Solutions for Chapter 8 Exercises

8.1 Each transaction requires 10,000 $\times$ 50 = 50,000 instructions.

CPU limit: 500M/50K = 10,000 transactions/second.

The I/O limit for A is 1500/5 = 300 transactions/second.

The I/O limit for B is 1000/5 = 200 transactions/second.

These I/O limits limit the machine.

8.2 System A

transactions	9	compute	1	
I/Os	45	latency	5	
times	900 ms	100 us	100 ms	exceeds 1 s

Thus system A can only support 9 transactions per second.

System B—first 500 I/Os (first 100 transactions)

transactions	9	compute	1	1	
I/Os	45	latency	5	5	
times	810 ms	100 us	90 ms	90 ms	990.1 ms

Thus system B can support 11 transactions per second at first.

8.3 Time/file = 10 seconds + 40 MB * 1/(5/8) seconds/MB = 74 seconds

Power/file = 10 seconds * 35 watts + (74 – 10) seconds * 40 watts = 2910 J

Number of complete files transferred = 100,000 J/2910 J = 34 files

8.4 Time/file = 10 seconds + 0.02 seconds + 40 MB * 1/(5/8) seconds/MB = 74.02 seconds

Hard disk spin time/file = 0.02 seconds + 40 MB * 1/50 seconds/MB = 0.82 seconds

Power/file = 10 seconds * 33 watts + 0.02 seconds * 38 watts + 0.8 seconds * 43 watts + 63.2 seconds * 38 watts = 330 J + 0.76 J + 34.4 J + 2401.6 J = 2766.76 J

Number of complete files transferred = 100000 J / 2766.76 J = 36 files

Energy for all 100 files = 2766.76 * 100 = 276676 J

8.5 After reading sector 7, a seek is necessary to get to the track with sector 8 on it. This will take some time (on the order of a millisecond, typically), during which the disk will continue to revolve under the head assembly. Thus, in the version where sector 8 is in the same angular position as sector 0, sector 8 will have already revolved past the head by the time the seek is completed and some large fraction of an additional revolution time will be needed to wait for it to come back again. By skewing the sectors so that sector 8 starts later on the second track, the seek will have time to complete, and then the sector will soon thereafter appear under the head without the additional revolution.

8.6 No solution provided.

8.7

a. Number of heads = 15
b. Number of platters = 8
c. Rotational latency = 8.33 ms
d. Head switch time = 1.4 ms
e. Cylinder switch time = 2.1 ms

8.8

a. System A requires 10 + 10 = 20 terabytes.

   System B requires 10 + 10 * 1/4 = 12.5 terabytes.

   Additional storage: 20 – 12.5 = 7.5 terabytes.
b. System A: 2 blocks written = 60 ms.

   System B: 2 blocks read and written = 120 ms.
c. Yes. System A can potentially accommodate more failures since it has more redundant disks. System A has 20 data disks and 20 check disks. System B has 20 data disks and 5 check disks. However, two failures in the *same* group will cause a loss of data in both systems.

8.9 The power failure could result in a parity mismatch between the data and check blocks. This could be prevented if the writes to the two blocks are performed simultaneously.

8.10 20 meters time: 20 m * 1/(1.5 * 10^8) s/m = 133.3 ns

2,000,000 meters time: 2000000 m * 1/(1.5 * 10^8) s/m = 13.3 ms

8.11 20 m: 133.3 * 10^{-9} s * 6 MB/sec = 0.8 bytes

2000000 m: 13.3 * 10^{-3} s * 6 MB/sec = 80 KB

8.12 4 KHz * 2 bytes/sample * 100 conversations = 800,000 bytes/sec

Transmission time is 1 KB/5 MB/sec + 150 µs = 0.00035 seconds/KB

Total time/KB = 800 * 0.00035 = 0.28 seconds for 1 second of monitoring

There should be sufficient bandwidth.

8.13

a. 0
b. 1
c. 1
d. 2
e. Each bit in a 3-bit sequence would have to be reversed. The percentage of errors is 0.01 * 0.01 * 0.01 = 0.000001 (or 0.0001%)

8.14

a. 1
b. 0

8.15

a. Not necessarily, there could be a single-bit error or a triple-bit error.
b. No. Parity only specifies whether an error is present, not which bit the error is in.
c. No. There could be a double-bit error or the word could be correct.

8.16 (Seek time + Rotational delay + Overhead) * 2 + Processing time

(0.008 sec + 0.5 / (10000/60) sec + 0.002) * 2 + (20 million cycles)(5 GHz) sec = (.008 + .003 + .002)*2 + .004 = 30 ms

Block processed/second = 1/30 ms = 33.3

Transfer time is 80 µsec and thus is negligible.

8.17 Possible answers may include the following:

- Application programmers need not understand how things work in lower levels.
- Abstraction prevents users from making low-level errors.
- Flexibility: modifications can be made to layers of the protocol without disrupting other layers.

8.18 For 4-word block transfers, the bus bandwidth was 71.11 MB/sec. For 16-word block transfers, the bus bandwidth was 224.56 MB/sec. The disk drive has a transfer rate of 50 MB/sec. Thus for 4-word blocks we could sustain 71/50 = 1 simultaneous disk transfers, and for 16-word blocks we could sustain 224/50 = 4 simultaneous disk transfers. The number of simultaneous disk transfers is inherently an integer and we want the sustainable value. Thus, we take the floor of the quotient of bus bandwidth divided by disk transfer rate.

8.19 For the 4-word block transfers, each block now takes

1. 1 cycle to send an address to memory
2. 150 ns/5 ns = 30 cycles to read memory
3. 2 cycles to send the data
4. 2 idle cycles between transfers

This is a total of 35 cycles, so the total transfer takes $35 \times 64 = 2240$ cycles. Modifying the calculations in the example, we have a latency of 11,200 ns, 5.71M transactions/second, and a bus bandwidth of 91.43 MB/sec.

For the 16-word block transfers, each block now takes

1. 1 cycle to send an address to memory
2. 150 ns or 30 cycles to read memory
3. 2 cycles to send the data
4. 4 idle cycles between transfers, during which the read of the next block is completed

Each of the next two remaining 4-word blocks requires repeating the last two steps. The last 4-word block needs only 2 idle cycles before the next bus transfer. This is a total of $1 + 20 + 3 * (2 + 4) + (2 + 2) = 53$ cycles, so the transfer takes $53 * 16 = 848$ cycles. We now have a latency of 4240 ns, 3.77M transactions/second, and a bus bandwidth of 241.5 MB/sec.

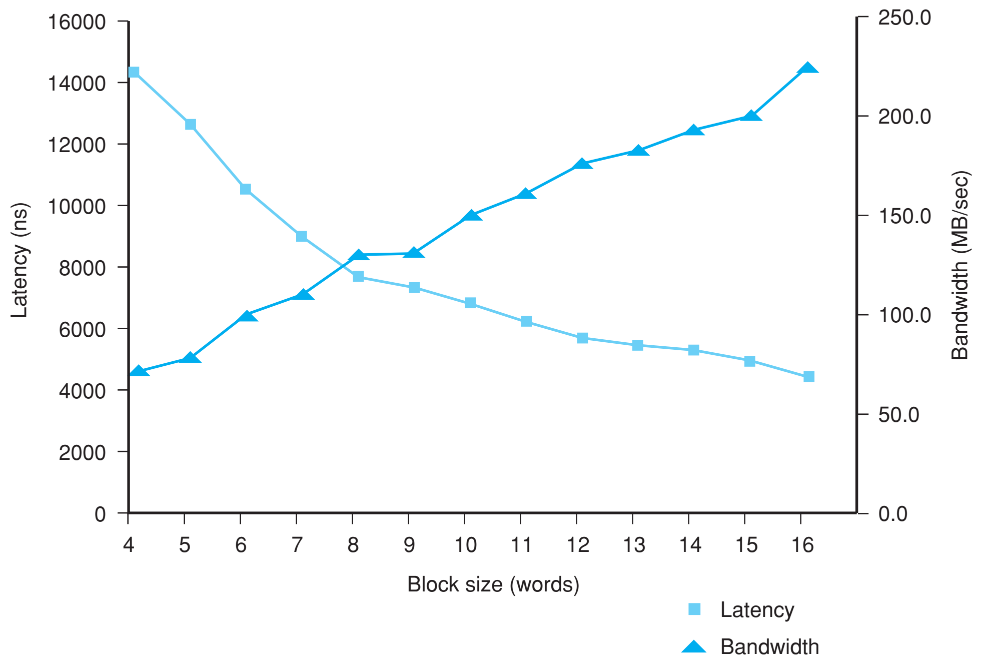
Note that the bandwidth for the larger block size is only 2.64 times higher given the new read times. This is because the 30 ns for subsequent reads results in fewer opportunities for overlap, and the larger block size performs (relatively) worse in this situation.

8.20 The key advantage would be that a single transaction takes only 45 cycles, as compared with 57 cycles for the larger block size. If because of poor locality we were not able to make use of the extra data brought in, it might make sense to go with a smaller block size. Said again, the example assumes we want to access 256 words of data, and clearly larger block sizes will be better. (If it could support it, we'd like to do a single 256-word transaction!)

8.21 Assume that only the 4-word reads described in the example are provided by the memory system, even if fewer than 4 words remain when transferring a block. Then,

	Block size (words)												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Number of 4-word transfers to send the block	1	2	2	2	2	3	3	3	3	4	4	4	4
Time to send address to memory (bus cycles)	1	1	1	1	1	1	1	1	1	1	1	1	1
Time to read first 4 words in memory (bus cycles)	40	40	40	40	40	40	40	40	40	40	40	40	40
Block transfer time, at 2 transfer bus cycles and 2 idle bus cycles per 4-word transfer (bus cycles)	4	8	8	8	8	12	12	12	12	16	16	16	16
Total time to transfer one block (bus cycles)	45	49	49	49	49	53	53	53	53	57	57	57	57
Number of bus transactions to read 256 words using the given block size	64	52	43	37	32	29	26	24	22	20	19	18	16
Time for 256-word transfer (bus cycles)	2880	2548	2107	1813	1568	1537	1378	1272	1166	1140	1083	1026	912
Latency (ns)	14400	12740	10535	9065	7840	7685	6890	6360	5830	5700	5415	5130	4560
Number of bus transactions (millions per second)	4.444	4.082	4.082	4.082	4.082	3.774	3.774	3.774	3.774	3.509	3.509	3.509	3.509
Bandwidth (MB/sec)	71.1	80.4	97.2	113.0	130.6	133.2	148.6	161.0	175.6	179.6	189.1	199.6	224.6

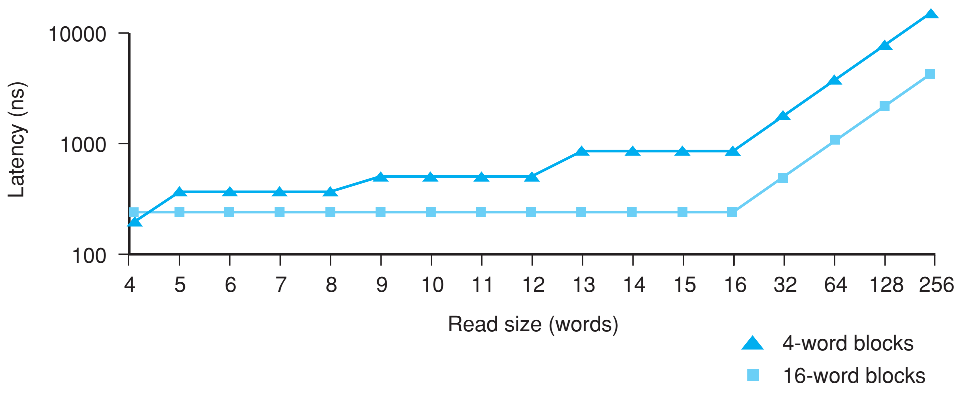
The following graph plots latency and bandwidth versus block size:



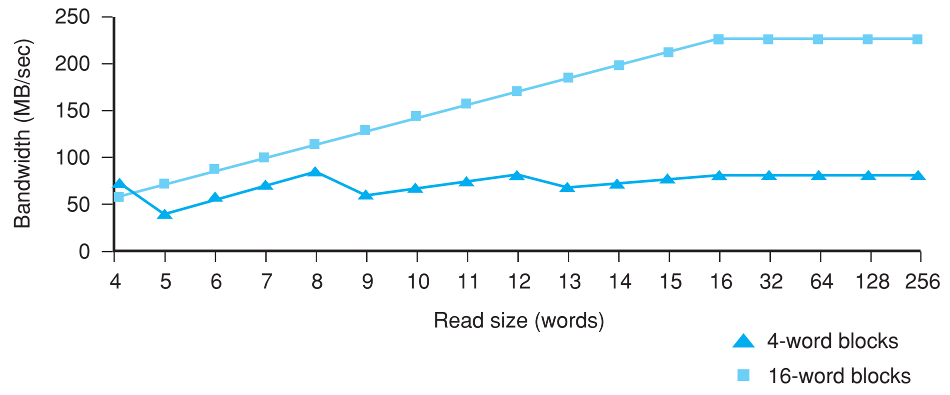
8.22 From the example, a 4-word transfer takes 45 bus cycles and a 16-word block transfer takes 57 bus cycles. Then,

	Read size (words)																
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**32**	**64**	**128**	**256**
Number of 4-word transfers to send the data	1	2	2	2	2	3	3	3	3	4	4	4	4	8	16	32	64
Number of 16-word transfers to send the data	1	1	1	1	1	1	1	1	1	1	1	1	1	2	4	8	16
Total read time using 4-word blocks (bus cycles)	45	90	90	90	90	135	135	135	135	180	180	180	180	360	720	1440	2880
Total read time using 16-word blocks (bus cycles)	57	57	57	57	57	57	57	57	57	57	57	57	57	114	228	456	912
Latency using 4-word blocks (ns)	225	450	450	450	450	675	675	675	675	900	900	900	900	1800	3600	7200	14400
Latency using 16-word blocks (ns)	285	285	285	285	285	285	285	285	285	285	285	285	285	570	1140	2280	4560
Bandwidth using 4-word blocks (MB/sec)	71.1	44.4	53.3	62.2	71.1	53.3	59.3	65.2	71.1	57.8	62.2	66.7	71.1	71.1	71.1	71.1	71.1
Bandwidth using 16-word blocks (MB/sec)	56.1	70.2	84.2	98.2	112.3	126.3	140.4	154.4	168.4	182.5	196.5	210.5	224.6	224.6	224.6	224.6	224.6

The following graph plots read latency with 4-word and 16-word blocks:



The following graph plots bandwidth with 4-word and 16-word blocks:



8.23

For 4-word blocks:

Send address and first word simultaneously =	1 clock
Time until first write occur =	40 clocks
Time to send remaining 3 words over 32-bit bus =	3 clocks
Required bus idle time =	2 clocks
Total time =	46 clocks

Latency = 64 4-word blocks at 46 cycles per block = 2944 clocks = 14720 ns
Bandwidth = $(256 \times 4$ bytes$)/14720$ ns = 69.57 MB/sec

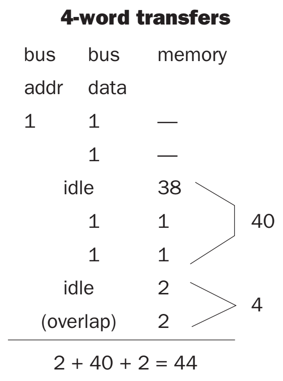
For 8-word blocks:

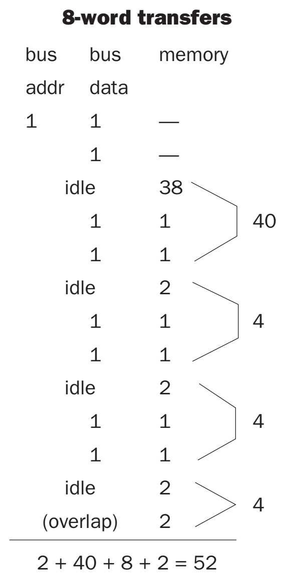
Send address and first word simultaneously =	1 clock
Time until first write occurs =	40 clocks
Time to send remaining 7 words over 32-bit bus =	7 clocks
Required bus idle time (two idle periods) =	4 clocks
Total time =	52 clocks

Latency = 32 8-word blocks at 52 cycles per block = 1664 clocks = 8320 ns

Bandwidth = (256 × 4 bytes)/8320 ns = 123.08 MB/sec

In neither case does the 32-bit address/32-bit data bus outperform the 64-bit combined bus design. For smaller blocks, there could be an advantage if the overhead of a fixed 4-word block bus cycle could be avoided.





8.24 For a 16-word read from memory, there will be four sends from the 4-word-wide memory over the 4-word-wide bus. Transactions involving more than one send over the bus to satisfy one request are typically called *burst transactions.*

For burst transactions, some way must be provided to count the number of sends so that the end of the burst will be known to all on the bus. We don't want another device trying to access memory in a way that interferes with an ongoing burst transfer. The common way to do this is to have an additional bus control signal, called BurstReq or Burst Request, that is asserted for the duration of the burst.

This signal is unlike the ReadReq signal of Figure 8.10, which is asserted only long enough to start a single transfer. One of the devices can incorporate the counter necessary to track when BurstReq should be deasserted, but both devices party to the burst transfer must be designed to handle the specific burst (4 words, 8 words, or other amount) desired. For our bus, if BurstReq is not asserted when ReadReq signals the start of a transaction, then the hardware will know that a single send from memory is to be done.

So the solution for the 16-word transfer is as follows: The steps in the protocol begin immediately after the device signals a burst transfer request to the memory by raising ReadReq and Burst_Request and putting the address on the Date lines.

1. When memory sees the ReadReq and BurstReq lines, it reads the address of the start of the 16-word block and raises Ack to indicate it has been seen.
2. I/O device sees the Ack line high and releases the ReadReq and Data lines, but it keeps BurstReq raised.
3. Memory sees that ReadReq is low and drops the Ack line to acknowledge the ReadReq signal.
4. This step starts when BurstReq is high, the Ack line is low, and the memory has the next 4 data words ready. Memory places the next 4 data words in answer to the read request on the Data lines and raises DataRdy.
5. The I/O device sees DataRdy, reads the data from the bus, and signals that it has the data by raising Ack.
6. The memory sees the Ack signal, drops DataRdy, and releases the Data lines.
7. After the I/O device sees DataRdy go low, it drops the Ack line but continues to assert BurstReq if more data remains to be sent to signal that it is ready for the next 4 words. Step 4 will be next if BurstReq is high.
8. If the last 4 words of the 16-word block have been sent, the I/O device drops BurstReq, which indicates that the burst transmission is complete.

With handshakes taking 20 ns and memory access taking 60 ns, a burst transfer will be of the following durations:

Step 1 20 ns (memory receives the address at the end of this step; data goes on the bus at the beginning of step 5)

Steps 2, 3, 4 Maximum (3×20 ns, 60 ns) = 60 ns

Steps 5, 6, 7, 4 Maximum (4 × 20 ns, 60 ns) = 80 ns (looping to read and then send the next 4 words; memory read latency completely hidden by handshaking time)

Steps 5, 6, 7, 4 Maximum (4 × 20 ns, 60 ns) = 80 ns (looping to read and then send the next 4 words; memory read latency completely hidden by handshaking time)

Steps 5, 6, 7, 4 Maximum (4 × 20 ns, 60 ns) = 80 ns (looping to read and then send the next four words; memory read latency completely hidden by handshaking time)

End of burst transfer

Thus, the total time to perform the transfer is 320 ns, and the maximum bandwidth is

(16 words × 4 bytes)/320 ns = 200 MB/sec

It is a bit difficult to compare this result to that in the example on page 665 because the example uses memory with a 200 ns access instead of 60 ns. If the slower memory were used with the asynchronous bus, then the total time for the burst transfer would increase to 820 ns, and the bandwidth would be

(16 words × 4 bytes)/820 ns = 78 MB/sec

The synchronous bus in the example on page 665 needs 57 bus cycles at 5 ns per cycle to move a 16-word block. This is 285 ns, for a bandwidth of

(16 words × 4 bytes)/285 ns = 225 MB/sec

8.26 No solution provided

8.27 First, the synchronous bus has 50-ns bus cycles. The steps and times required for the synchronous bus are as follows:

Send the address to memory: 50 ns

Read the memory: 200 ns

Send the data to the device: 50 ns

Thus, the total time is 300 ns. This yields a maximum bus bandwidth of 4 bytes every 300 ns, or

$$\frac{4 \text{ bytes}}{300 \text{ ns}} = \frac{4 \text{ MB}}{0.3 \text{ seconds}} = 13.3\frac{\text{MB}}{\text{second}}$$

At first glance, it might appear that the asynchronous bus will be *much* slower, since it will take seven steps, each at least 40 ns, and the step corresponding to the memory access will take 200 ns. If we look carefully at Figure 8.10, we realize that several of the steps can be overlapped with the memory access time. In particular, the memory receives the address at the end of step 1 and does not need to put the data on the bus until the beginning of step 5; steps 2, 3, and 4 can overlap with the memory access time. This leads to the following timing:

Step 1: 40 ns

Steps 2, 3, 4: maximum (3×40 ns, 200 ns) = 200 ns

Steps 5, 6, 7: 3×40 ns = 120 ns

Thus, the total time to perform the transfer is 360 ns, and the maximum bandwidth is

$$\frac{4 \text{ bytes}}{360 \text{ ns}} = \frac{4 \text{ MB}}{0.36 \text{ seconds}} = 11.1 \frac{\text{MB}}{\text{second}}$$

Accordingly, the synchronous bus is only about 20% faster. Of course, to sustain these rates, the device and memory system on the asynchronous bus will need to be fairly fast to accomplish each handshaking step in 40 ns.

8.28 For the 4-word block transfers, each block takes

1. 1 clock cycle that is required to send the address to memory
2. $\frac{200 \text{ ns}}{5 \text{ ns/cycle}} = 40$ clock cycles to read memory
3. 2 clock cycles to send the data from the memory
4. 2 idle clock cycles between this transfer and the next

This is a total of 45 cycles, and 256/4 = 64 transactions are needed, so the entire transfer takes $45 \times 64 = 2880$ clock cycles. Thus the latency is 2880 cycles $\times$ 5 ns/cycle = 14,400 ns.

Sustained bandwidth is $\frac{256 \times 4 \text{ bytes}}{14{,}400 \text{ ns}} = 71.11$ MB/sec.

The number of bus transactions per second is

$$\frac{64 \text{ transactions}}{14{,}400 \text{ ns}} = 4.44 \text{ transactions/second}$$

For the 16-word block transfers, the first block requires

1. 1 clock cycle to send an address to memory
2. 200 ns or 40 cycles to read the first four words in memory
3. 2 cycles to send the data of the block, during which time the read of the four words in the next block is started
4. 2 idle cycles between transfers and during which the read of the next block is completed

Each of the three remaining 16-word blocks requires repeating only the last two steps.

Thus, the total number of cycles for each 16-word block is $1 + 40 + 4 \times (2 + 2) = 57$ cycles, and $256/16 = 16$ transactions are needed, so the entire transfer takes, $57 \times 16 = 912$ cycles. Thus the latency is 912 cycles $\times$ 5 ns/cycle = 4560 ns, which is roughly one-third of the latency for the case with 4-word blocks.

Sustained bandwidth is $\frac{256 \times 4 \text{ bytes}}{4560 \text{ ns}} = 224.56$ MB/sec

The number of bus transactions per second with 16-word blocks is

$$\frac{16 \text{ transactions}}{4560 \text{ ns}} = 3.51\text{M transactions/second}$$

which is lower than the case with 4-word blocks because each transaction takes longer (57 versus 45 cycles).

8.29 First the mouse:

Clock cycles per second for polling = $30 \times 400 = 12{,}000$ cycles per second

$$\text{Fraction of the processor clock cycles consumed} = \frac{12 \times 10^3}{500 \times 10^6} = 0.002\%$$

Polling can clearly be used for the mouse without much performance impact on the processor.

For the floppy disk, the rate at which we must poll is

$$\frac{50\frac{\text{KB}}{\text{second}}}{2\frac{\text{bytes}}{\text{polling access}}} = 25\text{K}\frac{\text{polling accesses}}{\text{second}}$$

Thus, we can compute the number of cycles:

$$\text{Cycles per second for polling} = 25\text{K} \times 400 = 10 \times 10^6$$

$$\text{Fraction of the processor consumed} = \frac{10 \times 10^6}{500 \times 10^6} = 2\%$$

This amount of overhead is significant, but might be tolerable in a low-end system with only a few I/O devices like this floppy disk.

In the case of the hard disk, we must poll at a rate equal to the data rate in four-word chunks, which is 250K times per second (4 MB per second/16 bytes per transfer). Thus,

$$\text{Cycles per second for polling} = 250\text{K} \times 400$$

$$\text{Fraction of the processor consumed} = \frac{100 \times 10^6}{500 \times 10^6} = 20\%$$

Thus one-fifth of the processor would be used in just polling the disk. Clearly, polling is likely unacceptable for a hard disk on this machine.

8.30 The processor-memory bus takes 8 clock cycles to accept 4 words, or 2 bytes/clock cycle. This is a bandwidth of 1600 MB/sec. Thus, we need 1600/40 = 40 disks, and because all 40 are transmitting, we need 1600/100 = 16 I/O buses.

8.31 Assume the transfer sizes are 4000 bytes and 16000 bytes (four sectors and sixteen sectors, respectively). Each disk access requires 0.1 ms of overhead + 6 ms of seek.

For the 4 KB access (4 sectors):

- Single disk requires 3 ms + 0.09 ms (access time) + 6.1 ms = 9.19 ms
- Disk array requires 3 ms + 0.02 ms (access time) + 6.1 ms = 9.12 ms

For the 16 KB access (16 sectors):

- Single disk requires 3 ms + 0.38 ms (access time) + 6.1 ms = 9.48 ms
- Disk array requires 3 ms + 0.09 ms (access time) + 6.1 ms = 9.19 ms

Here are the total times and throughput in I/Os per second:

- Single disk requires (9.19 + 9.48)/2 = 9.34 ms and can do 107.1 I/Os per second.
- Disk array requires (9.12 + 9.19)/2 = 9.16 ms and can do 109.1 I/Os per second.

8.32 The average read is (4 + 16)/2 = 10 KB. Thus, the bandwidths are

Single disk: 107.1 * 10 KB = 1071 KB/second.

Disk array: 109.1 * 10 KB = 1091 KB/second.

8.33 You would need I/O equivalents of Load and Store that would specify a destination or source register and an I/O device address (or a register holding the address). You would either need to have a separate I/O address bus or a signal to indicate whether the address bus currently holds a memory address or an I/O address.

8.34

a. If we assume that the processor processes data before polling for the next byte, the cycles spent polling are 0.02 ms * 1 GHz – 1000 cycles = 19,000 cycles. A polling iteration takes 60 cycles, so 19,000 cycles = 316.7 polls. Since it takes an entire polling iteration to detect a new byte, the cycles spent polling are 317 * 60 = 19,020 cycles. Each byte thus takes 19,020 + 1000 = 20,020 cycles. The total operation takes 20,020 * 1000 = 20,020,000 cycles.

   (Actually, every third byte is obtained after only 316 polls rather than 317; so, the answer when taking this into account is 20,000,020 cycles.)

b. Every time a byte comes the processor takes 200 + 1000 = 1200 cycles to process the data. 0.02 ms * 1 GHz – 1200 cycles = 18,800 cycles spent on the other task for each byte read. The total time spent on the other task is 18,800 * 1000 = 18,800,000 cycles.

8.38 Some simplifying assumptions are the following:

- A fixed overhead for initiating and ending the DMA in units of clock cycles. This ignores memory hierarchy misses adding to the time.
- Disk transfers take the same time as the time for the average size transfer, but the average transfer size may not well represent the distribution of actual transfer sizes.
- Real disks will not be transferring 100% of the time—far from it.

Network: (2 µs + 25 µs * 0.6)/(2 µs + 25 µs) = 63% of original time (37% reduction)

Reducing the trap latency will have a small effect on the overall time reduction

8.39 The interrupt rate when the disk is busy is the same as the polling rate. Hence,

$$\text{Cycles per second for disk} = 250\text{K} \times 500 = 125 \times 10^6 \text{ cycles per second}$$

$$\text{Fraction of the processor consumed during a transfer} = \frac{125 \times 10^6}{500 \times 10^6} = 25\%$$

Assuming that the disk is only transferring data 5% of the time,

$$\text{Fraction of the processor consumed on average} = 25\% \times 5\% = 1.25\%$$

As we can see, the absence of overhead when an I/O device is not actually transferring is the major advantage of an interrupt-driven interface versus polling.

8.40 Each DMA tranfer takes

$$\frac{8\text{ KB}}{4\dfrac{\text{MB}}{\text{second}}} = 2 \times 10^{-3}\text{ seconds}$$

So if the disk is constantly transferring, it requires

$$\frac{1000 + 500\dfrac{\text{cycles}}{\text{transfer}}}{2 \times 10^{-3}\dfrac{\text{seconds}}{\text{transfer}}} = 750 \times 10^{3}\ \frac{\text{clock cycles}}{\text{second}}$$

Since the processor runs at 500 MHz,

$$\text{Fraction of processor consumed} = \frac{750 \times 10^{3}}{500 \times 10^{6}} = 1.5 \times 10^{-3} = 0.15\%$$

8.44 Maximum I/O rate of the bus: 1,000,000,000/8000 = 125,000 I/O/second

CPU bottleneck restricts throughput to 10,000 I/O / second

Time/I/O is 6.11 ms at disk, each disk can complete 1000/6.11 = 163.67 I/O/second

To saturate the CPU requires 10,000 I/O second.

$$\frac{10,000}{163.67} \approx 61\text{ disks.}$$

$$\frac{61\text{ disks}}{7\text{ disks/scs1 controller}} = 9\text{ scs1 controllers.}$$

8.45 First, check that neither the memory nor the I/O bus is a bottleneck. Sustainable bandwidth into memory is 4 bytes per 2 clocks = 800 MB/sec. The I/O bus can sustain 100 MB/sec. Both of these are faster than the disk bandwidth of 40 MB/sec, so when the disk transfer is in progress there will be negligible additional time needed to pass the data through the I/O bus and write it into memory. Thus, we ignore this time and focus on the time needed by the DMA controller and the disk. This will take 0.1 ms to initiate, 8 ms to seek, 16 KB/40 MB to transfer: total = 8.5 ms.

8.46 Disk access total time: 10,000/500,000,000 s + 20 ms = 20.002 ms

% delay trapping to OS: 0.01%

Network access total time: 10,000/5,000,000,000 s + 25 µs = 27 µs

% delay trapping to OS: 7.4%

8.47 Disk: (2 μs + 20 ms * 0.4)/(2 μs + 20 ms) = 40% of original time (60% reduction)

Reducing the trap latency will have virtually no effect on the overall time reduction

Network: (2 μs + 25 μs * 0.6)/(2 μs + 25 μs) = 63% of original time (37% reduction)

Reducing the trap latency will have a small effect on the overall time reduction