NOP after every conditional branch instruction to avoid the hardware associated with flushing the instruction in IF stage? Tell us first if this suggestion is feasible (meaning, it will produce correct output for the program)? If it is feasible, do you change (lose or gain) performance by doing so? Compare with the above case of lw.
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

2.8 In this question we focus on the **specific point of tapping of the branch control signal** in the ID stage for (a) ANDing with the equality inference and (b) for HDU_Br to produce STALL_BEQ. Reproduced below is the relevant extract of the block diagram. In particular, note that both the AND gate in ID-stage and the HDU_Br take the branch control signal from the output of control unit (Point B in the figure).
Mr. Bruin claims that he discovered a problem in this design. He argues that the branch control signal for the AND gate should be taken after the flush mux (Point C) in the design to avoid erroneous branching. For example consider the following stream:

\[
\text{lw} \quad $4, \quad $3(40) ; \\
\text{beq} \quad $4, \quad $0, \quad \text{loop1} ;
\]

The BEQ instruction should be stalled for 2 clock cycles to resolve its dependency on the LW. However, if register $4 contains 0 before the execution of LW, the AND gate sees a 1 on both of its inputs and would take the branch based on wrong value of $4!! So Mr. Bruin concludes that a false branch will occur. Comment on Mr. Bruin’s discovery.

He further offers a solution by moving the tapping of branch control signal from point B to point C instead. Evaluate the proposed solution by answering the following:

It is _______________________________ (a must / a feasible change but does not make any difference / a feasible change that improves the design / a sin) to move the tapping of branch control signal for the AND gate from point B to point C. Explain:

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

It is _______________________________ (a must / a feasible change but does not make any difference / a feasible change that improves the design / a sin) to move the tapping of branch control signal for the HDU_Br from point B to point C. Explain:

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

Another person suggests that instead of waiting for the control unit to generate the branch control signal, the OPCODE field can be re-coded so that we can identify BEQ instruction by inspection of a single bit among the six-bit OPCODE field. With this modification, we can bypass the control unit and get branch control signal from point A in the figure. Is this a good suggestion or bad one? Are there any other things we should take care of? Consider the following control sequence. Notice that in case the first BEQ is taken, the second BEQ should be flushed.

```plaintext
beq    $0 ,  $1 , loop1 ;
beq    $4 ,  $2 , loop2 ;
```

____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________

2.9 Take a closer look at the muxes used to provide forwarding help in the EX stage, reproduced below on the left hand side:

---

**Original lab design**

- Original read data
- Forwarded help from WB stage
- Forwarded help from MEM stage

**Modified design**

- Original read data
- Forwarded help from MEM stage
- Forwarded help from WB stage

---

We observe that the two muxes on the left are arranged in the particular order so that the forwarding help with higher priority (help from MEM stage) is fed into the second mux. Is this ordering significant? If the order of the muxes is reversed (as given in the "Modified design" on the right-hand side), can it be made to work? If so, what aspects/precautions need to be taken into consideration in the design of the FU (forwarding unit)? Answer the following questions:
In the following instruction sequences, we need the forwarded value for $3 ($rs). What should the 2 control signals be?

\begin{align*}
\text{add} & \quad \$10, \quad \$11, \quad \$12; \\
\text{add} & \quad \$3, \quad \$3, \quad \$3; \\
\text{or} & \quad \$6, \quad \$3, \quad \$4;
\end{align*}

In the original design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

In the modified design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

\begin{align*}
\text{add} & \quad \$3, \quad \$3, \quad \$3; \\
\text{add} & \quad \$10, \quad \$11, \quad \$12; \\
\text{or} & \quad \$6, \quad \$3, \quad \$4;
\end{align*}

In the original design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

In the modified design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

\begin{align*}
\text{add} & \quad \$3, \quad \$3, \quad \$3; \\
\text{add} & \quad \$3, \quad \$3, \quad \$3; \\
\text{or} & \quad \$6, \quad \$3, \quad \$4;
\end{align*}

In the original design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

In the modified design, \(\text{FW\_RS\_WB}=_____{0/1/X}\), \(\text{FW\_RS\_MEM}=_____{0/1/X}\)

From the observations made in above instruction sequences, can we generate the 2 forwarding control signals independent of each other (a) in the original design and (b) in the modified design?

\begin{align*}
\text{2.10} & \quad \text{[Based on Question \#6 of Fall 2006 midterm]} \\
\text{FU\_Br and FU in a 5-stage early branch design:}
\end{align*}

Your friend says that the MEM hazard cases shown in the above two streams are attended to by the FU\_Br in ID stage. \textit{Agree / Disagree}.
He further argues that one set of forwarding muxes in EX stage attending to the same very hazard redundantly (MEM hazard between a dependent instruction in EX stage and donor instruction in WB stage) can be removed. **Agree / Disagree.** Explain with a suitable example:

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>M</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[Based on Question #4 of Fall 1995 Final]

**Modified Pipeline Design (7-stage pipeline):**

Pipelined CPU: A variation of the 5-stage pipeline CPU is the following 7-stage pipeline CPU. Here we assume that the memory accesses take two clocks - one for TLB access and the second for cache access. Hence we have IF1 and IF2 in the place of IF stage and similarly MEM1 and MEM2 in the place of MEM stage. Many details are omitted in the simple block diagrams given below.

As before, we always try to resolve dependency problems through forwarding to the extent possible and will resort to stalling if forwarding cannot help.

**Late Branch**

![Late Branch Diagram]

7-stage pipelined version of the late-branch design of the 1st edition

**Early Branch**

![Early Branch Diagram]

7-stage pipelined version of the early-branch design of the 3rd ed. and our lab 6

Remove mux Pair 2? See Q 3.1.
(Treat it as removed for Q3.2)
3.1 The two pairs of forwarding muxes in ID stage (in the early branch design) provide forwarding help from R-type instructions in MEM1 and MEM2 to beq (and also other instructions) in ID stage. Let us investigate whether we really need 3 pairs of forwarding muxes in the EX stage. These muxes (#1, #2, and #3) provide forwarding help to a dependent instruction in the EX from (a) an R-type or lw instruction in WB stage, (b) an R-type instruction in MEM2 stage, and (c) an R-type instruction in MEM1 stage respectively (in that specific order to implement the needed priority). Mr. Trojan argues that the mux pair #2 can be removed but not the mux pair #1. Explain.

_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________

3.2 Compare the original 5-stage late-branch and early-branch pipelines with these 7-stage versions by answering questions in the tables on the next 7 pages (sorry, it is a long question).

3.3 Flushing of the two instructions in the IF1 and IF2 stages in the case of the 7-stage pipeline:

Note: This part of the design is common to both branch implementations (late or early).

The flushing arrangement shown on the side is extracted from the earlier diagrams. As you can see it is hardly complete. Two of your assistants submitted the following designs to you. You are asked to finalize this design. You can adopt any one of them as is, or take any one of them and modify to your liking.
Dependency of a R-type instruction on a load word instruction, stalling by HDU to resolve the dependency problem:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i     lw $1, 60(S2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1    add $4, $1, $6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Any bubbles? How many? Where are they inserted? Complete the Time-Space diagrams. This example is completed by us.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i     lw $1, 60(S2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1    sub $10, $11, $12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+2    add $4, $1, $6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Any bubbles? How many? Where are they inserted?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many comparators does the HDU (not HDU_Br) have? Where do the destination register addr. inputs to the comparators come from?

<table>
<thead>
<tr>
<th># of comparators</th>
<th>Destination reg. addr. input(s) come(s) from:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delay slots for lw: To avoid the use of HDU, how many delay slots should we declare for lw?

| # of Delay slots | |
|------------------| |
|                  | |

Bubbles = ___1 (0/1/2/3) Bubbles = ___1 (0/1/2/3) Bubbles = ___2 (0/1/2/3) Bubbles = ___2 (0/1/2/3)
Dependency of a R-type instruction on another R-type instruction; Forwarding:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>add $5, $7, $9</td>
<td>sub receives latest $1</td>
<td>sub receives latest $1</td>
<td>sub receives latest $1</td>
</tr>
<tr>
<td>i+1</td>
<td>xor $1, $2, $3</td>
<td>from xor when sub is</td>
<td>from xor first time</td>
<td>from xor when sub is</td>
</tr>
<tr>
<td>i+2</td>
<td>or $10, $11, $12</td>
<td>in ______ stage and xor</td>
<td>when sub is in ______</td>
<td>in ______ stage and xor</td>
</tr>
<tr>
<td>i+3</td>
<td>sub $3, $5, $1</td>
<td>is in ______ stage</td>
<td>is in ______ stage</td>
<td>is in ______ stage</td>
</tr>
</tbody>
</table>

Explain forwarding to instruction (i+3)

sub receives latest $1 from xor when sub is in ______ stage and xor is in ______ stage under the control of ____________ (FU/internal forwarding in register file).

sub receives latest $5 from ______________ due to ______________ (FU/internal forwarding in register file).

It receives the same value again second time when sub is in ______ stage and xor is in ______ stage under the control of ____________ (FU_Br/FU/ internal forwarding in register file).

sub receives latest $5 from add when sub is in ______ stage and add is in ______ stage under the control of ____________ (FU_Br/FU/ internal forwarding in register file). It receives the same value again second time when sub is in ______ stage and add is in ______ stage under the control of ____________ (FU_Br/FU/ internal forwarding in register file).
### FU_Br, FU details:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many comparators does the forwarding unit in ID stage (FU_Br, not FU) have?</td>
<td># of comparators in FU_Br =</td>
<td># of comparators in FU_Br =</td>
<td># of comparators in FU_Br =</td>
<td># of comparators in FU_Br =</td>
</tr>
<tr>
<td>How big are the forwarding muxes (n-bit wide m-to-1 mux)? How many? Where do the data inputs to the muxes come from?</td>
<td>Forwarding mux(es) in the A-leg of equality checker (size and number (which is same for the B-leg)) =</td>
<td>Forwarding mux(es) in the A-leg of equality checker (size and number (which is same for the B-leg)) =</td>
<td>Forwarding mux(es) in the A-leg of equality checker (size and number (which is same for the B-leg)) =</td>
<td>Forwarding mux(es) in the A-leg of equality checker (size and number (which is same for the B-leg)) =</td>
</tr>
<tr>
<td></td>
<td>Data inputs for this/these come from</td>
<td>Data inputs for this/these come from</td>
<td>Data inputs for this/these come from</td>
<td>Data inputs for this/these come from</td>
</tr>
</tbody>
</table>

| How many comparators does the forwarding unit in EX stage (FU, not FU_Br) have? | # of comparators in FU = | # of comparators in FU = | # of comparators in FU = | # of comparators in FU = |
| How big are the forwarding muxes (n-bit wide m-to-1 mux)? How many? Where do the data inputs to the muxes come from? | Forwarding mux(es) in the A-leg of ALU (size and number (which is same for the B-leg)) = | Forwarding mux(es) in the A-leg of ALU (size and number (which is same for the B-leg)) = | Forwarding mux(es) in the A-leg of ALU (size and number (which is same for the B-leg)) = | Forwarding mux(es) in the A-leg of ALU (size and number (which is same for the B-leg)) = |
| | Data inputs for this/these come from | Data inputs for this/these come from | Data inputs for this/these come from | Data inputs for this/these come from |

Note: Mux Pair #2 is removed.
### Priority in FU and FU_Br: Note:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority in FU (FU, not FU_Br): Forwarding to a dependent instruction standing in EX stage. Opt to forward from the nearer than the farther</td>
<td>The FU prefers to allow forwarding help from the __________ (MEM/WB) over _______________ (MEM/WB).</td>
<td>Priority is implemented by placing the forwarding muxes receiving forwarding help from (MEM/WB) upstream of the forwarding muxes receiving forwarding help from ____________ (MEM/WB).</td>
<td>The FU prefers to allow forwarding help from the __________ (MEM1/MEM2/WB) over (MEM1/MEM2/WB) as well as __________ (MEM1/MEM2/WB) Further _______________ (MEM1/MEM2/WB).</td>
<td>Note: Mux Pair #2 is removed. Priority is implemented by placing the forwarding muxes receiving forwarding help from __________ (MEM1/MEM2/WB) upstream of the forwarding muxes receiving forwarding help from __________ (MEM1/MEM2/WB).</td>
</tr>
<tr>
<td>Priority in FU_Br (FU_Br, not FU): Forwarding to a BEQ instruction standing in ID stage. Opt to forward from the nearer than the farther</td>
<td>Not applicable</td>
<td>No priority needs to be implemented in FU_Br. TRUE / FALSE Explain: _______________ _______________ _______________ _______________ _______________.</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Priority in FU_Br (FU_Br, not FU): Forwarding to a BEQ instruction standing in ID stage. Opt to forward from the nearer than the farther
Dependency of a BEQ instruction on a R-type instruction; Stalling through HDU_Br, Forwarding through FU_Br/FU:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>beq $2, $4, Target</td>
<td># of instructions that need to be flushed = ____________________</td>
<td># of instructions that need to be flushed = ____________________</td>
<td># of instructions that need to be flushed = ____________________</td>
</tr>
<tr>
<td>i</td>
<td>add $1, $2, $3</td>
<td># of clock cycles beq needs to be stalled = ____________________</td>
<td>beq receives latest $1 from add when beq is in ______ stage and add is in ______ stage under the control of (FU/internal forwarding in register file).</td>
<td>____________________</td>
</tr>
<tr>
<td>i+1</td>
<td>beq $1, $0, loop</td>
<td># of clock cycles beq needs to be stalled = ____________________</td>
<td>beq receives latest $1 from add when beq is in ______ stage and add is in ______ stage under the control of (FU/Internal forwarding in register file).</td>
<td>____________________</td>
</tr>
<tr>
<td>i</td>
<td>add $1, $2, $3</td>
<td># of clock cycles beq needs to be stalled = ____________________</td>
<td>beq receives latest $1 from add when beq is in ______ stage and add is in ______ stage under the control of (FU/Internal forwarding in register file).</td>
<td>____________________</td>
</tr>
<tr>
<td>i+1</td>
<td>xor $11, $12, $13</td>
<td># of clock cycles beq needs to be stalled = ____________________</td>
<td>beq receives latest $1 from add when beq is in ______ stage and add is in ______ stage under the control of (FU/Internal forwarding in register file).</td>
<td>____________________</td>
</tr>
<tr>
<td>i+2</td>
<td>beq $1, $0, loop</td>
<td># of clock cycles beq needs to be stalled = ____________________</td>
<td>beq receives latest $1 from add when beq is in ______ stage and add is in ______ stage under the control of (FU/Internal forwarding in register file).</td>
<td>____________________</td>
</tr>
</tbody>
</table>
Dependency of a BEQ instruction on a lw instruction; Stalling through HDU\_Br, Forwarding through FU\_Br/FU:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>lw $1, $2(40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>beq $1, $0, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How many clock cycles does the BEQ have to be stalled?</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
</tr>
<tr>
<td>i</td>
<td>lw $1, $2(40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>add $6, $5, $4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>beq $1, $0, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How many clock cycles does the BEQ have to be stalled?</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
</tr>
<tr>
<td>i</td>
<td>lw $1, $2(40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td>add $6, $5, $4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+2</td>
<td>or $16, $15, $14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+3</td>
<td>beq $1, $0, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How many clock cycles does the BEQ have to be stalled?</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
<td># of clock cycles beq needs to be stalled = beq receives latest $1 from lw when beq is in ______ stage and lw is in ______ stage under the control of (FU/Br/FU/internal forwarding in register file).</td>
</tr>
</tbody>
</table>
### Miscellaneous:

<table>
<thead>
<tr>
<th>Design item</th>
<th>In 5-stage late-branch</th>
<th>In 5-stage early-branch</th>
<th>In 7-stage late-branch</th>
<th>In 7-stage early-branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many comparators does the HDU_Br) have?</td>
<td>Not applicable</td>
<td># of comparators in HDU_Br = __________</td>
<td>Not applicable</td>
<td># of comparators in HDU_Br = __________</td>
</tr>
<tr>
<td>Destination register addr.(s) come(s) to HDU_Br from .....</td>
<td></td>
<td>Dest. Reg. addr.(s) come(s) from __________</td>
<td></td>
<td>Dest. Reg. addr.(s) come(s) from __________</td>
</tr>
<tr>
<td></td>
<td></td>
<td>__________</td>
<td></td>
<td>__________</td>
</tr>
<tr>
<td></td>
<td></td>
<td>__________</td>
<td></td>
<td>__________</td>
</tr>
<tr>
<td>Though it is not desirable to “delay” the BEQ execution, how late in the pipeline can you execute the BEQ instr.?</td>
<td>The latest stage for executing BEQ is __________ (EX/MEM/WB).</td>
<td>Not applicable</td>
<td>The latest stage for executing BEQ is __________ (EX/MEM1/MEM2/WB).</td>
<td>Not applicable</td>
</tr>
<tr>
<td>The earliest a BEQ can be executed from is:</td>
<td>Not applicable</td>
<td>The earliest stage for executing BEQ is __________ (IF1/IF2/ID/EX).</td>
<td>Not applicable</td>
<td>The earliest stage for executing BEQ is __________ (IF1/IF2/ID/EX).</td>
</tr>
</tbody>
</table>
4  [Based on Question #5 of Summer 2004 Midterm]

Modified Pipeline Design (4-stage pipeline):

4.1  Pipelined CPU design:

Refer to your lab #6 5-stage pipeline design.

For the sake of this problem let us assume that we have a very fast ALU and a very fast Data Memory. Because they are very fast we could combine the EX-stage and the MEM-stage into one stage called EXMEM. Also to make the problem simpler, in this question we don’t consider forwarding help for BEQ instructions. Hence the FU_Br in ID stage has been removed. A BEQ instruction is stalled until the dependency is resolved. On the next page, a partially modified 4-stage design is presented. The HDU is not needed in this design and is removed. The input connections to the FU and HDU_Br are reduced.

Complete the forwarding paths to carry forwarding data to the forwarding MUXes and also input connections to the FU (forwarding unit) on the next page.

4.2  Compare and contrast the 5-stage pipeline design of lab #6 with the 4-stage pipeline design on the next page.

4.2.1  Unlike in the 5-stage pipeline, we do not need the regular HDU for LW dependency in the 4-stage pipeline because ____________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

However, we still need HDU_Br to stall the BEQ instructions. Answer the following questions about stalling happened in the instruction sequences:

Stream #1:

lw  $4 , $3(40) ;
add $10, $4 , $6 ;

For this stream #1, _______ clock cycles is needed for stalling.
Remark: ____________________________________________

_______________________________________________________________________

_______________________________________________________________________

_______________________________________________________________________

Stream #2:

lw  $4 , $3(40) ;
beq $10, $4 , loop1 ;

For this stream #2, _______ clock cycles is needed for stalling.
Remark: ____________________________________________

_______________________________________________________________________

_______________________________________________________________________

_______________________________________________________________________
Stream #3:

add $4, $3, $2 ;
beq $10, $4, loop1 ;

For this stream #3, ________ clock cycles used for stalling.
Remark: ____________________________________________________________
_____________________________________________________________________
_____________________________________________________________________

4.2.2 In the 5-stage pipeline, the **PCWrite** is under the control of ____________________________
____________________________________ (HDU/HDU_Br/FU/FU_Br/Successful Branch/
Successful Jump/Combination of these/none of these/none, no need to control, activated all the
time).
In the 4-stage pipeline, the PCWrite is under the control of ____________________________
_____________________________________ (HDU/HDU_Br/FU/FU_Br/Successful Branch/
Successful Jump/Combination of these/none of these/none, no need to control, activated all the
time).

4.2.3 The forwarding unit (FU) in the case of the 5-stage pipeline has _____ (0/1/2/3/4/5/6) ________
(1-bit/2-bit/3-bit/4-bit/5-bit/32-bit) comparators where as the FU in the case of the 4-stage
pipeline has _____ (0/1/2/3/4/5/6) ________ (1-bit/2-bit/3-bit/4-bit/5-bit/32-bit) comparators.
The FU in the case of the 4-stage pipeline produces ____________________ (one/two) outputs,
of size __________ (1-bit / each 1-bit / 2-bit / each 2-bit) to control the forwarding muxes.

4.2.4 The HDU_Br (Hazard Detection Unit assisting beq) in the case of the 5-stage pipeline has
_____ (0/1/2/3/4/5/6) ________ (1-bit/2-bit/3-bit/4-bit/5-bit/32-bit) comparators where as the
same in the case of the 4-stage pipeline has _____ (0/1/2/3/4/5/6) ________ (1-bit/2-bit/3-bit/4-
bit/5-bit/32-bit) comparators.

4.2.5 _______ (Like / Unlike) in the case of the 5-stage pipeline, we __________ (need / don’t need) prioritization in the 4-stage
pipeline in providing forwarding help to the instr #3 in the
sequence of add instructions on the right.

4.2.6 If the clock frequency is the same for the two pipelines and we ignore the control (branch) hazard,
the performance of the 4-stage pipeline is ____________________________
(better than / equal to / worse than / sometimes better than and sometimes worse than) the 5-stage
pipeline performance.
Explain. ________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________

4.2.7 In the 4-stage pipeline, since the ALU and the Memory are both in one stage, they can work
simultaneously and this merging of ALU with Memory in a single stage does not call for
extending the clock period (even if we use the original ALU and Data memory which are NOT
fast). TRUE / FALSE Explain. ______________________________________________
Late Branch (OLD Lab6) Pipelined CPU (Late Branch from 1st Ed.) for the EE457 class Lab #6

3/26/2000

Original drawing provided by Prof. Dubois
Early Branch

Designed by: Gandhi Puvvada

Drawn by: Wei-chen Hsu

Detailed implementation of Early Branch suggested in 3rd Ed.

Branch

0 1

0 1

0 1

0 1

0 1

0 1

0 1

0 1

0 1

0 1

0 1