Memory Mapped I/O

Certain memory addresses correspond to registers in I/O devices and not normal memory.

Control Register:	Indicates if it is okay to read/write data register
Data Register:	Contains I/O data

Register	**Location**	**Contains**
Receiver Control	0xffff0000	Lowest two bits: Interrupt Enable Bit, Ready Bit
Receiver Data	0xffff0004	Received data stored at lowest byte
Transmitter Control	0xffff0008	Lowest two bits: Interrupt Enable Bit, Ready Bit
Transmitter Data	0xffff000c	Transmitted data stored at lowest byte

Describe MIPS code to read a byte from the receiver and immediately send it to the transmitter.

We would load a byte from the receiver data address and store it into the transmitter data address.

```
lui $t0 0xffff
lb $t1 4($t0)
sb $t112($t0)
```

Polling and Interrupts

Operation	**Definition**	**Pro/Con**	**Good For**
Polling	Periodically check to see if the device is ready to transfer data.	+Predictable performance +Easy to implement -Wastes time repeatedly checking if ready	Applications that require infrequent polling, low data rate.
Interrupts	Device makes an asynchronous request for data transfer.	+Allows data transfer on demand -Overhead of saving state before invoking interrupt service routine. -If interrupts might occur, performance is less predictable.	High data rate I/O that would otherwise be too expensive to poll. Would use lots of CPU when active, but none when inactive.

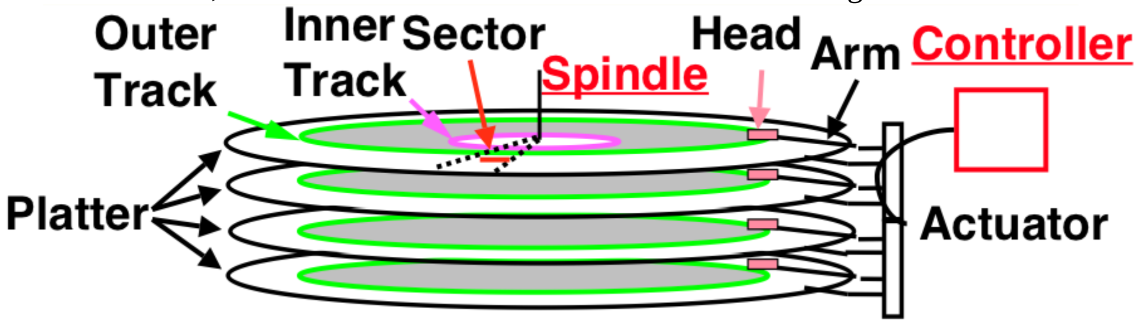
See end of sp08 “I/O Basics” lecture for more details.

Disk Organization

Magnetic disks are one of the most common types of I/O devices. Bits are encoded by the controlling the polarity of magnetic fields on some sort of substrate. Since the magnetic fields do not require power

Based on notes by Aaron Staley, which were in turn based on notes by David Jacobs.

to be maintained, disks are considered a form of non-volatile storage.



An additional term not shown here is that the collection of corresponding tracks across all the platters is called a cylinder.
There are two ways to address disks. Logical addressing treats the disk drive as one big array of blocks. Physical addressing uses a (cylinder, sector, platter) tuple to specify a blocks physical position in the disk drive.

Disk Performance
Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead

RAID
Big disks are expensive (and dangerous). We can use an array of smaller disks to simulate the behavior of one larger disk with a more reasonable cost.

RAID 0	No redundancy, just multiple disks
RAID 1	Mirroring for redundancy, doubles read bandwidth
RAID 2	Bit-level striping, increases bandwidth further
RAID 3	Parity Disks, allows recovery from a single disk failure
RAID 4	Block-level striping with Parity Disk, increases bandwidth
RAID 5+	Striped Parity, reduces wear and tear

Disk Exercises

We have a 7200 RPM drive with 3 ms seek time and a 20 MB/sec transfer rate once the head is in place. Assume that the controller overhead is negligible.

1) What is the overall throughput reading 1 MeBi of contiguous data on a random track?

120 rotations per second = 8.33 ms / rotation. On average, we will have to wait half a rotation for the data to rotate under the head => 4.16 ms average rotation time.

20 MB/sec transfer rate => 50 ms to transfer 1 Mebi

Throughput = data/sec = 1 Mebi / (50 + 3 + 4.16) = 17.49 Mebi/sec

2) What is the overall throughput reading 1MeBi of data spread randomly across the drive as 8 KiBi

Based on notes by Aaron Staley, which were in turn based on notes by David Jacobs.

files? (this is why disk fragmentation is bad)
50 ms spent in transfer time, as before.

2^7 files, 3 ms seek time + 4.16 ms rotation time per file.

Throughput = 1 Mebi / (50 + 2^7*(7.16)) = 1.03 Mebi / sec

Performance Metrics

In order to get any meaningful definition of performance, we need to develop a quantitative metric that we all can agree on. This is harder than it sounds. We briefly talked about these when discussing pipelining.

Response Time, Execution Time, Latency – the time it takes to complete one task
Throughput, Bandwidth – tasks completed per time unit

Megahertz Myth

A processor's performance is determined by more than just the clock speed.

CPU time = Instruction Count * CPI * clock period

Exercises

You are the lead developer of a new video game at AE, Inc. The graphics are quite sexy, but the frame rates (performance) are horrible. Doubly unfortunately, you have to show it off at a shareholder meeting tomorrow. What do you do?

You need to render your latest and greatest über-l33t animation. If your rendering software contains the following mix of instructions, which processor is the best choice?

Operation	Frequency
ALU	30%
Load	30%
Store	20%
Branch	20%

A's CPI	B's CPI	C's CPI
1	1	1
3	5	3
2	3	4
3	2	2

Average CPI:

A: 1*.3 + 3*.3 + 2*.2 + 3*.2 = 2.2
B: 2.8
C: 2.4

A wins.

What if the processors had different clock speeds? Assume A is a 1 Ghz processor, B is a 1.5 Ghz processor, and C is a 750 Mhz processor.

1/frequency = seconds/cycle
cycles/inst * seconds/cycle = seconds/inst, a better estimate of performance.

seconds / cycle: A = 1 ns, B = .66 ns, C = 1.33 ns
seconds / inst: A = 2.2 ns, B = 1.86 ns, C = 3.2 ns.

B wins.
But wait, these processors are made by different manufacturers, and use different instruction sets. So the renderer (for the different architectures) takes a different number of instructions on each. Which is best if your main loop on A averages 1000 instructions; on B it averages 800 instructions; and on C it averages 1200 instructions?
We now have instructions/program.
seconds/inst [from part b] * inst/program = seconds/program, the runtime.

instructions / program: A = 1000, B = 800, C = 1200
seconds/program: A = 2.2 us, B = 1.493 us, C = 3.84 us.
B produces the fastest program, and wins.

Parallel Computing

Parallel computing refers more to multicore and multiprocessor machines. This is sometimes also called "supercomputing." Since the processors are physically closer together, there is a potential for much faster communications between them. However, synchronizing the processors can prove a difficult problem.

Amdahl's Law

The potential speedup from parallelization is limited by the amount a program can be parallelized. Let s be the fraction of the work that must be done sequentially and P be the number of processors. Then,

$$\text{Speedup}(P) \leq 1/s$$

Exercise

What are the contributing factors to Amdahl's law? Why isn't it an equality?

This would be an equality if the runtime of the parallel portion were reduced to 0, ie, infinite parallelization with no overhead. Neither of these conditions hold true in practice, so the speedup is strictly less than this ideal value, how much so determined by the degree of parallelization and the (parallelization) overhead.