

# CS61c – Final Review

Fall 2004

Andy Carle

12/12/2004

## Topics Before Midterm

- C & Malloc
- Memory Management
- MIPS
- Number Representation
- Floating Point
- CAL

## Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- Pipelining
- Caches
- Virtual Memory
- I/O and Performance

## Today's Focus

- Focus on material from after the midterm. Don't forget to go back and look over the old review session and your midterm
- More emphasis on material that we covered just after the midterm, less on stuff from the past few weeks.
  - Lots of stuff on digital logic
  - Like one slide on performance
- My intent is to help you “page in” (no replacement policy, please) material from the second half of the course that you may have long since forgotten

## Topics Since Midterm

- **Digital Logic**
  - **Verilog**
  - **State Machines**
- CPU Design
- Pipelining
- Caches
- Virtual Memory
- I/O and Performance

## Digital Logic (and such)

- Truth Tables
- Boolean Algebra
- Canonical SOP
- Combinational Logic
- State Machines
- Timing Diagrams & Tables
- Programmable Logic Arrays

### Digital Logic – Truth Tables

- A table describing the output of a function for every possible input
- Usually done bitwise when dealing with digital logic
- $2^n$  entries (each input can have two possible states)

### Digital Logic – Boolean Operators

B	A	AND	OR	NAND	NOR	XOR	XNOR	NOT
0	0	0	0	1	1	0	1	1
0	1	0	1	1	0	1	0	0
1	0	1	1	0	0	1	0	0
1	1	1	1	0	0	0	1	1

AB	A+B	(AB)'	(A+B)'	A⊕B	A=B	A'
----	-----	-------	--------	-----	-----	----

### Digital Logic – Boolean Algebra

- An algebraic expression using Boolean operators
  - e.g.  $AB + A'B + A(B+C')$
- Similar laws to normal algebra
  - Distributive:  $AB + A'B + AB + AC'$
  - Idempotent:  $AB + A'B + AC'$
  - Complementarity:  $B + AC'$

### Digital Logic – Boolean Algebra Laws

$x \cdot x = x$	$x + x = 1$	complementarity laws of 0's and 1's identities idempotent law commutativity associativity distribution uniting theorem DeMorgan's Law
$x \cdot 0 = 0$	$x + 1 = 1$	
$x \cdot 1 = x$	$x + 0 = x$	
$x \cdot x = x$	$x + x = x$	
$x \cdot y = y \cdot x$	$x + y = y + x$	
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	
$xy + xz = x$	$(x + y)z = x$	
$x'x = x + y$	$(x + y) = x \cdot y$	

### Digital Logic – Boolean Algebra

Minimize the following using Boolean Algebra simplification rules:

$$s = \overline{((a + ab) + b) + (a + \bar{b})}$$

What does this function do?

### Digital Logic – Boolean Algebra Solution

Simplifies to:

$$s = \overline{ab} + \overline{ab}$$

This is an XOR

#### Digital Logic – Canonical SOP Form

- Standardized form to describe a truth table uniquely
- For every 1 in the output column of the Truth Table have a term in the SOP
- Put every input variable in every term, with the ones that were 0 for that entry in the TT negated

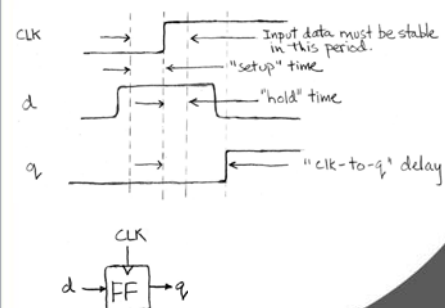
#### Digital Logic – Combinational Logic

- Built from Boolean Algebra operators turned in to gates
- Output is purely a function of current input
- Therefore, can not have memory or remember state
- However, this is perfect for describing a truth table as we have seen it so far!

#### Digital Logic – Sequential Logic & State

- To implement many practical circuits we need some form of “memory”.
- Registers (created from Flip Flops) are our statefull circuit elements.
- Adding state to a circuit introduces a notion of time centered around a clock
  - Complicates Truth Table
  - Necessitates Timing Diagrams

#### Digital Logic – Timing



#### Digital Logic – Finite State Machines

- An abstraction of any system with a finite number of states and logical transitions between them
- Useful when trying to come up with the truth table for a problem
- States in a FSM are represented by data stored in registers

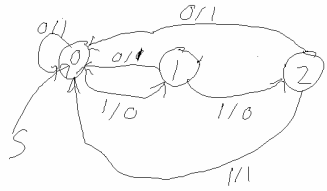
#### Digital Logic – Putting It All Together

##### Exercise

Come up with the FSM, Truth Table (NS/Output), Canonical SOP form, Simplified Boolean Equation, and circuit diagram for this function over a continuous bit stream:

Output 1 if the input was a 0. Output 0 if the input was a 1. Unless it was the 3<sup>rd</sup> 1 in a row, in which case you output 1 (and start over counting 1s.

### FSM Solution – Created At Review



### Solution

- $MSB(NS) = S1' * S0 * n$
- $LSB(NS) = S1' * S0' * n$
- $OUT = S1' * n' + S1 * S0'$

### Digital Logic – Verilog

- Hardware Description Language
- Verilog description is a collection of interconnected modules
- All modules work in parallel
- Structural vs. Behavioral

### Digital Logic – Verilog

#### Exercise

Implement the following circuit in both structural and behavioral Verilog:



### Digital Logic – Verilog

#### Behavioral

```
module pri_enc(in0, in1, in2, e0, e1);
  input in0, in1, in2;
  output e0, e1;
  assign e0 = (~in1 & in0) | in2;
  assign e1 = in1 | in2;
endmodule;
```

### Digital Logic – Verilog

#### Structural

```
module pri_enc(in0, in1, in2, e0, e1);
  input in0, in1, in2;
  output e0, e1;
  wire notIn1, and01;
  not(in1, notIn1);
  and(notIn1, in0, and01);
  or(and01, in2, e0);
  or(in1, in2, e1);
endmodule;
```

### Digital Logic – Programmable Logic Arrays

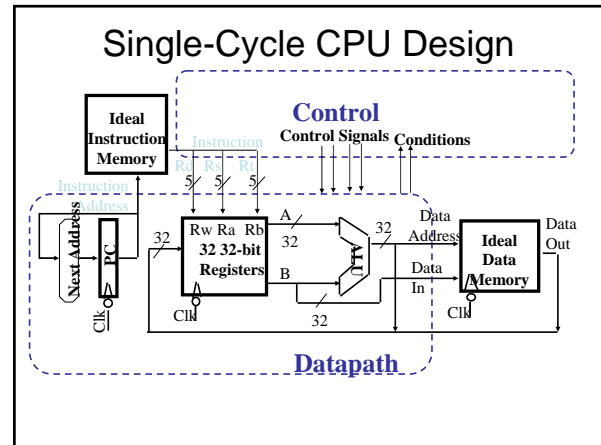
- Creating customized hardware is expensive
- We would like to be able to pre-fabricate a circuit and then allow it to be programmed by the developer
- PLAs are the answer!
- Review how to program one on your own

### Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- Pipelining
- Caches
- Virtual Memory
- I/O and Performance

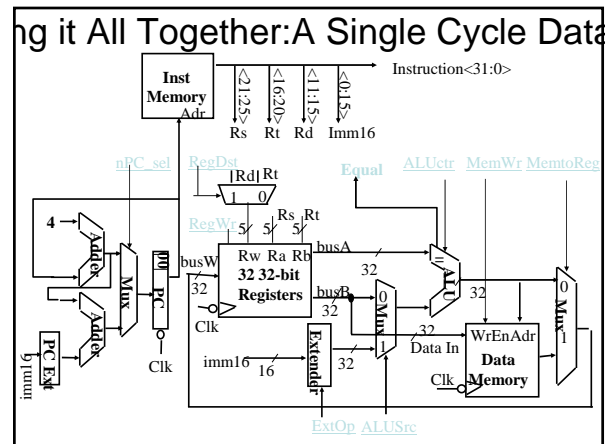
### Single Cycle CPU Design

- Overview Picture
- Two Major Issues
  - Datapath
  - Control
- Control is the hard part, but is made easier by the format of MIPS instructions



### CPU Design – Steps to Design/Understand a CPU

1. Analyze instruction set architecture (ISA) => datapath requirements
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points.
5. Assemble the control logic



### CPU Design – Components of the Datapath

- Memory (MEM)
  - instructions & data
- Registers (R: 32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Extender (sign extend)
- ALU (Add and Sub register or extended immediate)
- Add 4 or extended immediate to PC

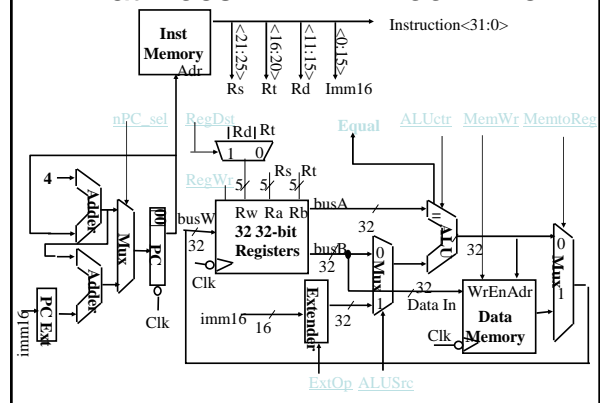
### CPU Design – Control Signals

- Branch: 1 for branch, 0 for other
- ALU control
- MemWrite, MemRead(=MemtoReg): 1 if writing to/reading from memory, 0 if not
- ALUSrc: choice of ALU input; 1 for immed, 0 for reg
- RegWrite: 1 if writing a reg, 0 if not
- RegDst: 1 if output reg is specified in bits 15-11 (R-fmt), 0 if output reg is in bits 20-16 (I-fmt)
- MemtoReg: 1 if writing reg from memory, 0 if writing reg from ALU
- PCSrc: 1 for branch address, 0 for PC+4

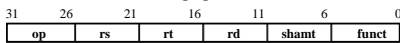
### CPU Design – Instruction Implementation

- Instructions supported (for our sample processor):
  - lw, sw
  - beq
  - R-format (add, sub, and, or, slt)
  - corresponding I-format (addi ...)
- You should be able to,
  - given instructions, write control signals
  - given control signals, write corresponding instructions in MIPS assembly

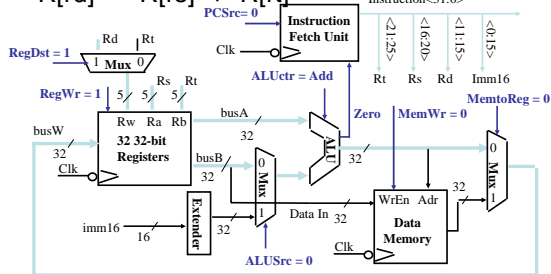
### What Does An ADD Look Like?



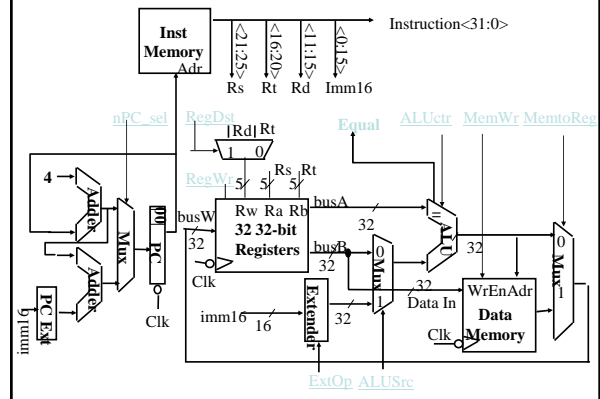
### Add

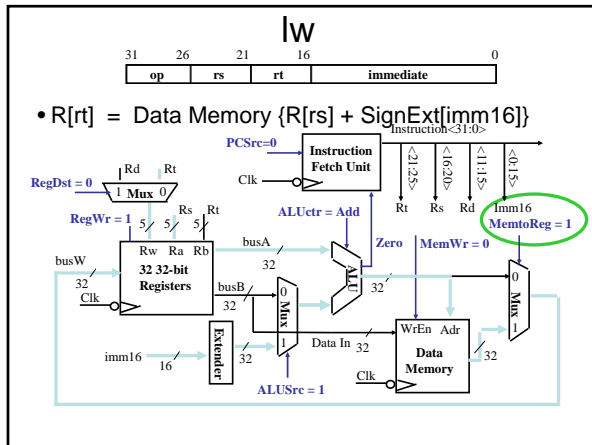
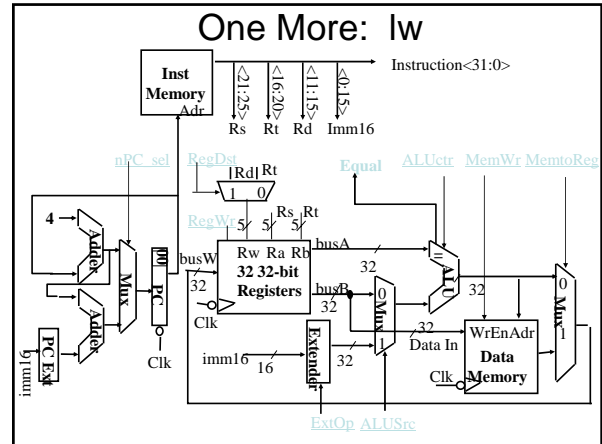
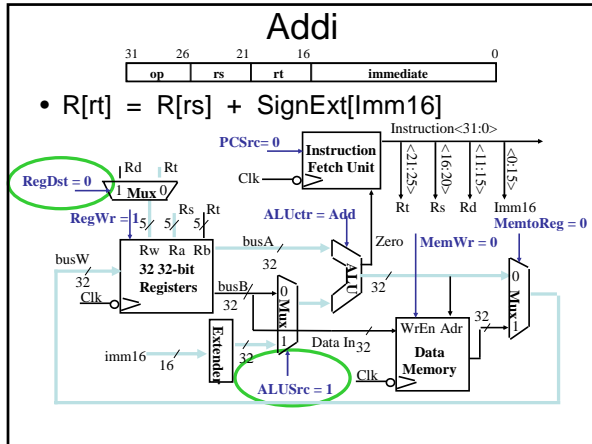


•  $R[rd] = R[rs] + R[rt]$



### How About ADDI?





### A Summary of the Control Signals (1/2)

inst	Register Transfer	Control Signals
ADD	$R[r_d] \leftarrow R[r_s] + R[r_t]$ ; $PC \leftarrow PC + 4$	$ALUSrc = \text{RegB}$ , $ALUctr = \text{"add"}$ , $\text{RegDst} = r_d$ , $\text{RegWr}$ , $nPC\_sel = \text{"+4"}$
SUB	$R[r_d] \leftarrow R[r_s] - R[r_t]$ ; $PC \leftarrow PC + 4$	$ALUSrc = \text{RegB}$ , $ALUctr = \text{"sub"}$ , $\text{RegDst} = r_d$ , $\text{RegWr}$ , $nPC\_sel = \text{"+4"}$
ORI	$R[r_t] \leftarrow R[r_s] + \text{zero\_ext}(Imm16)$ ; $PC \leftarrow PC + 4$	$ALUSrc = \text{Im}$ , $\text{Extop} = \text{"Z"}$ , $ALUctr = \text{"or"}$ , $\text{RegDst} = r_t$ , $\text{RegWr}$ , $nPC\_sel = \text{"+4"}$
LOAD	$R[r_t] \leftarrow \text{MEM}[R[r_s] + \text{sign\_ext}(Imm16)]$ ; $PC \leftarrow PC + 4$	$ALUSrc = \text{Im}$ , $\text{Extop} = \text{"Sn"}$ , $ALUctr = \text{"add"}$ , $\text{MementoReg} = r_t$ , $\text{RegWr}$ , $nPC\_sel = \text{"+4"}$
STORE	$\text{MEM}[R[r_s] + \text{sign\_ext}(Imm16)] \leftarrow R[r_s]$ ; $PC \leftarrow PC + 4$	$ALUSrc = \text{Im}$ , $\text{Extop} = \text{"Sn"}$ , $ALUctr = \text{"add"}$ , $\text{MemWr} = 1$ , $nPC\_sel = \text{"+4"}$
BEQ	if $(R[r_s] == R[r_t])$ then $PC \leftarrow PC + \text{sign\_ext}(Imm16)$    00 else $PC \leftarrow PC + 4$	$nPC\_sel = \text{"Br"}$ , $ALUctr = \text{"sub"}$

### A Summary of the Control Signals (2/2)

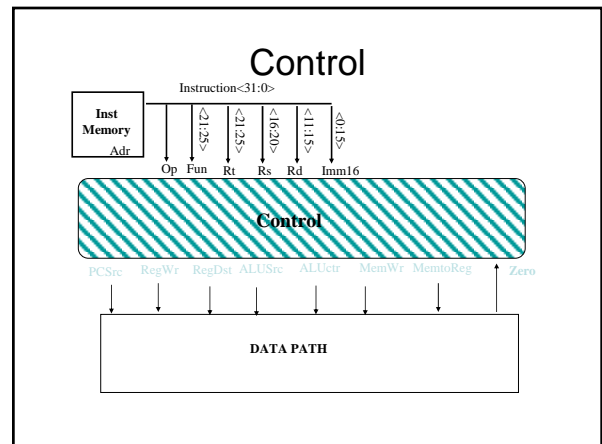
See Appendix A

func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MementoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

R-type:  $31 \text{ op} \quad 26 \text{ rs} \quad 21 \text{ rt} \quad 16 \text{ rd} \quad 11 \text{ shamt} \quad 6 \text{ funct} \quad 0$  add, sub

I-type:  $31 \text{ op} \quad 26 \text{ rs} \quad 21 \text{ rt} \quad 16 \text{ immediate} \quad 0$  ori, lw, sw, beq

J-type:  $31 \text{ op} \quad 26 \text{ target address} \quad 0$  jump



### CPU Design – Designing the Control

- ISA design
  - Instruction formats
  - opcode/funct assignment
- → Truth table
- → Logic expressions of control signals
- → Simplified logic expressions
  
- Similar instructions producing similar control signals should have similar opcode/funct code
  - Add(funct=32) / Addu(funct=33)

# BREAK!

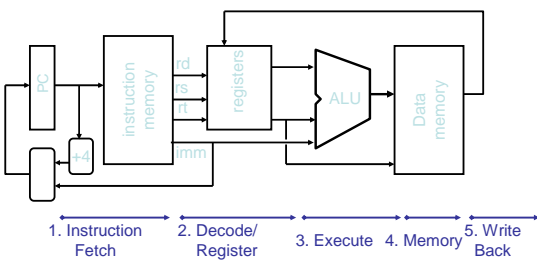
### Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- **Pipelining**
- Caches
- Virtual Memory
- I/O and Performance

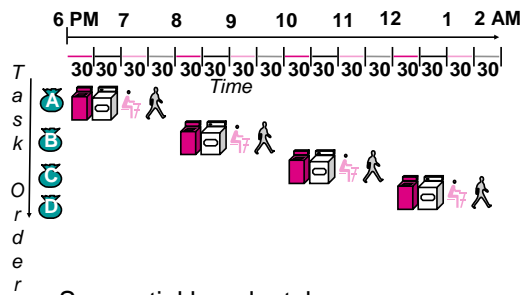
### Pipelining

- View the processing of an instruction as a sequence of potentially independent steps
- Use this separation of stages to optimize your CPU by starting to process the next instruction while still working on the previous one
- In the real world, you have to deal with some interference between instructions

### Review Datapath

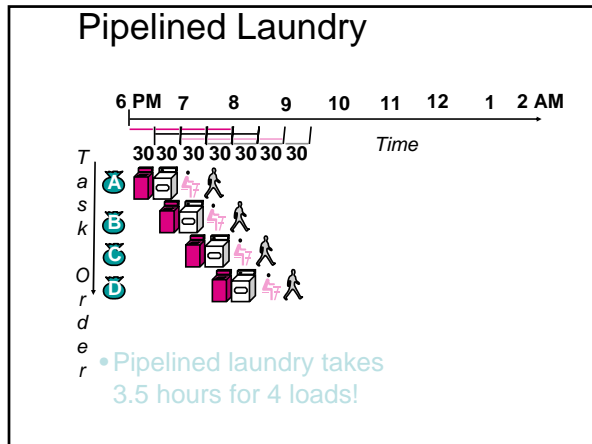


### Sequential Laundry



- Sequential laundry takes 8 hours for 4 loads





- ### Pipelining -- Key Points
- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
  - **Multiple** tasks operating simultaneously using different resources
  - Potential speedup = **Number pipe stages**
  - Time to "**fill**" pipeline and time to "**drain**" it reduces speedup

- ### Pipelining -- Limitations
- Pipeline rate limited by **slowest** pipeline stage
  - Unbalanced lengths of pipe stages also reduces speedup
  - Interference between instructions – called a hazard

- ### Pipelining -- Hazards
- **Hazards** prevent next instruction from executing during its designated clock cycle
    - **Structural hazards**: HW cannot support this combination of instructions (single person to fold and put clothes away)
    - **Control hazards**: Pipelining of branches & other instructions **stall** the pipeline until the hazard; "**bubbles**" in the pipeline
    - **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)

- ### Pipelining – Structural Hazards
- Avoid memory hazards by having two L1 caches – one for data and one for instructions
  - Avoid register conflicts by always writing in the first half of the clock cycle and reading in the second half
    - This is ok because registers are *much* faster than the critical path

- ### Pipelining – Control Hazards
- Occurs on a branch or jump instruction
  - Optimally you would always branch when needed, never execute instructions you shouldn't have, and always have a full pipeline
    - This generally isn't possible
  - Do the best we can
    - Optimize to 1 problem instruction
    - Stall
    - Branch Delay Slot

### Pipelining – Data Hazards

- Occur when one instruction is dependant on the results of an earlier instruction
- Can be solved by forwarding for all cases except a load immediately followed by a dependant instruction
  - In this case we detect the problem and stall (for lack of a better plan)

### Pipelining -- Exercise

Assuming one instruction executed per clock cycle, delayed branch, forwarding, interlock on load hazards, and a full pipeline: how many cycles will this code take? Where is there forwarding?

```
Loop: lw,   $t0, 0($s1)
      addu, $t0, $t0, $s2
      sw,   $t0, 0($s1)
      addiu, $s1, $s1, -4
      bne,  $s1, $zero, Loop
      addu  $s3, $t1, $t1
```

### Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- Pipelining
- **Caches**
- Virtual Memory
- I/O and Performance

### Caches

- The Problem: Memory is slow compared to the CPU
- The Solution: Create a fast layer in between the CPU and memory that holds a subset of what is stored in memory
- We call this creature a Cache

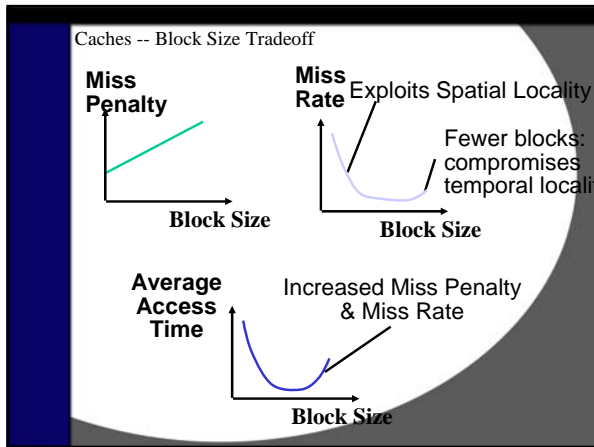
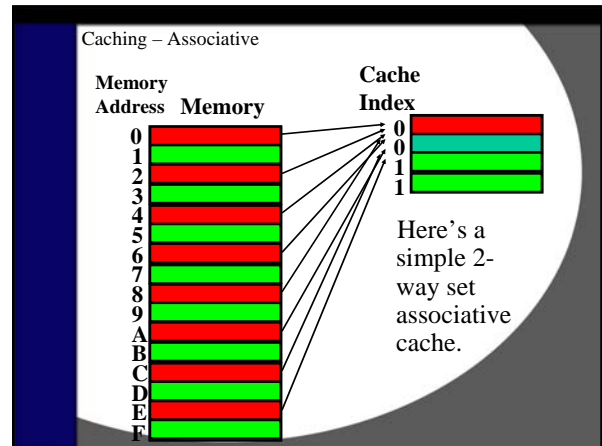
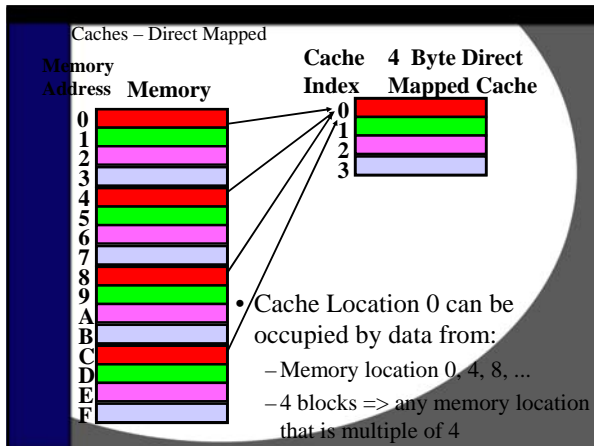
### Caches – Reference Patterns

- Holding an arbitrary subset of memory in a faster layer should not provide any performance increase
  - Therefore we must carefully choose what to put there
- Temporal Locality: When a piece of data is referenced once it is likely to be referenced again soon
- Spatial Locality: When a piece of data is referenced it is likely that the data near it in the address space will be referenced soon

### Caches – Format & Mapping

Tag	Index	Offset
(msb)tttttttttttttttt	iiiiiiiiiii	ooooo(lsb)

- **Tag:** Unique identifier for each block in memory that maps to same index in cache
- **Index:** Which "row" in the cache the data block will map to (for direct mapped cache each row is a single cache block)
- **Block Offset:** Byte offset within the block for the particular word or byte you want to access



Caches – Exercise 1

How many bits would be required to implement the following cache?

Size: 1MB  
 Associativity: 8-way set associative  
 Write policy: Write back  
 Block size: 32 bytes  
 Replacement policy: Clock LRU (requires one bit per data block)

Caches – Solution 1

Number of blocks = 1MB / 32 bytes = 32 Kblocks ( $2^{15}$ )

Number of sets = 32Kblocks / 8 ways = 4 Ksets ( $2^{12}$ )  
 → 12 bits of index

Bits per set =  $8 * (8 * 32 + (32 - 12 - 5) + 1 + 1 + 1)$   
 → 32bytes + tag bits + valid + LRU + dirty

Bits total = (Bits per set) \* (# Sets) =  $2^{12} * 2192 = 8,978,432$  bits

Caches – Exercise 2

Given the following cache and access pattern, classify each access as hit, compulsory miss, conflict miss, or capacity miss:

Cache:  
 Word addressed  
 2 words/block  
 8 blocks  
 2-way set associative  
 LRU replacement

Access Pattern (word addresses):  
 3, 10, 15, 0, 5, 1, 9, 4, 10, 16, 0, 3

#### Caches – Solution 2

3 (**comp, s1, w0**), 10 (**comp, s1, w1**),  
15 (**comp, s3, w0**), 0 (**comp, s0, w0**),  
5 (**comp, s2, w0**), 1 (**hit**),  
9 (**comp, s0, w1**), 4 (**hit**),  
10 (**hit**), 16 (**conf, s0, w0**),  
0 (**conf, s0, w1**), 3 (**hit**)

#### Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- Pipelining
- Caches
- **Virtual Memory**
- I/O and Performance

#### Virtual Memory

- Caching works well – why not extend the basic concept to another level?
- We can make the CPU think it has a much larger memory than it actually does by “swapping” things in and out to disk
- While we’re doing this, we might as well go ahead and separate the physical address space and the virtual address space for protection and isolation

#### VM – Virtual Address

- Address space broken into fixed-sized pages
- Two fields in a virtual address
  - VPN
  - Offset
- Size of offset =  $\log_2(\text{size of page})$

#### VM – Address Translation

- VPN used to index into page table and get Page Table Entry (PTE)
- PTE is located by indexing off of the Page Table Base Register, which is changed on context switches
- PTE contains valid bit, Physical Page Number (PPN), access rights

#### VM – Translation Look-Aside Buffer (TLB)

- VM provides a lot of nice features, but requires several memory accesses for its indirection – this really kills performance
- The solution? Another level of indirection: the TLB
- Very small fully associative cache containing the most recently used mappings from VPN to PPN

### VM – Exercise

Given a processor with the following parameters, how many bytes would be required to hold the entire page table in memory?

- Addresses: 32-bits
- Page Size: 4KB
- Access modes: RO, RW
- Write policy: Write back
- Replacement: Clock LRU (needs 1-bit)

### VM – Solution

Number of bits page offset = 12 bits

Number of bits VPN/PPN = 20 bits

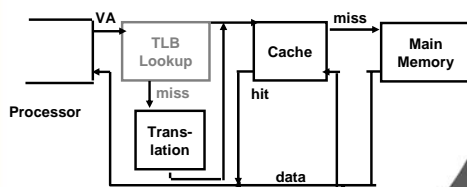
Number of pages in address space  
= 232 bytes/212 bytes = 220 = 1Mpages

Size of PTE = 20 + 1 + 1 + 1 = 23 bits

Size of PT = 220 \* 23 bits = 24,117,248 bits  
= **3,014,656 bytes**

### The Big Picture

CPU – TLB – Cache – Memory – VM



### Big Picture – Exercise

What happens in the following cases?

- TLB Miss
- Page Table Entry Valid Bit 0
- Page Table Entry Valid Bit 1
- TLB Hit
- Cache Miss
- Cache Hit

### Big Picture – Solution

- TLB Miss – Go to the page table to fill in the TLB. Retry instruction.
- Page Table Entry Valid Bit 0 / Miss – Page not in memory. Fetch from backing store. (Page Fault)
- Page Table Entry Valid Bit 1 / Hit – Page in memory. Fill in TLB. Retry instruction.
- TLB Hit – Use PPN to check cache.
- Cache Miss – Use PPN to retrieve data from main memory. Fill in cache. Retry instruction.
- Cache Hit – Data successfully retrieved.

• **Important thing to note: Data is *always* retrieved from the cache.**

### Topics Since Midterm

- Digital Logic
  - Verilog
  - State Machines
- CPU Design
- Pipelining
- Caches
- Virtual Memory
- **I/O and Performance**

### IO and Performance

- IO Devices
- Polling
- Interrupts
- Networks

### I/O Device Examples and Speeds

- I/O Speed: bytes transferred per second (from mouse to Gigabit LAN: 100-million-to-1)

Device	Behavior	Partner	Data Rate (KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	1,000,000.00

**When discussing transfer rates, use 10<sup>x</sup>**

### IO – Problems Created by Device Speeds

- CPU runs far faster than even the fastest of IO devices
- CPU runs many orders of magnitude faster than the slowest of currently used IO devices
- Solved by adhering to well defined conventions
  - Control Registers

### IO – Device Communication

- Some processors handle IO via special instructions. This is called Programmed IO
- MIPS (and many other platforms) use variations on Memory Mapped IO where writing to or reading from certain portions of the address space actual communicates with the IO device.
- SPIM fakes this communication by using 4 special device registers

### SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
- Read from keyboard (**receiver**); 2 device regs
- Writes to terminal (**transmitter**); 2 device regs

Receiver Control 0xffff0000	Unused (00...00)	(IE)	Ready
Receiver Data 0xffff0004	Unused (00...00)	Received Byte	
Transmitter Control 0xffff0008	Unused (00...00)	(IE)	Ready
Transmitter Data 0xffff000c	Unused	Transmitted Byte	

Cal CS51C L39 IO (R3) Garcia, Fall 2004 © UCB

### IO – Polling

- CPU continuously checks the device's control registers to see if it needs to take any action
- Extremely easy to program
- Extremely inefficient. The CPU spends potentially huge amounts of times polling IO devices.

## IO – Interrupts

- Asynchronous notification to the CPU that there is something to be dealt with for an IO device
- Not associated with any particular instruction – this implies that the interrupt must carry some information with it
- Causes the processor to stop what it was doing and execute some other code

## IO – Types of Interrupts

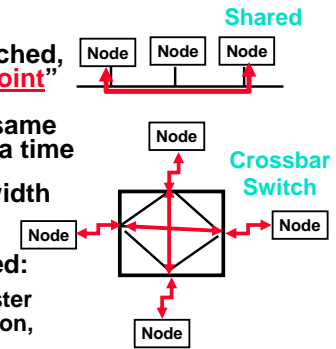
- **Exception:** A signal marking that something strange has happened. This is usually the direct result of an instruction (i.e. an overflow exception)
- **Interrupt:** An asynchronous signal that some wants the attention of the CPU, interrupts the CPU in the middle of what it is doing (i.e. a device interrupt)
- **Trap:** A synchronous exception that is explicitly called for by the programmer to force the processor to do something (i.e. a kernel trap)

## IO – Interrupts for Device IO

- Device raises a flag in the processor – usually called an interrupt line
  - When this line is asserted the processor jumps to a specific location
- This location is an entry point to the Interrupt Service Routine (ISR)
- ISR uses the information that is available about which device raised the interrupt flag to jump to the proper device routine
- Device routine deals with whatever caused the IO Device to generate the interrupt

## Shared vs. Switched Based Networks

- **Shared Media vs. Switched:** in switched, pairs (“point-to-point” connections) communicate at same time; shared 1 at a time
- **Aggregate bandwidth (BW) in switched network is many times shared:**
  - point-to-point faster since no arbitration, simpler interface



CS51C L39 IO - Networks (88)

Garcia, Fall 2004 © UCB

## Networks – Props To Kansas City: Sprint Headquarters



## Networks – Protocols

- A protocol establishes a logical format and API for communication
- Actual work is done by a layer beneath the protocol, so as to protect the abstraction
- Allows for encapsulation – carry higher level information within lower level “envelope”
- Fragmentation – packets can be broken in to smaller units and later reassembled

Networks – TCP in 20 seconds

- TCP guarantees in-order delivery of complete packets
- Accomplishes this by keeping a lot of extra data around in case it needs to be resent
- Sender holds on to old data and retries the transmission if it does not get an ACK back from receiver within a time limit
- TCP Packets look like this: (kinda)

**Checksum**

Net ID	Net ID	Len	ACK INFO	CMD/ Address /Data	
Header				Payload	Trailer

Networks – Alright, I Know You're Curious...

### TCP Packet Format

The diagram shows a 32-bit TCP packet structure. The fields are: source port number (16 bits), destination port number (16 bits), sequence number (32 bits), acknowledgement number (32 bits), head len (4 bits), unused (4 bits), window size (16 bits), Internet checksum (16 bits), ptr to urgent data (16 bits), options (variable length), and data (variable length).

Networks – Complications

- Packet headers eat in to your total bandwidth
- Software overhead for transmission limits your effective bandwidth significantly

Networks – Exercise

What percentage of your total bandwidth is being used for protocol overhead in this example:

- Application sends 1MB of true data
- TCP has a segment size of 64KB and adds a 20B header to each packet
- IP adds a 20B header to each packet
- Ethernet breaks data into 1500B packets and adds 24B worth of header and trailer

Networks – Solution

1MB / 64K = 16 TCP Packets  
 16 TCP Packets = 16 IP Packets  
 64K/1500B = 44 Ethernet packets per TCP Packet  
 16 TCP Packets \* 44 = 704 Ethernet packets  
 20B overhead per TCP packet + 20B overhead per IP packet + 24B overhead per Ethernet packet =  
 20B \* 16 + 20B \* 16 + 24B \* 704 = 17,536B of overhead  
 We send a total of 1,066,112B of data. Of that, 1.64% is protocol overhead.

### Magnetic Disks

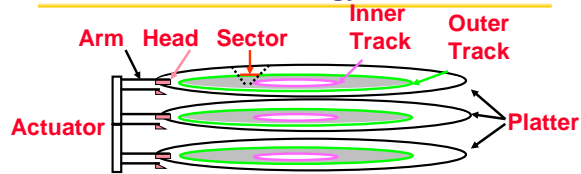
The diagram shows a computer system with the following components: Processor (active), Memory (passive), Devices (Input, Output), Keyboard, Mouse, Disk, Network, Display, and Printer.

- **Purpose:**
  - Long-term, nonvolatile, inexpensive storage for files
  - Large, inexpensive, slow level in the memory hierarchy (discuss later)

Cal CS81C L49 (D) Disks (90) Garcia, Fall 2004 © UCB



## Disk Device Terminology



- Several **platters**, with information recorded magnetically on both **surfaces** (usually)
- Bits recorded in **tracks**, which in turn divided into **sectors** (e.g., 512 Bytes)
- **Actuator** moves **head** (end of **arm**) over track ("**seek**"), wait for **sector** rotate under **head**, then read or write



CS61C L40 10: Disks (87)

Garcia, Fall 2004 © UCB

## Disk Performance Model /Trends

- **Capacity** : + 100% / year (2X / 1.0 yrs)  
Over time, grown so fast that # of platters has reduced (some even use only 1 now!)
- **Transfer rate (BW)** : + 40%/yr (2X / 2 yrs)
- **Rotation+Seek time** : – 8%/yr (1/2 in 10 yrs)
- **Areal Density**
  - Bits recorded along a track: **Bits/Inch (BPI)**
  - # of tracks per surface: **Tracks/Inch (TPI)**
  - We care about **bit density per unit area** **Bits/Inch<sup>2</sup>**
  - Called **Areal Density** = BPI x TPI
- **MB/\$**: > 100%/year (2X / 1.0 yrs)
  - Fewer chips + areal density



CS61C L40 10: Disks (88)

Garcia, Fall 2004 © UCB

## Disks – RAID

- Idea was to use small, relatively inexpensive disks in place of large, very expensive disks to reduce cost and increase Mean Time to Failure
- Some RAID models are more successful than others
- We are not going to go over them today, since we just learned this last week.
- That being said, *make sure you know them!*

## Performance -- Metrics

- Best overall raw computational power
- Least Cost
- Best power / Cost
- Response Time?
- Throughput?

Benchmarks help quantify these ideas.

## That Was A LOT of Slides!

- Stick around if you have any more questions
- See you at the final, this Tuesday 12/14 from 12:30 to 3:30 at 230 Hearst Gym (and if that isn't confusing, I don't know what is)
- Don't forget to review stuff from before the midterm!
- **STUDY!!!**