Advanced Computer Architecture

Lecture No. 30

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.3.3, 8.4

Summary

- Nested Interrupts
- Interrupt Mask
- DMA

Nested Interrupts

(Read from Book, Jordan Page 397)

Interrupt Mask

(Read from Book, Jordan Page 397)

Priority Mask

(Read from Book, Jordan Page 398)

Examples

Example # 1[23]
Assume that three I/O devices are connected to a 32-bit, 10 MIPS CPU. The first device is a hard drive with a maximum transfer rate of 1MB/sec. It has a 32-bit bus. The second device is a floppy drive with a transfer rate of 25KB/sec over a 16-bit bus, and the third device is a keyboard that must be polled thirty times per second. Assuming that the polling operation requires 20 instructions for each I/O device, determine the percentage of CPU time required to poll each device.

Solution:
The hard drive can transfer 1MB/sec or 250 K 32-bit words every second. Thus, this hard drive should be polled using at least this rate.

Using $1K=2^{10,}$ the number of CPU instructions required would be

$$250 \times 2^{10} \times 20 = 5120000 \text{ instructions per second.}$$

[23] Adopted from [H&P org]

Percentage of CPU time required for polling is

$$(5.12 \times 10^6)/ (10 \times 10^6) = 51.2\%$$

The floppy disk can transfer 25K/2= 12.5×2^{10} half-words per second. It should be polled with at least this rate. The number of CPU instructions required will be $12.5 \times 2^{10} \times 20 = 256{,}000$ instructions per second.

Therefore, the percentage of CPU time required for polling is

$$(0.256 \times 10^6)/ (10 \times 10^6) = 2.56\%.$$

For the keyboard, the number of instructions required for polling is

$$30 \times 20 = 600 \text{ instructions per second.}$$

Therefore, the percentage of CPU time spent in polling is

$$600 / (10 \times 10^6) = 0.006\%$$

It is clear from this example that while it is acceptable to use polling for a keyboard or a floppy drive, it is very risky to use polling for the hard drive. In general, for devices with a high data rate, the use of polling is not adequate.

Example # 2[2]

a. What should be the polling frequency for an I/O device if the average delay between the time when the device wants to make a request and the time when it is polled, is to be at most 10 ms?
b. If it takes 10,000 cycles to poll the I/O device, and the processor operates at 100MHz, what % of the CPU time is spent polling?
c. What if th[24]e system wants to provide an average delay of 1msec?

Solution:

a. Assuming that the I/O requests are distributed evenly in time, the average time that a device will have to wait for the processor to poll is half the time between polling attempts. Therefore, to provide an average delay of 10 ms, the processor will have to poll every 20 ms, or 50 times per second.
b. If each polling attempt takes 10,000 cycles, then the processor will spend 500,000 cycles polling each second. The % of CPU time spent in polling is then $(0.5 \times 10^6)/(100 \times 10^6)=0.5\%$
c. To provide an average delay of 1ms, the polling frequency must be increased. The processor will have to poll every 2ms, or 500 times per second. This will consume 5,000,000 cycles for polling. The % of CPU time spent polling then becomes 5/100=5%.

[24] Adopted from [Schaum]

Example # 3[25]
What percentage of time will a 20MIPS processor spend in the busy wait loop of an 80-character line printer when it takes 1 msec to print a character and a total of 565 instructions need to be executed to print an 80 character line. Assume that two instructions are executed in the polling loop.

Solution:
Out of the total 565 instructions executed to print a line, 80x2=160 are required for polling. For a 20MIPS processor, the execution of the remaining 405 instructions takes $405/ (20x10^6) = 20.25\mu sec$. Since the printing of 80 characters takes 80ms, (80-0.02025) =79.97msec is spent in the polling loop before the next 80 characters can be printed. This is 79.97/80=99.96% of the total time.

Example # 4[26]
Consider a 20 MIPS processor with several input devices attached to it, each running at 1000 characters per second. Assume that it takes 17 instructions to handle an interrupt. If the hardware interrupt response takes 1μsec, what is the maximum number of devices that can be handled simultaneously?

Solution:
A service for one character requires $17/ (20x10^6) +1\mu sec=1.85\mu sec$. Since each device runs at 1000 characters per second, 1.85 ms of handling time is required by each device every second. Therefore the maximum number of devices that can be handled is $1/ (1.85x10^{-3}) = 540$.

Example # 5[27]
Assume that a floppy drive having a transfer rate of 25KB per second is attached to a 32 bit, 10MIPS CPU using an interrupt driven interface. The drive has a 16-bit data bus. Assume that the interrupt overhead is 20 instructions. Calculate the fraction of CPU time required to service this drive when it is active.

Solution:
Since the floppy drive has a 16-bit data bus, it can transfer two bytes at one time. Thus its transfer rate is 25/2 = 12.5K half-words (16-bits each) per second. This corresponds to an overhead of 20 instructions or $12.5K \times 20 = 12.5 \times 2^{10} \times 20 = 256000$ instructions per second.

Example # 6[28]
A processor with a 500 MHz clock requires 1000 clock cycles to perform a context switch and start an ISR. Assume each interrupt takes 10,000 cycles to execute the ISR

[25] Adopted from [H&J]
[26] Adopted from [H&J]
[27] Adopted from [H&P org]
[28] Adopted from [Schaum]

and the device makes 200 interrupt requests per second. Also, assume that the processor polls every 0.5msec during the time when there are no interrupts. Further assume that polling an I/O device requires 500 cycles. Compute the following:

a. How many cycles per second does the processor spend handling I/O from the device if only interrupts are used?
b. What fraction of the CPU time is used in interrupt handling for part (a)?
c. How many cycles per second are spent on I/O if polling is also used with interrupts?
d. How often should the processor poll so that polling incurs the same overhead as interrupts?

Solution:

a. The device makes 200 interrupt requests per second, each of which takes
10,000 + 2x1000 (context switching to the ISR and back from it)
= 12,000 cycles.

   Thus, a total of 200x12,000=2,400,000 cycles per second are spent handling I/O using interrupts.

b. The percentage of the processor time used in interrupt handling is 2,400,000/(500x$10^{6)}$ or 0.48%.

c. There are 200 interrupt requests per second, or one interrupt request every 5 ms. Every interrupt consumes a total of 12,000 cycles, as calculated in part (a). For a 500 MHz CPU, this is

   12000/(500 x 10^6) = 24 microseconds.

   For 200 interrupts per second, this is 4.8 msec.

   This leaves 1000 - 4.8 = 995.2 msec for polling.

   Since the processor polls once every 0.5 msec during the time when there is no interrupt, this corresponds to

   995/0.5 = 1990 times per second.

   The total number of cycles required for polling is

   1990 x 500 = 995,000 cycles per second.

   Thus, the total time spent on I/O when using polling with interrupts is

2,400,000 + 995,000 = 3,395,000 cycles per second.

d. The interrupt overhead is 1000 cycles per second for a context switch to the ISR and 1000 cycles per second back from it. This is a total of 2 x 1000 cycles per second. With 200 interrupts per second, this is
200 x 2000 = 400,000 cycles per second.

The polling overhead is 500 cycles per second. Thus, for the same overhead as interrupts, the polling operation should be performed
400,000 / 500 = 800 times per second,
or 1/800 = every 1.25 msec.

Direct Memory Access (DMA)

Direct memory access is a technique, where by the CPU passes its control to the memory subsystem or one of its peripherals, so that a contiguous block of data could be transferred from peripheral device to memory subsystem or from memory subsystem to peripheral device or from one peripheral device to another peripheral device.

Advantage of DMA

The transfer rate is pretty fast and conceptually you could imagine that through disabling the tri-state buffers, the system bus is isolated and a direct connection is established between the I/O subsystem and the memory subsystem and then the CPU is free. It is idle at that time or it could do some other activity. Therefore, the DMA would be quite useful, if a large amount of data needs to be transferred, for example from a hard disk to a printer or we could fill up the buffer of a printer in a pretty short time.
As compared to interrupt driven I/O or the programmed I/O, DMA would be much faster. What is the consequence? The consequence is that we need to have another chip, which is a **DMA controller.** "A DMA controller could be a CPU in itself and it could control the total activity and synchronize the transfer of data". DMA could be considered as a technique of transferring data from I/O to memory and from memory to I/O without the intervention of the CPU. The CPU just sets up an I/O module or a memory
subsystem, so that it passes control and the data could be passed on from I/O to memory or from memory to I/O or within the memory from one subsystem to another subsystem without interaction of the CPU. After this data transfer is complete, the control is passed from I/O back to the CPU.
Now we can illustrate further the advantage of DMA using following example.

Example of DMA

If we write instruction load as follows:

```
load [2], [9]
```

This instruction is illegal and not available in the SRC processor. The symbols [2] and [9] represent memory locations. If we want to have this transfer to be done then two steps would be required. The instruction would be:

```
load  r1, [9]
store r1, [2]
```

Thus it is not possible to transfer from one memory location to another without involving the CPU. The same applies to transfer between memory and peripherals connected to I/O ports. For example we cannot have:

```
out [6], datap
```

It has to be done again in two steps:

```
load r1, [6]
out   r1, datap
```

Similar comments apply to the "in" instruction. Thus the real cause of the limited transfer rate is the CPU itself. It acts as an unnecessary middle man. The example illustrates that in general, every data word travels over the system bus twice and this is not necessary, and therefore, the DMA in such cases is pretty useful.

DMA Approach

The DMA approach is to turn off i.e. through tri-state buffers and therefore, electrically disconnect from the system bus, the CPU and let a peripheral device or a memory subsystem or any other module or another block of the same module communicate directly with the memory or with another peripheral device. This would have the advantage of having higher transfer rates which could approach that of limited by the memory itself.

Disadvantage of DMA

The disadvantage however, would be that an additional DMA controller would be required, that could make the system a bit more complex and expensive. Generally, the DMA requests have priority over all other bus activities including interrupts. No interrupts may be recognized during a DMA cycle.