EECS150 - Digital Design
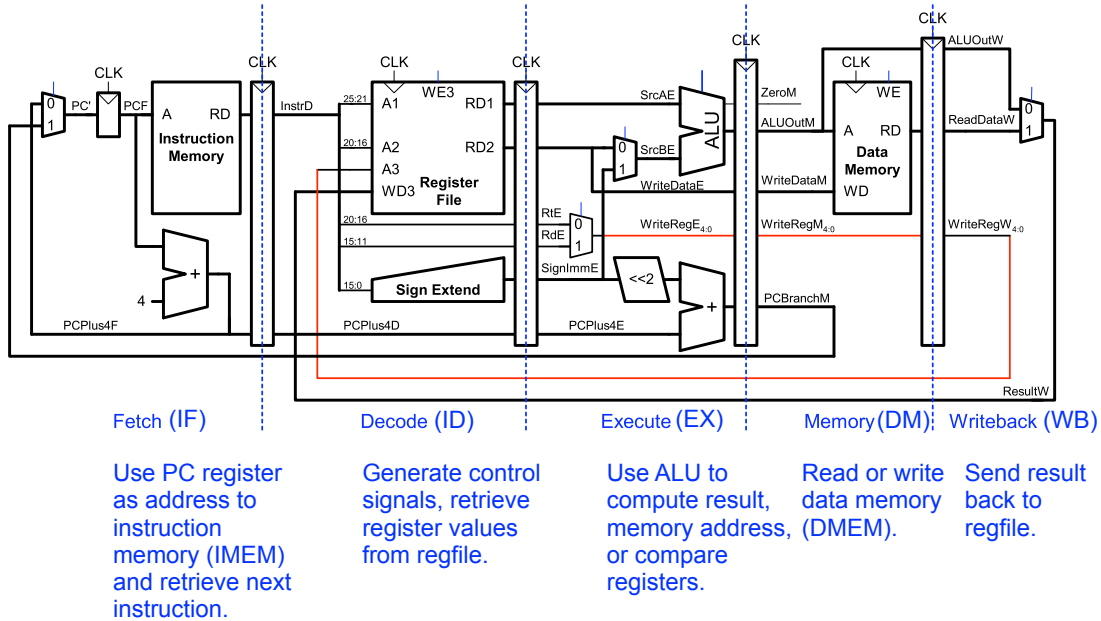## Lecture 10 - Project Introduction Part 1

Feb 17, 2011

John Wawrzynek

Project Overview
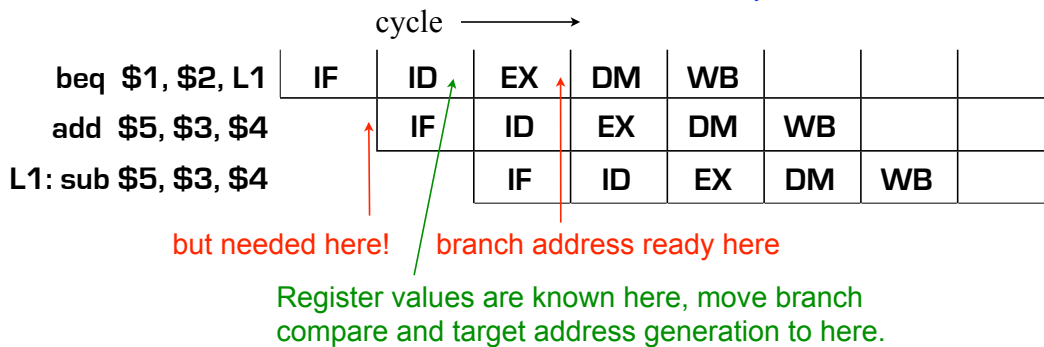
A. MIPS150 pipeline structure

B. Serial Interface

C. Memories, project memories and FPGAs

D. Video subsystem

E. Ethernet Interface

F. Project specification and grading standard

MIPS 5-stage Pipeline Review

Fetch (IF)	Decode (ID)	Execute (EX)	Memory (DM)	Writeback (WB)
Use PC register as address to instruction memory (IMEM) and retrieve next instruction.	Generate control signals, retrieve register values from regfile.	Use ALU to compute result, memory address, or compare registers.	Read or write data memory (DMEM).	Send result back to regfile.

MIPS 5-stage Pipeline

Control Hazard Example

cycle ⟶

	IF	ID	EX	DM	WB		
beq $1, $2, L1	IF	ID	EX	DM	WB		
add $5, $3, $4		IF	ID	EX	DM	WB	
L1: sub $5, $3, $4			IF	ID	EX	DM	WB

but needed here!   branch address ready here

Register values are known here, move branch compare and target address generation to here.
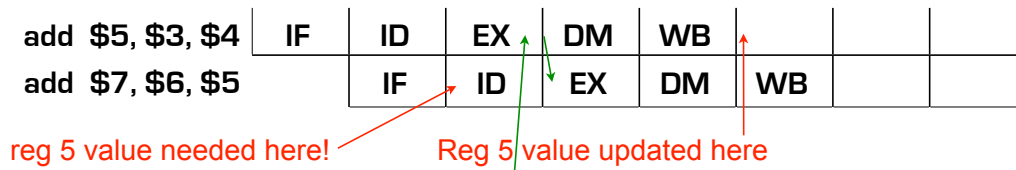
Still one remaining cycle of branch delay. "Architected branch delay slot" on MIPS allows compiler to deal with the delay. Other processors without architected branch-delay slot use branch predictors or pipeline stalling.

MIPS 5-stage Pipeline

Data Hazard Example

add $5, $3, $4	IF	ID	EX	DM	WB			
add $7, $6, $5		IF	ID	EX	DM	WB		

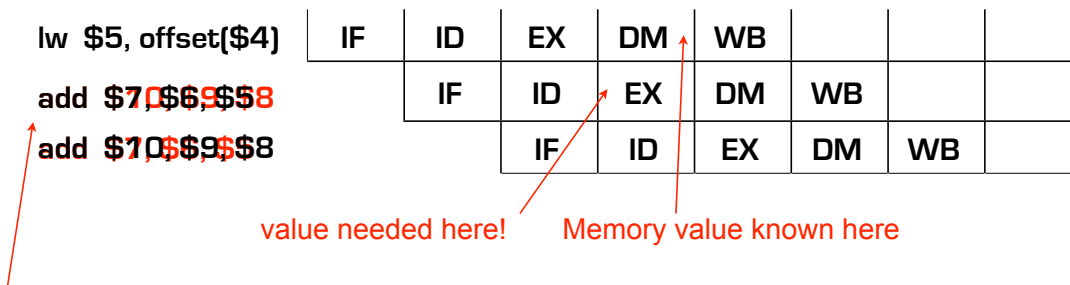reg 5 value needed here!   Reg 5 value updated here

New value is actually known here. Send it directly from the output register
of the the ALU to its input (and also down the pipeline to the register file).

Logic must be added to detect when such a hazard exists and control
multiplexors to forward correct value to ALU.   No alternative except
to stall pipeline (thus hurting performance).

MIPS 5-stage Pipeline

Load Hazard Example

lw $5, offset($4)	IF	ID	EX	DM	WB			
add $7, $6, $5		IF	ID	EX	DM	WB		
add $10, $9, $8			IF	ID	EX	DM	WB	

value needed here!   Memory value known here

"Architected load delay slot" on MIPS allows compiler to deal with the
delay. Note, regfile still needs to be bypassed.

No other alternative except for stalling.

Processor Pipelining

Deeper pipeline example.

IF1	IF2	ID	X1	X2	M1	M2	WB		
	IF1	IF2	ID	X1	X2	M1	M2	WB	

Deeper pipelines => less logic per stage => high clock rate.

But

Deeper pipelines => more hazards => more cost and/or higher CPI.

Cycles per instruction might go up because of unresolvable hazards.

Remember, Performance = # instructions X Frequency$_{clk}$ / CPI

How about shorter pipelines ... Less cost, less performance

MIPS150 Pipeline

The blocks in the datapath with the greatest delay are: IMEM, ALU, and DMEM. Allocate one pipeline stage to each:

| I | X | M |

Use PC register as address to IMEM and retrieve next instruction. Instruction gets stored in a pipeline register, also called "instruction register", in this case.
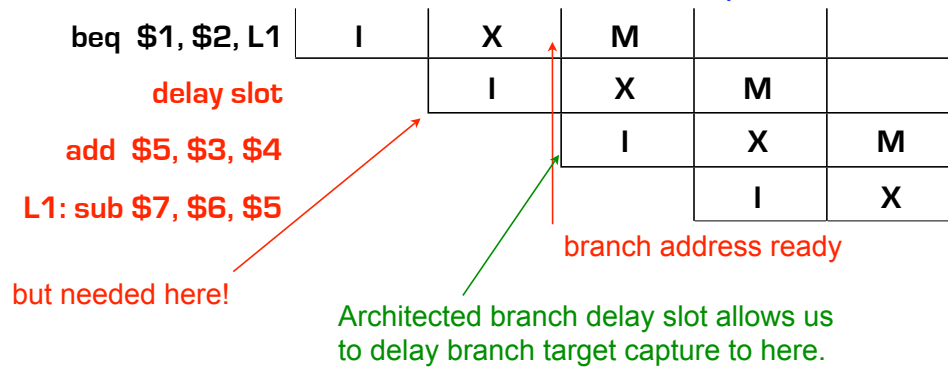
Use ALU to compute result, memory address, or compare registers for branch.

Access data memory or I/O device for load or store. Allow for setup time for register file write.

Most details you will need to work out for yourself. Some details to follow ... In particular, let's look at hazards.
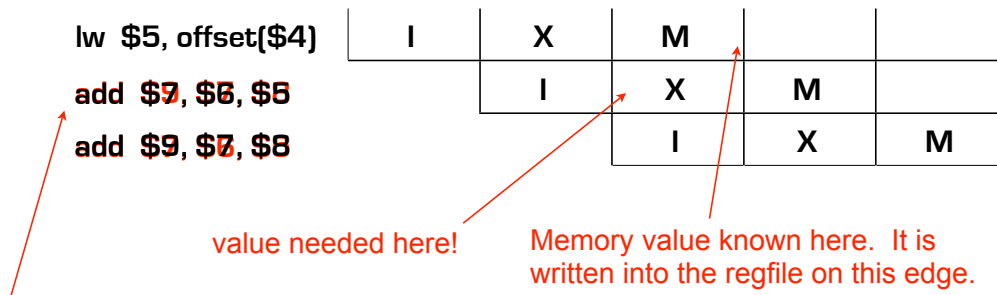
MIPS 3-stage Pipeline

Control Hazard Example

beq $1, $2, L1	I	X	M	
delay slot		I	X	M
add $5, $3, $4			I	X	M
L1: sub $7, $6, $5				I	X

branch address ready

but needed here!

Architected branch delay slot allows us
to delay branch target capture to here.

Therefore no extra logic is required.

MIPS 3-stage Pipeline

Load Hazard

lw $5, offset($4)	I	X	M	
add $7, $6, $5		I	X	M
add $9, $7, $8			I	X	M

value needed here!

Memory value known here. It is
written into the regfile on this edge.

"Architected load delay slot" on MIPS allows compiler to deal with the
delay. No regfile bypassing needed here assuming regfile "write
before read".

MIPS 3-stage Pipeline

Data Hazard

add $5, $3, $4	I	X	M		
add $7, $6, $5		I	X	M	

reg 5 value needed here!   reg 5 value updated here
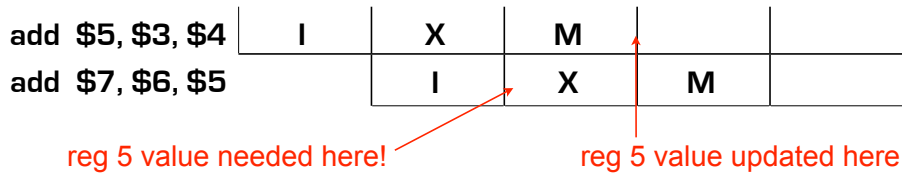
Ways to fix:

1. Stall the pipeline behind first add to wait for result to appear in register file. NOT ALLOWED this semester.

2. Selectively forward ALU result back to input of ALU.

- Need to add mux at input to ALU, add control logic to sense when to activate. A bit complex to design. Check book for details.

Project CPU Pipelining Summary

3-stage pipeline	I instruction fetch	X execute	M access data memory
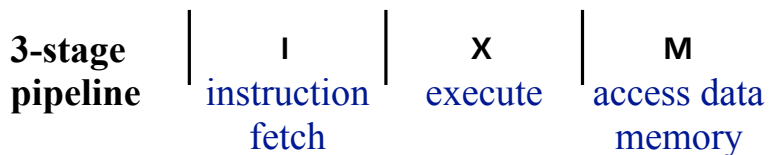
- Pipeline rules:
  - Writes/reads to/from DMem use leading edge of "M"
  - Writes to RegFile use trailing edge of "M"
  - Instruction Decode and Register File access is up to you.
- 1 Load Delay Slot, 1 Branch Delay Slot
  - No Stalling may be used to accommodate pipeline hazards (in final version).
- Other:
  - Target frequency to be announced later (50-100MHz)
 - Minimize cost
  - Posedge clocking only

Background for Lab Next Week

Final Project: Spring 2011
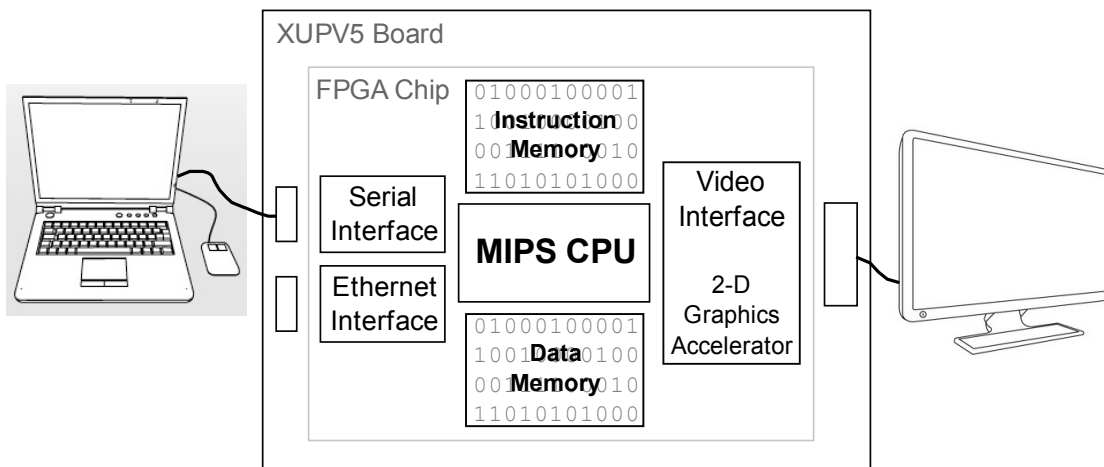
- Executes most commonly used MIPS instructions.
- Pipelined (high performance) implementation.
- Serial console interface for shell interaction, debugging.
- Ethernet interface for high-speed file transfer.
- Video interface for display with 2-D vector graphics acceleration.
- Supported by a C language compiler.

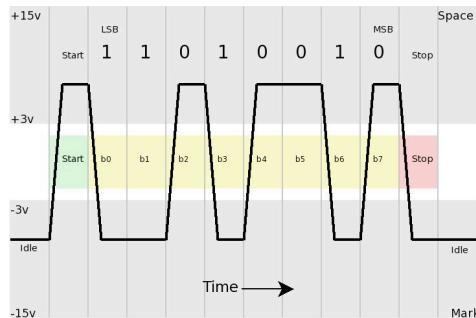# Board-level Physical Serial Port
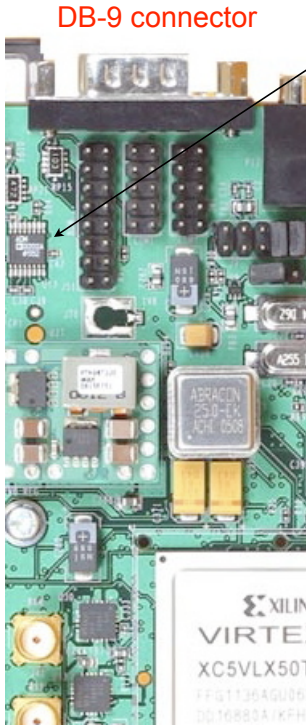
DB-9 connector
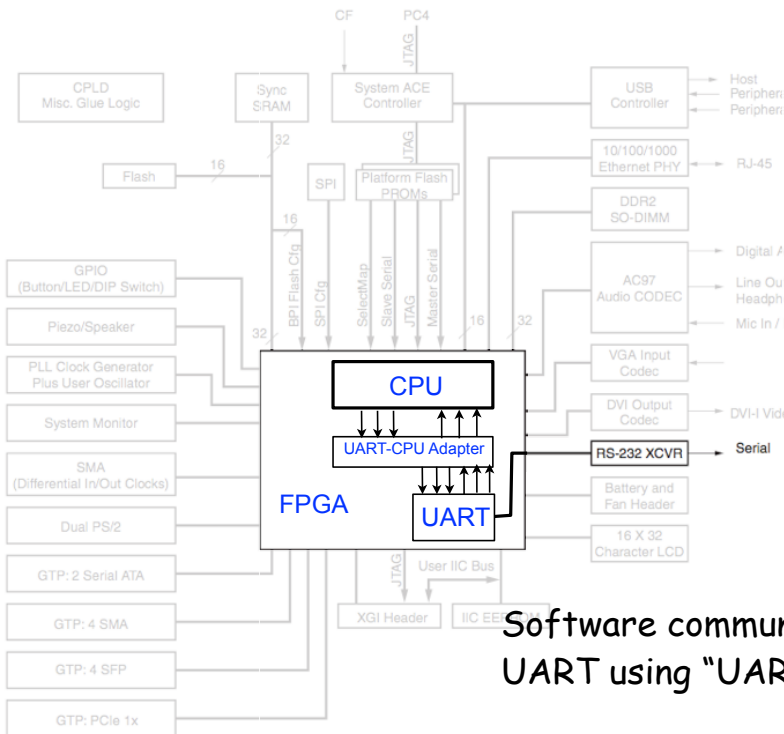
RS-232 Transmitter/Receiver
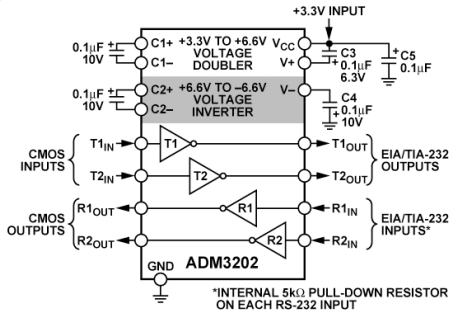
Implements standard signaling voltage levels for serial communication. Allows FPGA board to communicate with any other RS-232 device.

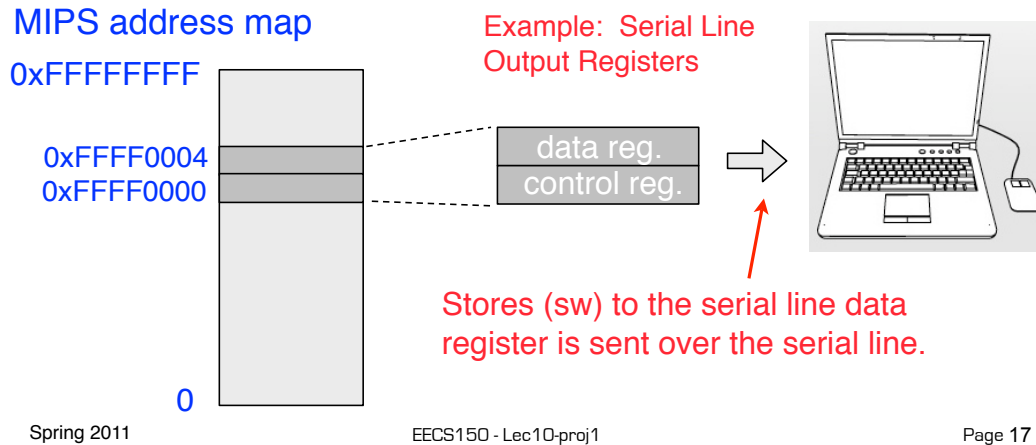Oscilloscope trace of ASCII "K" transmission.

FPGA Serial Port

UART: **Universal Asynchronous Receiver and Transmitter** converts to/from serial format with start/stop bits.

Software communicates with UART using "UART-CPU Adapter".

MIPS uses Memory Mapped I/O

- Certain addresses are not regular memory

- Instead, they correspond to registers in I/O devices

MIPS address map

0xFFFFFFFF

0xFFFF0004
0xFFFF0000

0

Example: Serial Line Output Registers

data reg.
control reg.

Stores (sw) to the serial line data register is sent over the serial line.
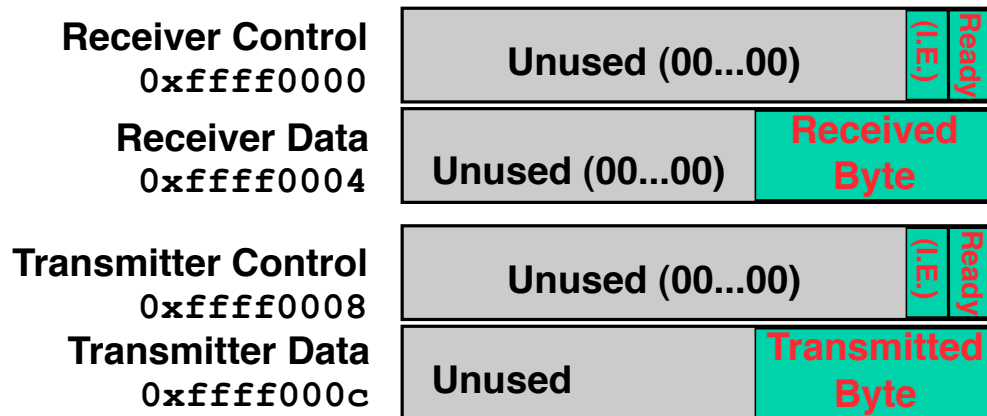
Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - Control Register, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - Data Register, holds data for transfer
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg (0 ⇒ 1) to say its OK
- Processor then loads from (input) or writes to (output) data register

MIPS150 Serial Line Interface

- Serial-Line Interface is a memory-mapped device.

- Modeled after SPIM terminal/keyboard interface.
  - Read from keyboard (<u>receiver</u>); 2 device regs
  - Writes to terminal (<u>transmitter</u>); 2 device regs

Receiver Control
`0xffff0000`

| Unused (00...00) | (I.E.) | Ready |

Receiver Data
`0xffff0004`

| Unused (00...00) | Received Byte |

Transmitter Control
`0xffff0008`

| Unused (00...00) | (I.E.) | Ready |

Transmitter Data
`0xffff000c`

| Unused | Transmitted Byte |

Serial I/O

- Control register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read;
    $1 \Rightarrow 0$ when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character;
    $0 \Rightarrow$ Transmitter still busy writing last char
    - I.E. bit (not used in our implementation)

- Data register rightmost byte has data
  - Receiver: last char from serial port; rest = 0
  - Transmitter: when write rightmost byte, writes goes to serial port.

"Polling" MIPS code

- Input: Read from keyboard into $v0

```
                lui   $t0, 0xffff  #ffff0000
  Waitloop1:    lw    $t1, 0($t0)  #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop1
                lw    $v0, 4($t0)  #data
```

- Output: Write to display from $a0

```
                lui   $t0, 0xffff  #ffff0000
  Waitloop2:    lw    $t1, 8($t0)  #control
                andi  $t1,$t1,0x1
                beq   $t1,$zero, Waitloop2
                sw    $a0, 12($t0) #data
```