Computer Science 605.611

Problem Set 11

1. A computer system can use any one of the three options listed below to manage the transfer of 10000 blocks of data each second to an attached I/O device over a 32-bit bus.

#1 interrupt driven I/O

#2 DMA based I/O

#3 direct programmed controlled I/O

The data blocks are evenly distributed throughout each 1-second interval and each block contains 4096 bytes of data. How many interrupts each second must be handled with each option?

a) (3) The number of interrupts per second with option 1 = \_**10000**\_\_\_\_

One interrupt per block

b) (3) The number of interrupts per second with option 2 = \_\_\_**1**\_\_\_\_\_\_

Once all blocks moved, has to interrupt once

c) (3) The number of interrupts per second with option 3 = \_\_\_**0**\_\_\_\_\_\_

No interrupts since the CPU has to constantly be polled

2. (5) Memory mapped I/O and port mapped I/O are two techniques for accessing I/O devices. Which one of these techniques is used for our MIPS system described in module 11?

**MIPS uses a Memory mapped I/O system.**

3.(5) Let “latency” be defined as the time for the CPU to detect the arrival of a unit of input data from an I/O device and to acquire (i.e., read) the data. Each of the following three techniques (1) interrupt driven I/O, (2) DMA based I/O, and (3) direct programmed controlled I/O is used to transfer a single byte of data over the same 8-bit bus. Which technique provides the shortest latency? Explain your answer.

For this small of a data transfer 3, direct programmed controlled I/O would provide the shortest amount of latency. This is because, interrupt driven I/O may cause a bunch of overhead due to needed to switch tasks, DMA based I/O causes a huge amount of overhead for just one byte, direct program controlled I/O causes no interrupts or context switching with this small of a data transfer.

4. (4) For each of the statements listed below, indicate whether the statement applies to (1) memory mapped I/O or to (2) port mapped I/O. Select either (1) or (2) as your answer for each statement.

* uses separate input/output instructions to transfer I/O data **2**
* reduces the amount of memory address space available for code and data **1**
* allows lw and sw instructions to be used to perform I/O **1**
* requires a separate I/O bus **2**

5. (5) A system includes a direct-mapped data cache with 8192 lines. Each cache line is 256 bytes in size. Interrupt driven I/O is used to receive four data bytes from some input device.

The device sends the data bytes one at a time via the device data register which is memory mapped to address 0xFFFF0008. A separate interrupt is sent to the CPU each time that the device sends a data byte. In response to each interrupt, an ISR (interrupt service routine) is invoked which reads a byte from the memory mapped device data register and merges the data byte into low byte of register $v0 after shifting the previous contents left 8 bits. After four reads, register $v0 will contain four bytes arranged from left to right within the register. The data bytes sent by the I/O device are 0xA1, 0xB2, 0xC3 and 0xD4. However, the final contents of register $v0 are observed to be 0xA1A1A1A1 instead of the expected pattern: 0xA1B2C3D4. Explain how this could happen, assuming that the device does not malfunction.

There are a few possibilities. The one that I can think of is that it is caused by the direct mapped data cache. Itll first read the first data byte A1 which is merged into the low byte of $v0. Since this is a interrupt driven I/O when B2 is being read and merged into the low byte the processor will attempt to access the memory cache to fetch the ISR code. This causes a cache miss. When the ISR executes it merged B2 into low bytes of $v0 evicting A1 from the cache. This continues to happen with C3 and D4 evicting the line containing the previous byte. This then causes $v0 to only see the line pointing to A1 (same cache line as the ISR code), eventually $v0 ends up merging a bunch of A1s into it giving us the observed final register 0xA1A1A1A1.

6. (5) A disk system has an average seek time of 20 milli-seconds. The seek time is the average time required to move the access head to a randomly selected track. The disk rotates at a constant rate. The total time to get to a randomly selected sector is the sum of the average seek time plus the average rotational latency. The average rotational latency is the average time required for the beginning of a randomly selected sector on the track to rotate to the point where it is beneath the access head. This rotational latency equals one-half the time it takes for one revolution of the disk. Each track contains 128 sectors and each sector contains 512 bytes of data. The disk rotates at the rate of 7680 RPM (revolutions per minute). How many micro-seconds does it take to position the access head to the beginning of a randomly selected sector on a randomly selected track?

20 milliseconds = average seek time

7680 RPM = (RPM/60seconds) = 128 revolutions per second or rotational time

Rotational latency = 0.5\*1/128 = 1/256 seconds = ~3.90625 milliseconds

Seek time + rotational latency = positioning access head = 23.90625 ms = **23,906.25 microseconds**

7. In general, a processor performs a read from memory or a read from an I/O device, by sending a read request along with the read address during one clock cycle and obtaining the requested data in the next clock cycle. If the requested data is not returned within the next clock cycle, the processor must pause and wait one or more extra additional cycles for the data to become available. These extra clock cycles are referred to as “wait states”.

a) (3) If the processor runs at a 50 MHz clock rate, what is the maximum number of reads per second that can be performed if no wait states are required for each read operation?

Since there are no wait states for each operation that means a read only takes 2 cycles. Since we have a clock rate of 50 MHz. We know that the maximum number of reads per second would be clock rate/cycles per read operation, thus we would get **25,000,000 reads per second**

b) (3) If the processor’s clock rate is increased to 100 MHz, what is the maximum number of reads per second that can be performed if 18 wait states are required for each read operation?

We can look at this problem exactly like we looked at the last one. This time however, the amount of clock cycles per each read operation is increased to 20 due to the 18 wait states. Our clock rate is now 100 MHz. This means we have a 100 MHz/20 cycles = **5,000,000 reads per second**

8. Shown below is a bus system with a central arbiter that handles competing requests from four I/O devices. Device 4 has a higher priority than Device 3. Devise 3 has a higher priority than Device2 and Device 2 has a higher priority than Device 1. The arbiter grants access to the highest priority request that is present. The system provides a total bandwidth of 160,000,000 bytes per second. The available bus bandwidth must be shared among the four devices. Data blocks transferred by each device are all 2048 bytes in size. For this problem, ignore any time consumed by the arbiter.

A diagram of a device

Description automatically generated

A single data block is transferred for each request that is granted by the arbiter. The number of transfers requested per second for the devices are as follows: Device 1 makes 20000 requests per second, Device 2 makes 10000 requests per second, Device 3 makes 30000 requests per second and Device 4 makes 40000 requests per second.

a) (5) If bus access is granted strictly on a priority basis, how many data blocks will each device actually be able to transfer per second?

Device 1 transfers \_\_0\_\_ blocks

Device 2 transfers \_\_0\_\_ blocks

Device 3 transfers \_\_0\_\_ blocks

Device 4 transfers \_40000\_ blocks

b) (5) Suppose instead that the system has a total bus bandwidth of 102,400,000 bytes per second. With a fair bus allocation policy, competing requests from devices are granted in a round robin fashion so that each device in turn is allowed to transfer one block at a time (e.g., Device 4, Device 3, Device 2, Device 1, Device 4, Device 3, Device 2, Device 1, etc.). Device 1 makes 12000 requests per second, Device 2 makes 10000 requests per second, Device 3 makes 8000 requests per second and Device 4 makes 40000 requests per second.

How many 2048-byte data blocks will each device get to transfer per second with this fair bus allocation policy?

Device 1 transfers \_9.98\_ blocks

Device 2 transfers \_12.5\_ blocks

Device 3 transfers \_15.63\_ blocks

Device 4 transfers \_3.13\_ blocks

Device bandwith = 102,400,000 = 25,600,000 per device

Transferred blocks equal device 25600000 / (RPS\*2048)

9. When an interrupt occurs, control must be transferred from the currently executing program to a subroutine that handles the activity required to service the interrupt. For each interrupt, a “context switch” is said to occur. In addition to transferring control, a context switch includes the saving and restoration of any required registers. Assume that it takes 1000 clock cycles to perform each context switch when an I/O device triggers an interrupt. The interrupt handler subroutine takes an additional 2,000 cycles to service the device request. Once the device has been serviced, another 1000 cycles are required to perform the context switch needed to return from the interrupt handler back to the program that was running. The CPU clock rate is 1 GHz.

a) (4) What is the maximum number of requests per second that a device can generate if the system must complete all activity associated with each interrupt, including the context switches, before the next interrupt occurs?

One interrupt is 4000 clock cycles. So we can determine the maximum number of requests per second quite easily. 1GHz/4000cc = **250,000**

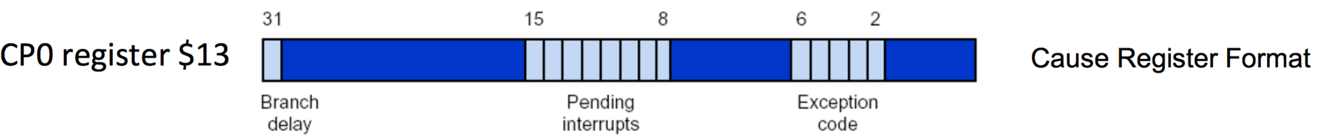
b) (4) Interrupt driven I/O is used for a different device that issues 10000 evenly spaced requests per second and transfers 1 byte per request. A total of 4000 clock cycles are required to handle each interrupt. Each instruction takes one clock cycle to execute. Part of each second is consumed in handling interrupts. How many instructions can be executed in the remaining part of each second that is not consumed by context switches and executing the interrupt handl?

Total Cycles consumed in a second is 40,000,000. Since the clock rate is 1 GHz, we can determine the remaining cycles to be 1GHz – Total cycles consumed. So **960,000,000** instructions can still be run in the remaining part of each second.

c) (4) Suppose that instead of interrupt driven I/O, DMA is used to transfer one data block each second. Each data block contains 10000 bytes of data. With DMA, it takes 5000 clock cycles to setup and manage each block transfer including the context switches and interrupt handler. Each instruction takes one clock cycle. How many instructions other than those required for I/O can the CPU execute each second?

The only variable we actually need to pay attention to is that it takes 5000 clock cycles to setup and manage the block transfer. Since only one data block is transferred each second only 5000 cycles are required per second for DMA. Due to this we can determine that **999,995,000 instructions** would remain to be used.

10. The registers used in managing exceptions and interrupts on our MIPS system are contained in the CP0 system coprocessor. CP0 has its own set of 32 registers and uses register $13 as the cause register. The format of the cause register is shown below. Bits 2 through 6 contain the current 5-bit exception code which identifies the type of exception that occurred. Bits 8 through 15 indicate which interrupts (#0 through #7) are currently pending (i.e., waiting to be serviced). The interrupt priorities range from the lowest for interrupt #0 up to the highest for #7.



a) (3) Write down a short MIPS instruction sequence that reads the cause register and places the exception code into CPU register $t0. Register $t0 should only contain the exception code right justified within the register. The instruction sequence must contain only true-ops. No CPU registers other than $t0 should be modified. No CP0 registers other than the cause register should be used by any of the instructions in your sequence.

**Mfc0 $t0, $13 - To copy the contents of CP0 into $t0**

**Srl $t0, $t0, 2 – shift logically right**

b) (5) If the current contents of CP0 register $13 is the two’s complement representation of decimal -2147472828, what is the current exception code (0 through 31) and which interrupt (#0 through #7) is the highest priority pending interrupt?

So we first need to see what is in $13,

1000 0000 0000 0000 0100 0000 0001 0100

We know that bits 2-6 are the exception code or **01000**

And bits 8-15 contain the pending interrupts 00000001

Since bits 8-15 are 00000001 and 1 is in the first position we can determine that the highest pending interrupt is **interrupt 0**

c) (3) Assume that the current contents of CP0 register $13 is the two’s complement representation of decimal -7. Use hex to show the final contents of CPU register $5 produced by the following instructions running on the multi-cycle data path:

addiu $13, $0, 15 – add sign extended immediate into $13

mfc0 $5, $13 – copy CPO $13 into register 5

addu $5, $5, $13 – add $5 and $13

So register $13 first contains 0x0000000F after addiu. Mfc0 is then ran and moves 0xFFFFFFF9 (-7) into register $5. Register 5 is then added against register $13 and stores it in $5 giving us

**0x00000008**

11. (10) RAID systems are defined and described in Module 11 Lectures Part 2.

A RAID system is to be constructed using some number of identical disk drives each of which can hold 32 terabytes of data. The disks in each system as a group must hold a large database of size 128 terabytes along with any required parity. What is the minimum number of disks required to hold the combined data and required parity if the disk system is implemented as a:

* 1. RAID0 system? Minimum number of disks required =\_4\_

b) RAID6 system? Minimum number of disks required = \_6\_

c) RAID4 system? Minimum number of disks required = \_5\_\_

d) RAID5 system? Minimum number of disks required = \_5\_\_

e) RAID1 system? Minimum number of disks required = \_8\_\_

12. Each of the stripes on a RAID 5 system contains five blocks: four data blocks along with the corresponding parity block. The parity block is computed as the cumulative XOR of the 4 data blocks. Blocks are also called strips. There are no strips or parity blocks currently in memory. For the purposes of this problem, assume that each strip or block contains only 4 bits.

A diagram of a number of strips

Description automatically generated with medium confidence

a) (3) The following information is given: strip4 =1110, strip5 = 1001, strip6 = 0101,

strip7 = 0111 and P4-7 = 0101.

Strip6 is overwritten with the new pattern 0110. This also requires overwriting the parity block P4-7 with a new pattern. What is the pattern for the new parity block P4-7?

To determine this we need to XOR between block 4, 5, 6 (new pattern), and 7. This gives us the final 4 bit pattern in P4-7, **0110**.

b) (5) Strip15 is overwritten with the pattern 0110 (call it Nstrip15). This also requires overwriting the parity block P12-15 with a new pattern (call it NP12-15).

Can the new pattern NP12-15 be determined based just on the following information?

strip12 = 0001, strip13 = 1111, strip15 = 0111, Nstrip15 = 0110 and P12-15 = 0011.

If not, explain why not.

If yes, show the 4-bit pattern for NP12-15.

This can be determined. To do this we need to a similar process to the last question. We XOR P12-15, Strip15, and Nstrip15. This leaves us with **0010**.

13. (10) Shown below is a RAID-DP (i.e., RAID6 with dual parity) disk system. D1, D2, D3 and D4 are the data disks. RP is the horizontal or row parity disk and DP is the diagonal parity disk. For the purposes of this problem, each strip or block on a disk is a 4-bit pattern. Two of the disks, D2 and D4 have crashed so their data contents can no longer be accessed. None of the data from any disk is currently in memory.

A group of colorful rectangular boxes with numbers

Description automatically generated

Module 11 example set 7 illustrates a technique to reconstruct blocks on up to 2 disks that have failed and are unavailable. Use the scheme described in the example set, substituting exclusive-OR in place of addition, to reconstruct the contents of the two missing disks by determining the 4-bit pattern for each of the following:

D2\_blue = \_0100\_ D4\_red= \_1001\_

D2\_purple= \_1001\_ D4\_orange= \_0101\_

D2\_white= \_0011\_ D4\_blue= \_0111\_

D2\_red= \_0001\_ D4\_purple= \_1111\_

Fill-in the blanks above to show the correct 4-bit patterns.