Lecture 10: Memory-mapped I/O and Lab 4

David Black-Schaffer
davidbbs@stanford.edu
EE183 Spring 2003

Overview

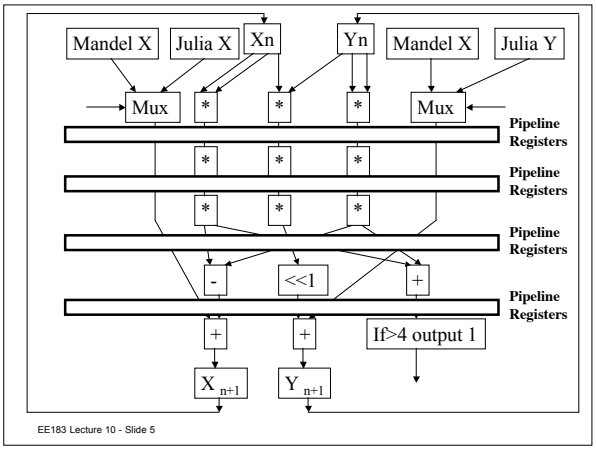
- Pipelining in Lab 3/4
  - Do a little bit of work in each stage (fast)
  - Keep the control simple by passing it down the pipeline
  - Deal with Data Hazards through forwarding
- Memory-mapped I/O
  - MUX the data lines based on the address
  - Use LOAD/STORE to special addresses to transfer data between external devices

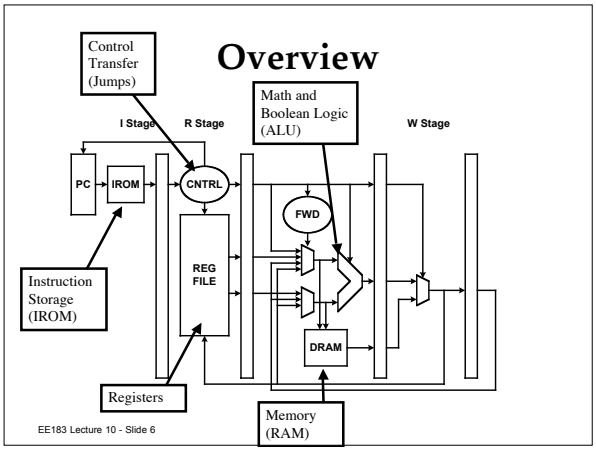
Public Service Announcement

- Xilinx Programmable World
  - Tomorrow, May 6th
  - http://www.xilinx.com/events/pw2003/index.htm - free!
- Guest Lecture
  - Wednesday, May 7th
    Gary Spivey on ASIC & FPGA Design for Speed
  - Also, they are hiring. Info session in Tressider at noon.

Logistics

- Lab 2 writeup due tonight by midnight
- Very cool guest lecture tomorrow… (want to break that 200MHz barrier in lab 3? ;)
- Next Monday is the last lecture.





Instruction Format

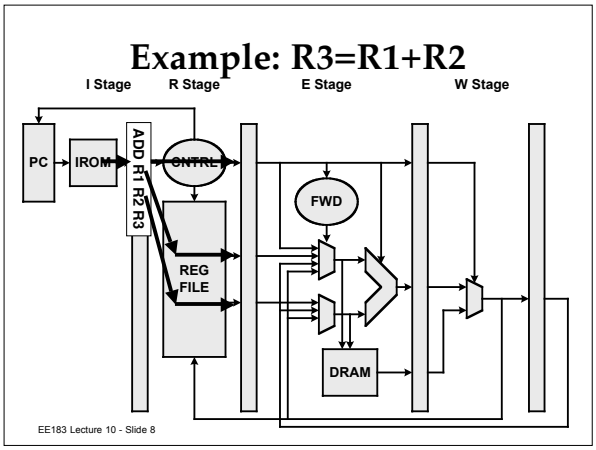
- Different for different types of instructions
- ALU instructions (similar to others):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1														
0	1	WC			OP					RA			RB		

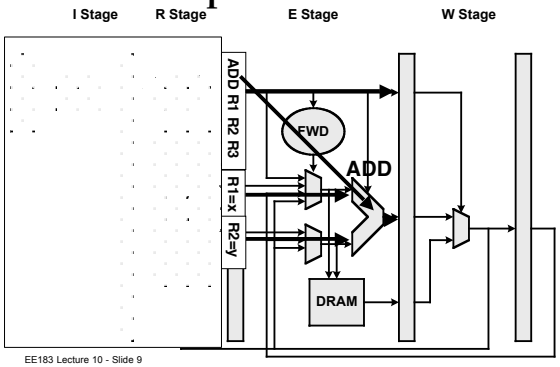
Writeback Register | Operation Code | Source Register A | Source Register B

- See Lab 3 handout

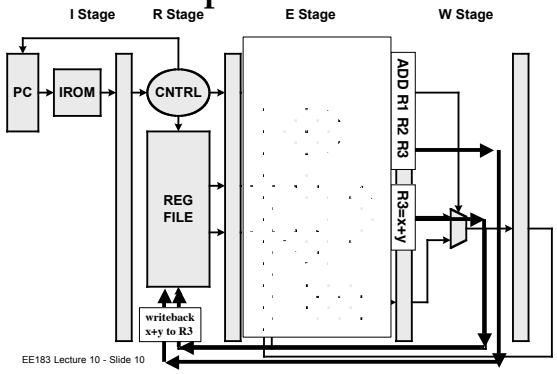
EE183 Lecture 10 - Slide 7



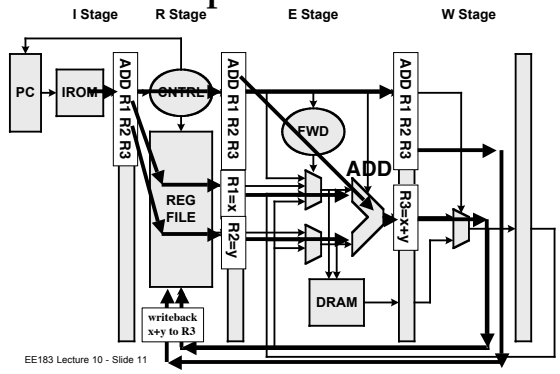
Example: R3=R1+R2



Example: R3=R1+R2



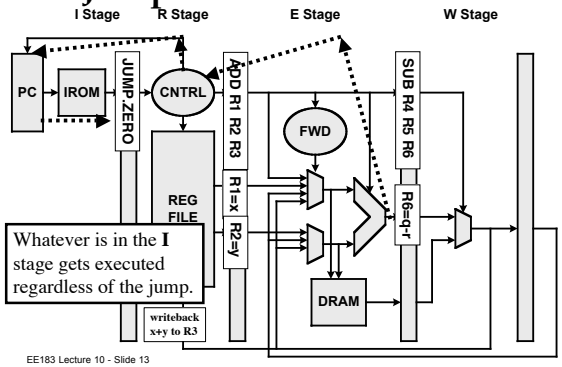
Example: R3=R1+R2



Notes

- Do a little bit of the work in each stage
- Carry it along through the pipeline registers
- We want each stage to be as fast as possible and as independent as possible
- Can we do this?
- Not entirely…

Jumps - Control Hazards

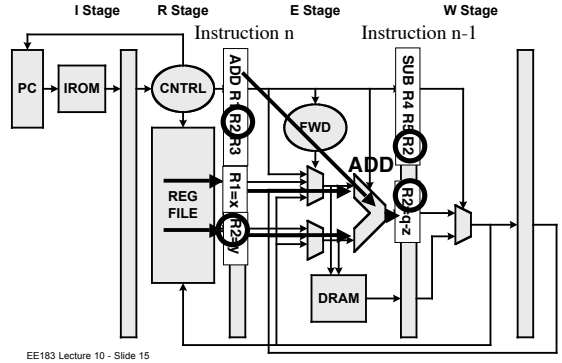


Whatever is in the **I** stage gets executed regardless of the jump.

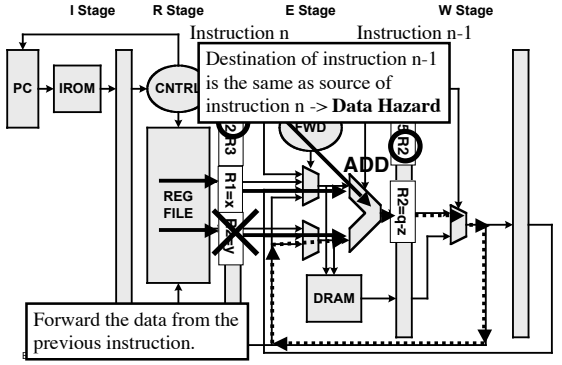
Control Hazards Solution

- Insert a NOP after each branch statement (JUMP)
- Now we don't care if the next instruction is executed because it never does anything

Data Hazards



Data Hazards



Tricky Parts

- **Prelab for Lab 3**:
- Forwarding logic
  - Really only a few cases, but it takes a bit of thinking to work it out
- Branching (Jump) logic:
  - Not too complicated either, but if you get it wrong it will be hard to debug

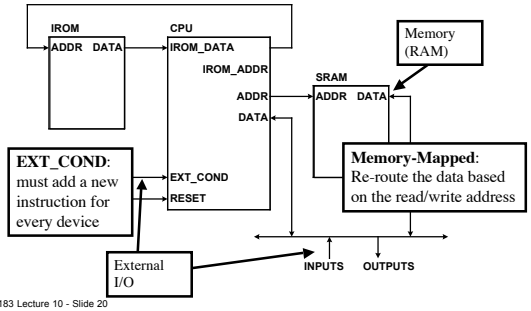
Lab 3 Requirements

- Implement the processor with forwarding
- Demo the three test programs
  - Output the result register to the LEDs
- Implement a Timer
  - Free-running counter which the processor can use to do things at a specified interval much like the slow clock in Labs 1/2.
  - Can be either a new JUMP condition or a memory-mapped device
- Implement a memory-mapped VGA display
- Demo a program which uses the VGA and Timer

Today

- Memory-mapped I/O
- Final Project (Lab 4)

Processor I/O



Memory-mapped I/O

- Add logic to look for LOAD/STORE to a particular address/range of addresses
- Re-route the signals to the external device
- Example:
  - If I do a STORE to 0xFFF then send that data not to the DRAM but to the VGA
  - If I do a LOAD from 0xFFD then take the data not from the DRAM but from the Timer

Implementation

- Simple: look at the RAM address bus and re-route the data and RE/WE signals based on the address (some MUXes)
- Have the external devices respond to these signals to send/receive data

Usage

```
LOADLIT R1, 0xfff  # Define our VGA memory-mapped
                   #  address for sending the VGA
                   #  {x,y} to the VGA module

ZEROS R0           # Initialize our {x,y} to zero
# Loop writing to the VGA
_LOOP
STORE R1, R0       # Write the value of R0 to the VGA
                   #  this is our {x,y} location
INCA R0, R0        # Increment the {x,y} location
JUMP _LOOP
```

- How do we control the color?

Color

- Instead of just one VGA location {x,y} address, let's have two:
  0xFFF - VGA {x,y} {6-bits x,6-bits y}
  0xFFE - VGA color (4-bits)
- Now the VGA module is more complicated, but we can control the color

Usage

```
LOADLIT R1, 0xfff       # Define our VGA memory-mapped
                        #  addresss the VGA {x,y}
LOADLIT R2, 0xffe       #  and the VGA color

LOADLIT R3, 0x001       # Define a constant for our color



ZEROS R0                # Initialize our {x,y} to zero
# Loop writing to the VGA
_LOOP
STORE R1, R0            # Write the value of R0 to the VGA
                        #  this is our {x,y} location
STORE R2, R3            # Write the color to the VGA

INCA R0, R0             # Increment the {x,y} location
JUMP _LOOP
```


Memory-Mapped I/O

- Route the STORE/LOAD data to different places based on the address of the STORE/LOAD
- Bunch of MUXes on the Address/Data lines
- Good way to communicate with many devices (PCI works this way)
- Works better with Tri-stated lines

Memory-Mapped I/O

Any questions?

EE183 Final Project

- Final Project = Lab 4
- Extend Lab 3 processor to add functionality and increase performance
  - 20 additional points on grading for difficulty
- Final demo program
- Two main parts
  - Additional Instructions (JAL/JR)
  - Something cool…

EE183 Final Projects (Lab 4)

- Part 1: Add **JR** (jump register) and **JAL** (jump-and-link) to the processor
  - JAL can have a fixed register to write to, but JR must take an arbitrary register
  - Find a good place in the ISA to encode these instructions
  - Modify the assembler
  - Produce a simple demo program

EE183 Final Projects (Lab 4)

- Part 2: extend & use the processor
- Three options:
  - **Accelerate** a slow software process in hardware (DES, FIR filters, fractals, cache, graphics, etc.)
  - **Interface** to some external hardware in a cool manner (keyboards, audio, video, etc.)
  - Something else cool
- Lab 4 pre-lab due Wed. May 21
  - Overview of what you are going to do and how
  - Talk to us about how doable your project is first!

Goals for Lab 4

- Accelerate:
  - **Quantitatively** assess the performance of your implementation
  - Performance registers to measure effectiveness
  - Compare to other (possible) implementations
  - Works very nicely for implementing the fractal lab on the processor

Goals for Lab 4

- Interface:
  - Better be something pretty cool
  - Show how the processor works in the system. (I.e., if it can be better done without a processor then it isn't a good project.)
  - Elegant interface to the processor.
  - Probably a memory-mapped interface, but make it a smart one.
  - Nice for audio filters/keyboard interfaces/graphics

Goals for Lab 4

- Something else cool:
  - Really cool
  - If you're excited about it come talk to us and we'll see if it is a good project
  - Wide open…

Midterm

- Don't forget the midterm a week from this Wednesday.
- 7-9pm in SEQ 102 (next door)

- 45 minute "quiz"
- On the material in the lectures.

Lecture 6 Key Points

- Pass intermediate results and control info down the pipeline to make it simple
- MUX the RAM data lines based on the address to interface to external devices

- Logistics
  - **Guest lecture on Speed on Wed.**
  - **Xilinx Programmable World tomorrow**
  - **Last EE183 Lecture ever next Monday**