CS 152
Computer Architecture and Engineering

Lecture 9 – Pipelining III

2006-9-26

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: Udam Saini and Jue Sun

www-inst.eecs.berkeley.edu/~cs152/
Recall: First Lecture ...

Our goal for Fall 06:

All projects successful: We want every group to get every CPU working.
Last time: A Hazard Taxonomy

- **Structural** Hazards
- **Data** Hazards (RAW, WAR, WAW)
- **Control** Hazards (taken branches and jumps)

On each clock cycle, we must **detect the presence of all of these hazards**, and **resolve them before they break the “contract with the programmer”**.
Last Time: Hazard Resolution Toolkit

- **Stall** earlier instructions in pipeline.
- **Forward** results computed in later pipeline stages to earlier stages.
- **Add** new hardware or **rearrange** hardware design to eliminate hazard.
- **Change ISA** to eliminate hazard.
- **Kill** earlier instructions in pipeline.
- **Make hardware handle** concurrent requests to eliminate hazard.
Specifications for Lab 3

At-risk hazards for Lab 3

Preferred hazard resolution tools.

Tips for control design
### Lab 3: ISA Specifications

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

**Single “delay slot”**

**No load “delay slot”**
Remember: Online MIPS documentation

The level of detail needed for a pipelined design can only be found in this document.

<table>
<thead>
<tr>
<th>And</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>rs</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

**Format:** \( \text{AND \ rd, rs, rt} \)

**Purpose:**
To do a bitwise logical AND

**Description:** \( \text{rd} \leftarrow \text{rs} \ \text{AND} \ \text{rt} \)

The contents of GPR \( rs \) are combined with the contents of GPR \( rt \) in a bitwise logical AND operation. The result is placed into GPR \( rd \).

**Restrictions:**
None

**Operation:**
\[
\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs}] \ \text{and} \ \text{GPR}[\text{rt}] 
\]

**Exceptions:**
None
Hazard Diagnosis
Data Hazards: Read After Write

Read After Write (RAW) hazards. Instruction I2 expects to read a data value written by an earlier instruction, but I2 executes “too early” and reads the wrong copy of the data.

Lab 3 solution: use forwarding heavily, fall back on stalling when forwarding won’t work or slows down the critical path too much.
Full bypass network...
Common bug: Multiple forwards ...

Which do we forward from?

Let's consider the following operations:

1. **ADD R4, R3, R2**
2. **OR R2, R3, R1**
3. **AND R2, R2, R1**

Which do we forward from?

- **ID (Decode)**
- **EX**
- **MEM**
- **WB**

**Mux, Logic**

**RegFile**

**HL7**

**Ext**

**Data Memory**

**Din**

**Dout**

**MemToReg**
Data Hazards: WAR and WAW ...

Write After Read (WAR) hazards. Instruction I2 expects to write over a data value after an earlier instruction I1 reads it. But instead, I2 writes too early, and I1 sees the new value.

Write After Write (WAW) hazards. Instruction I2 writes over data an earlier instruction I1 also writes. But instead, I1 writes after I2, and the final data value is incorrect.

WAR and WAW not possible in our 5-stage pipeline. However, TA test code checks for these, and every semester a few WAR/WAWs are found. Why?
### LW and Hazards

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

No load "delay slot"
Questions about LW and forwarding

Do we need to stall?

ADDIU R1 R1 24
OR R3,R3,R2 LW R1 128(R29)

From WB

Mux, Logic

RegFile
rs1
rs2
wd
WE
rd1
rd2

Data Memory
Addr
Din
Dout
WE
MemToReg

IR
Ext

B
M
A
Y
R

IR
EX
MEM
WB

IR

UC Regents Fall 2006 © UCB
Questions about LW and forwarding

Do we need to stall?

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

From WB

Mux, Logic

RegFile
rs1
rs2
ws
wd
rd1
rd2

WE

Data Memory

IR

MemToReg

IR

WE, MemToReg

IR

IR

IR

IR

DIR (Decode)

EX

MEM

WB

A

M

Y

B

Data Memory

Addr

Din

Dout

WE

MemToReg

ADDIU R1 R1 24
LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?

Questions about LW and forwarding

ADDIU R1 R1 24

LW R1 128(R29) OR R1,R3,R1

Do we need to stall?
Resolving a RAW hazard by stalling

Sample program

ADD R4, R3, R2
OR R5, R4, R2

Let ADD proceed to WB stage, so that R4 is written to regfile.

Keep executing OR instruction until R4 is ready. Until then, send NOPS to IR 2/3.

Freeze PC and IR until stall is over.

New datapath hardware

(1) Mux into IR 2/3 to feed in NOP.

(2) Write enable on PC and IR 1/2
UDAM'S GROUP, SPR 05 ... Members

Bryant

Michael

Udam

Daniel
Problem 5

• Stalling Logic was incorrect
• Solution: Break up signal into sub-wires, to reduce confusion and facilitate debugging.
Problems

Don’t fall into trap of using one giant module

– Makes it really hard to find problems
– Too many things are happening at the same time
– To solve this, break things apart into sub-modules and use layers of abstraction.
Stalling problems ...

**SYNCMEISTER GROUP, FALL 05**

- Make sure control signals are synched with corresponding data. Sometimes control signals must be delayed.

- Take extra precautions so that stale data is not reintroduced into your processor pipeline.
“Synchronous” memory makes it harder ...

For this design, you must use new RAM files for your instruction and data memories. **Note that Verilog files that we refer to in this lab are all in the m:\lab3\ directory.** Start by reading the Readme in m:\lab3\Lab3Help. The new RAMs (called sdatamem.v and sinstrmem.v) are *fully synchronous*. This means that you must set up the address and any data to be written *before* the edge of the clock. Both reads and writes are synchronous in this way. Keep this in mind when you work on your pipeline. One way to view the result, is that some of the registers that you see in the pipeline diagrams that we show you (for instance, the PC or the S and D registers) are partially duplicated in the RAM block. This means, for instance, that you still need to keep a separate PC register, but that you also need to pipe the value of the address *before* the PC register to the actual RAM block; on a clock edge, the new address will be clocked into both the PC register and the internal address registers of the RAM.

A common error students make when using the synchronous RAMs is that they design a 5-stage pipeline that has 6 stages (oops). Examine your design carefully to make sure you have not made this mistake.
Synchronous Memory Reads ...

Lab 3 Synchronous Memory

Lab 2 Asynchronous Memory

Data Memory

Addr

Dout
Problems

The Next PC Calculation

- Have to use this when start using synchronous memory
- Very hard to get right when dealing with stalls
- Try to get an understanding of the dynamics of this early on, and don’t start writing Verilog until you do.
- Make designs flexible for stalls early on.
Branches and Hazards

Single “delay slot”

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

24
Recall: Control hazard and hardware

Stage #1  
Instr Fetch

Stage #2  
Decode & Reg Fetch

Stage #3

To branch control logic

PC  
D  
O

Instr Mem

Addr  Data

0x4

IR

IR

IR

==

RegFile

rs1

rs2

ws

wd

rd1

rd2

WE

Ext

A

M

B

UC Regents Fall 2006 © UCB
Recall: After more hardware, change ISA.

Sample Program (ISA w/o branch delay slot)

I1: BEQ R4,R3,25
I2: AND R6,R5,R4
I3: SUB R1,R9,R8

ID stage computes if branch is taken
If branch is taken, this instruction MUST NOT complete!
Questions about branch and forwards

BEQ R1 R3 label

Will this work as shown?

To branch control logic

Mux, Logic

ID (Decode)
Q. Why might this be hard (I)?

A. Delay slot logic.
Lessons learned

- Pipelining is hard
- Study every instruction
- Write test code in advance
- Think about interactions ...
The Break Instruction
Lab 3: ISA Specifications

Also: RESET signal, BREAK release signal, etc ...

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

The **break** instruction is special. See COD for its bitfield. Although this is normally an exception-causing instruction, you should treat it more like a halt instruction. After being *decoded*, the break instruction should freeze the pipeline from advancing further. This means that the PC will not advance further, the break instruction will stay in the decode stage, and later instructions will drain from the pipeline as they complete. The proper terminology for this is that the **break** instruction will "stall" in the decode stage. Assume that there will be a single input signal called "release" that comes from outside. When it is high, you should release a blocked **break** instruction exactly once (you need to
Q. Why might this be hard (I)?

A. **BREAK** and delay slot logic.

Diagram showing the pipeline stages of a processor, including the ID (Decode), EX, MEM, and WB stages, with arrows connecting the stages and labeled instructions. The diagram includes various components like RegFile, Mux, Logic, and Data Memory, with data paths marked with arrows and labels.
Why might this be hard (II)?

A. **BREAK release is tricky.**
Problems

Break

– Getting this to work correctly can be tricky
– Make sure not to skip the instructions coming right before or after the break
– In simulation, we used a continuous assignment for the break release, but on the board, this was registered, so we got different results between the two.
Reset
From Lab 3 ...

Make sure that the RESET line causes important processor state to be reset! Remember that "initial" blocks in Verilog will be ignored by the synthesizer. Many bugs can be introduced when registers contain random initial state! One obvious thing that must be reset is the PC. Are there other things?

- The reset signal should correctly affect the phantom pipeline registers inside of synchronous memories.
- You should be able to reset the CPU and the TFTP/FPGA_TOP logic independently. The RESET signal on switch 5 should only reset the CPU, not the TFTP/FPGA_TOP logic.
- The RESET and BREAK switches should work correctly with SINGLE_CLOCK. For example, assume CLK_SOURCE is 0 (putting us in single-stepping mode), and that initially no pushbuttons are pushed. Then, you press RESET (and keep it depressed). The processor should not reset (yet). However, when you press the SINGLE_CLOCK pushbutton, your processor should reset. If you then let go of the reset pushbutton, and then release the SINGLE_CLOCK pushbutton, and then press the SINGLE_CLOCK pushbutton again, the processor should see a clock edge unqualified by the reset signal. BREAK should interact with SINGLE_CLOCK in the same way.
Problem 2

- Pressing Reset and Step buttons caused odd errors on the board.
- Solution: Put Reset and Release on the processor clock. When in stepping mode, make Reset and Release the raw signal.
Reset & PC

- PC counter at reset
  - Especially on the board

Watch the first instruction
  - Don’t lose or repeat

Behavior under stalling
  - Different stalls may affect PC differently
Clocks
Finally, the CLOCK net for your pipeline should be connected either to the clk from the DLL off of the XILINX board or to your debounced SINGLE_CLOCK signal. Let the first switch of the second set of 8 dipswitches (switch10) be the choice (call this signal "CLK_SOURCE"):

```
processor_clock = CLK_SOURCE ? LAB_CLK: SINGLE_CLOCK;
```

Plus, the TFTP clock ...
Problems

Handling Clock Boundaries

– Make sure to look at how positive edges of different clocks can interact (ie ButtonParser, SDRAM arbiter, etc.)

– Make sure to use different clocks when doing simulation to try to root out these type of bugs.
Problems

Memory Mapped I/O
- Difficult to get time correctly
- Pay attention to which signal are synchronous and which are asynchronous
- Understand how this module interacts with other modules in the processor
Problems: Clocks

- Keep your clocks straight.
- Don’t mix clock signals.
- You can only have 4 clocks; sometimes putting on board might unintentionally go over clock limit.
Control Implementation
Recall: What is single cycle control?

Combinational Logic (Only Gates, No Flip Flops)
Just specify logic functions!
In pipelines, all IR registers are used

A “conceptual” design -- for shortest critical path, IR registers may hold decoded info, not the complete 32-bit instruction
Two goals when specifying control logic

**Bug-free:** One “0” that should be a “1” in the control logic function breaks contract with the programmer.

**Efficient:** Logic function specification should map to hardware with good performance properties: fast, small, low power, etc.

Should be easy for humans to read and understand: sensible signal names, symbolic constants ...
Midterm week begins on Thursday ...

<table>
<thead>
<tr>
<th>Th 9/28</th>
<th>Midterm Review Session in Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>F 9/29</td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
</tr>
<tr>
<td>Sa 9/30</td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
<tr>
<td>Su 10/1</td>
<td></td>
</tr>
<tr>
<td>M 10/2</td>
<td>Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)</td>
</tr>
<tr>
<td>T 10/3</td>
<td></td>
</tr>
<tr>
<td>W 10/4</td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
</tr>
<tr>
<td>Th 10/5</td>
<td>HW 2 available (due at Midterm II review session)</td>
</tr>
<tr>
<td>F 10/6</td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
</tbody>
</table>

**HW graded on effort**

**Thursday review session. Will cover format, material, and ground rules for test.**

**Midterm (6-9, 306 Soda), no class that day.**
And concurrently, Lab 3 deadlines ... 

<table>
<thead>
<tr>
<th>Day</th>
<th>Activity</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/28</td>
<td>Midterm Review Session in Class</td>
<td>HW 1 due in class</td>
</tr>
<tr>
<td>F 9/29</td>
<td></td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM</td>
</tr>
<tr>
<td>Sa 9/30</td>
<td></td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
<tr>
<td>Su 10/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M 10/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T 10/3</td>
<td>Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)</td>
<td></td>
</tr>
<tr>
<td>W 10/4</td>
<td></td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
</tr>
<tr>
<td>Th 10/5</td>
<td>VLSI</td>
<td>HW 2 available (due at midterm II review session)</td>
</tr>
<tr>
<td>F 10/6</td>
<td></td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
</tbody>
</table>

Lab 2 team evals, Lab 3 design document, Weekend: start design work
## Lab 2 Team Evaluations due Thursday

<table>
<thead>
<tr>
<th>Name</th>
<th>Evaluation</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>30%</td>
<td>Bill <em>did</em> design the modules he was assigned to do, I'll give him that much. But he was missing in action during the 20 hours we spent in the lab doing integration and bug-fixes. This may be related to his behavior during our first meeting. Bill started yelling and screaming after we decided not to use the gated-clock reset scheme he proposed for the processor -- and then he stomped out of the room and slammed the door.</td>
</tr>
<tr>
<td>Sue</td>
<td>100%</td>
<td>5 hours into our final debugging section, Sue spotted the bug that kept our processor from working -- a bug that was my mistake. She saved our group. She also designed her own bug-free modules on-time, and set up our CVS system.</td>
</tr>
<tr>
<td>Joe</td>
<td>100%</td>
<td>A solid contribution from Joe -- he did everything I would expect from a good team member. He did many of the testing and integration tasks that Bill was assigned to do but never did.</td>
</tr>
</tbody>
</table>
Lab 3 Design Document Details

1. The identity of the **spokesperson** for the lab, and a roster of group members. The responsibility of the spokesperson is to communicate questions to the TA and reply to questions from the TA. Choose a different spokesperson than the one you had for Lab 2.

2. A short description of the structure of the design. The description will be accompanied by preliminary high-level schematics of your datapath, and a preliminary discussion of the controller.

3. A description of the unit test benches and multi-unit test benches you intend to create for your processor, and a description of machine language programs you intent to write to use in complete processor testing. Also, a **test plan**, using the epoch charting method shown in the 9/7 lecture, that shows when you plan to run each type of test.

4. A tentative **division of labor**, showing the tasks each group member intends to do.

5. The "**paranoia**" section: discuss potential areas of difficulty in the lab. An early guess of critical timing paths for the design should be a part of this section.
### Lab 3 deadlines after the mid-term ...

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/28</td>
<td>Midterm Review Session in Class</td>
<td></td>
</tr>
<tr>
<td>9/29</td>
<td></td>
<td>HW 1 due in class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab 3: Preliminary Design Document and Team Evaluations due to TAs,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11:59PM</td>
</tr>
<tr>
<td>9/30</td>
<td></td>
<td>Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
<tr>
<td>10/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/3</td>
<td>Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)</td>
<td>Lab 3: Final Design Document due to TAs, 11:59PM</td>
</tr>
<tr>
<td>10/4</td>
<td>VLSI</td>
<td>HW 2 available (due at Midterm II review session)</td>
</tr>
<tr>
<td>10/5</td>
<td>VLSI</td>
<td>Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory</td>
</tr>
<tr>
<td>10/6</td>
<td>VLSI</td>
<td></td>
</tr>
</tbody>
</table>

Lab 3 design doc, checkoffs, later in week ...

---

CS 152 L7: Pipelining I