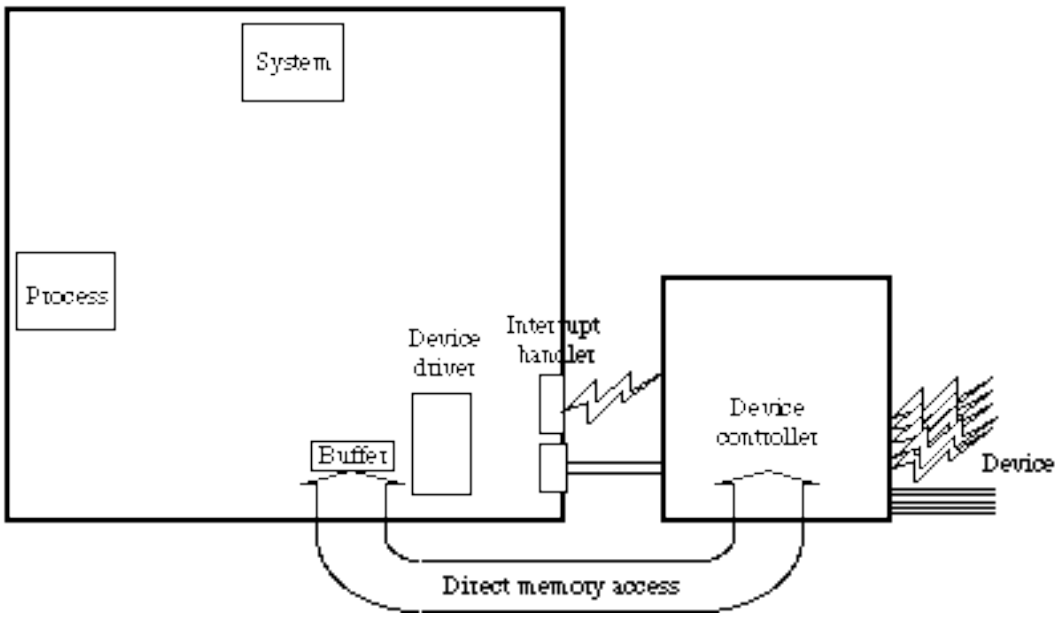
DEVICE CONTROL HARDWARE

One way to reduce the demands on the central processor is to provide special-purpose hardware to look after the specific needs of various devices. In this pattern of operation, the device itself is connected to the computer through an interface box, which looks after the character-by-character (or whatever) management of the device, but only communicates with the computer comparatively infrequently to transmit or receive a whole record (or whatever) of data. Such boxes are called **device controllers** or **input-output processors** or **channels**, and we have already mentioned them in the specific context of discs in the chapter *DISCS – THE HARDWARE VIEW*. You can think of these interfacing devices as hardware implementations of **doio** – or, at least, of what happens once **doio** has decided which device to use. We are concerned only with the operating system end of the controllers – which illustrates, of course, the whole point of having them.

Simple controllers might be little more than hardware buffers which collect input from the device until a record is full, then send it to the computer, or accept a record from the computer and manage the communication with the device. Obvious interlocks and status flags are provided – typically buffer full, buffer empty, device fault, and similar indicators. (Compare the software implementation in *DEVICE CONTROL SOFTWARE*.) Such simple interface devices can be very useful if they do what you want them to do – but they can cause serious problems if they don't.

A device controller is attached to the computer in just the same way as any other device so far as the control operations are concerned, though it's common to provide some form of direct memory access if large quantities of data have to be communicated quickly. The combined system can be thought of like this :



More elaborate controllers are themselves programmable, so the operating system can determine the controller's behaviour to suit each transaction. In the example given below, the programme executed by the controller lives in the computer's memory, so the computer is clearly in control.

EXAMPLES.

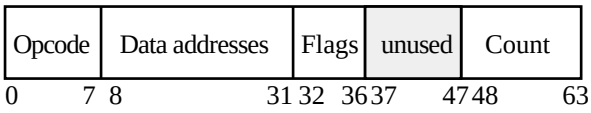
The first example[INT5] which follows shows how the idea of a programmable device controller can be implemented in practice. The fine detail isn't particularly important for this course; but you should be aware of the general principles involved. The description is fairly old, but much the same techniques are still in use. The second example is of a quite different, and much more recent, hardware system which nevertheless uses the same channel interface, ensuring complete device independence.

2-3.3.4 INSTRUCTIONS

What classes of commands does the processor interpret? There are three basic groupings of I/O commands:

1 Data transfers: read; read backwards; write; sense (read device status)
2 Device control: control (page eject, tape rewind, etc.)
3 Branching: transfers of control within the channel program

The channel fetches the channel commands [Channel Command Words (CCW)] from memory and decodes them according to the following format.



The *opcode* (bits 0-7) indicates the command to be performed; it actually consists of two parts: 2 to 4 operation bits and 4 to 6 modifier bits. The operation bits are standard while the modifier bits vary for each type of device.

The *data address* (bits 8-31) specifies the beginning location of the data field referenced by the command. The *count field* specifies the byte length of the data field. The data address and count are used primarily for the data transfer-type commands.

The *flag bits* further specialize the command. The principal flags are:

1 The *command chain* flag (bit 33) denotes that the next sequential CCW is to be executed on normal completion of the current command. (*Note:* under default conditions, i.e., all flags = 0, the channel stops at completion of current command.)

2 The *data chain* flag (bit 32) denotes that the storage area designated by the next CCW is to be used with the current command, once the current data area count is exhausted.

3 The *suppress length indication* flag (bit 34) suppresses the indication to the program of an incorrect length (see Figure 2-12).

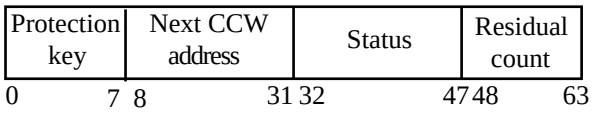
4 The *skip* flag (bit 35) species suppression of transfer of information to storage .

5 The *programmed controlled interruption* flag (bit 36) causes the channel to generate an interruption condition when this CCW takes control of the channel.

A group of multiple CCWs linked together by command chains, data chains, or transfers is called a *channel program*. The CPU starts the channel executing a channel program.

2-3.3.5 SPECIAL FEATURES

Status The channel has an internal register that acts as the instruction address register. (A multiplexor or block multiplexor actually has several such registers, one per device.) In addition, three specific words of memory are used for status information. The Channel Address Word (CAW), which starts at location 72_{10} in core, contains the address of the first instruction to be executed by the channel. The channel refers to the CAW only during the execution of the Start I/O (SlO) instruction by the CPU. The Channel Status Word (CSW), actually a doubleword, contains coded information indicating the status of the channel. The format of the CSW, located at location 64, is as follows:



Key – protection key being used by channel

Address – address of next channel command

Status – e.g., building on fire, I/O completed, I/O error occurred, etc.

Count – how many bytes of the last CCW were not processed? (usually zero unless the channel abnormally terminated an I/O operation)

2-3.4 Examples of I/O Programs

Let us write a program to skip to the top of a new page, print two lines, and eject to the next page. Figure 2-12 contains four I/O commands coded in hexadecimal to perform our task.

Appendix A presents the I/O command opcodes for various devices; the I/O program above is intended for a model 3211 printer device. The first command (opcode 8B) causes the printer to advance the paper to the top of the next page. In Appendix A this opcode can be found under the 3211 description Skip to Channel N Immediately (opcode 1NNNN001). Thus $8B_{16}$ = 10001011 means Skip to Channel 1 Immediately.

LOCATION	DATA					
	OPCD	ADDR	FLAG	UNUS	CNT	
400	8B	?	40	00	XXXX	SKIPTONEW PAGE
408	01	010008	60	00	0008	PRINTLINE
410	0B	?	40	00	XXXX	ADVANCE1LINE
418	89	010010	00	00	0014	PRINT20-CHARACTER LINE:EJECTTO NEXTPAGEAND STOP
010008		'GOODDAY'				
010010		'YOUARENOWONA370				

Figure 2-12 Example of an I/O channel program

The channel referred to is not an I/O channel. There is a control tape or chain that is internal to the printer and has holes or marks in several columns (channels). By convention, channel 1 has a mark that corresponds to the top of the page. This is necessary since various sizes of paper may be used in the printer (regular size 11-inch long paper, 13-inch legal-length paper, etc.). The second command, opcode = 01_{16}, prints the eight characters stored at location 010008 (GOOD DAY). The next command, opcode = $0B_{16}$, advances the paper by one line. This command is necessary because the 01 opcode above does not cause the paper to move, and without the command three 01's in a row would all print to be on the same line — this would save a lot of paper but would generally be a bit difficult to read. The second and third commands could have been combined into the single Write, Space 1 after Print command, opcode = 09_{16}. The final command, opcode = 89_{16}, prints the 20 characters that are stored at location $0100l0_{16}$ (YOU ARE NOW ON A 370) and then skips to the top of the next page.

In Figure 2-12 all of the I/O commands except the last have the command chain flag set (40_{16}). In addition, the Suppress Length Indication (SLI) flag (20_{16}) is required in the second CCW (flag 60 = 40 + 20) since the data count of 8 bytes is different from the standard data count of 132 bytes used by a 3211 printer. If the SLI flag were not set, the I/O program would stop after executing the second command. The SLI should also be used in the last CCW, but since the I/O program is going to stop then anyway, it is not really needed. The data address field and count fields are not used in control commands such as Skip 1 Line and Eject to New Page.

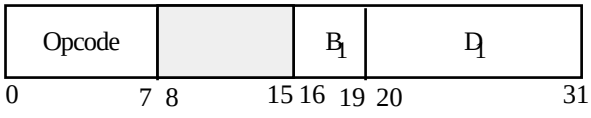
2-3.5 Communications between the CPU and the Channel

Now that we have seen how the channel itself works, we will examine how it communicates with the CPU. The purpose of having a channel is to free the CPU from having to control detailed I/O operations. The CPU and the channel are usually in a master/slave relationship. This means that the CPU tells the channel when to start and commands it to stop or change what it is doing. On the other hand, the channel cannot usually start any operation unless instructed by the CPU.

There are two types of communications between the CPU and the channel: (1) CPU to channel I/O instructions initiated by the CPU and (2) channel to CPU interrupt initiated by the channel.

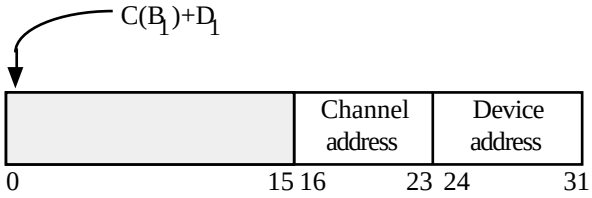
This section will describe the first of these types, the relationship between I/O instructions and the channel. The second type of communication, description of the I/O interrupts, will be left for Section 2-4.

All CPU I/O instructions have the following format:



The channel and device number are specified by the sum of the contents of register B_1 and the contents of the D_1 field. Bits 16-23 of the sum contain the

channel address, while bits 24-31 contain the device on the channel.



We are mainly concerned with three CPU I/O instructions:

1 START I/O (SIO): Two items are needed to start I/O: (1) the channel and device number and (2) the beginning address of the channel program. A START I/O instruction, such as SIO X'00E', specifies the channel number 0 and device number 0E. Locations 72-75 in memory contain the CAW, which specifies the start of the channel program.

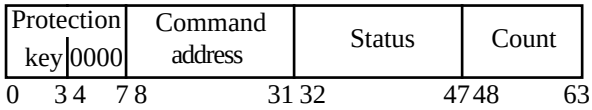
2 TEST I/O (TIO): The CPU indicates the state of the addressed channel and device by setting the Condition Code (busy or not). The CC can then be tested by the standard branch conditional instruction.

3 HALT I/O (HIO): Execution of the current I/O operation at the addressed I/O device and channel is abruptly terminated.

After executing an SIO or a TIO, the CPU gets a Condition Code of either:

8 ~ OK (not busy)
2 ~ busy
1 ~ not operational
4 ~ indicates that there is a lot more to tell us in the CSW, which was just stored at location 64.

The Channel Status Word provides the detail status of an I/O device or the conditions under which an I/O operation has been terminated. The CSW may be set in the process of I/O interrupts and sometimes during execution of START I/O, HALT I/O, and TEST I/O. The format of the CSW is:



An SIO causes I/O to start only if the channel returns a Condition Code of 8. If any other CC is returned; the channel has rejected the I/O request. The reason for the rejection can be found in the CC or CSW.

Although the I/O interrupt mechanism has some powerful capabilities, as explained in Section 24, it is not needed to perform simple I/O processing. For example, assuming that I/O interrupts are disabled, the following sequence will start up an I/O program and check that it completes correctly:

```
         .
         .
         .
         LA    1,CCWADOR     SET I/O PROGRAM
         ST    1,72          ADDRESS INTO CAW
         SIO   X'OOE'        START I/O ON
                             DEVICE X'OOE'
         BC    4+2+1,ERROR   IF NOT CC=8, THEN
                             ERROR
TESTIO   TIO   X'OOE'        TEST IF I/O
                             COMPLETED YET
         BC    4+2,TESTIO    IF STILL BUSY, TEST
                             AGAIN
         .
         .
         .
```

If we monitored the Condition Code of each Test I/O, the sequence would be:

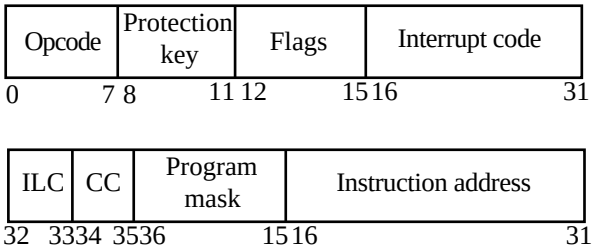
TIO CONDITION CODE

2 ~ busy
2 ~ busy
.....
2 ~ busy
} I/O in progress
4 ~ CSW storedI/O completed
8 ~ OK (not busy)
loop completed, CC ≠ 4 or 2.

2-4.2 Interrupt Mechanism

Now we must consider how CPU status can be saved and control transferred to an interrupt handling routine. The "state" or current condition of the CPU is stored in a doubleword register called the Program Status Word. This corresponds to the CSW of the I/O processor. The PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The active or controlling PSW is the "current" PSW. By storing the PSW during an interruption, the status of the CPU can be preserved for inspection or reloading. By loading a new PSW, or part of one, the state of the CPU can be changed.

The format of the PSW is as follows:



Each of the five classes of interrupts – I/O, program, supervisor call, external, and machine-check – has associated with it two doublewords, called "old" and "new" PSWs, stored in the main memory at predetermined storage locations. When an interrupt occurs, the interrupt hardware mechanism (1) stores the current PSW in the old position and (2) loads the current PSW from the new position.

The interrupt routine can access the appropriate old PSW to ascertain the condition that caused the interrupt. The old PSW contains an interrupt code and the location of the program being executed when the interrupt occurred.

Each of the program interrupt and external interrupt causes has a unique interrupt code: invalid operation = 01; privileged operation = 02; fixed point overflow = 08, etc., for program interrupts. When an I/O interrupt occurs, the PSW interrupt code indicates the channel and device causing the interrupt. The CSW status field, which is automatically set at the same time, contains information that indicates the cause of the interrupt.

2-4.3 Interrupt Handler Processing

A big advantage of using special hardware for device control is that it fits in well with ideas of device independence – the principle that it should be possible to write a programme without knowing any details of the devices which it will eventually use. A device controller effectively provides its device with a standard interface which is much easier for the operating system to handle than a collection of quite different devices.

Here is a description of a device[IMP34] which is interchangeable with a disc pack. A lot of the details are not particularly relevant to this course, but we've left some in for interest. It's worth looking at the rest of the material for the light it throws on other parts of the course.

The economics of semiconductor storage were revolutionised earlier this decade by the wide availability of 256-Kbit dynamic random access memory, and the competitive position has been assured by the price/performance advantages and favourable environmental characteristics of 1-Mbit dram. Currently, most plug compatible manufacturers offer semiconductor storage. This article is based upon the Hitachi SC-12-1/SU-16-1 which is marketed and supported in Europe as the Comparex 6086-S1/6580-1.

Semiconductor storage subsystems typically comprise a controller, which presents a standard IBM 370 or 370/XA channel interface, and a semiconductor storage device, which supplants the head-disk assembly of a conventional disk. The subsystem emulates standard devices such as 3880/3380, though not all cylinders may be present. All CCWs applicable to the emulated devices are accepted, and appropriate CSWs are returned.

A semiconductor storage subsystem can comprise between one and four storage directors connected to between one and four semiconductor storage devices each of between 1 6-Mb and 256-Mb capacity. A storage director may be connected to as many as four channels on four separate cpus. A typical maximal configuration is illustrated in figure 1.

Data is stored electronically and so access is not subject to the electro-mechanical delays of latency due to disk platter rotation, and seek/settle due to head movement. This results in a data access time of 0.3 milliseconds, and a device service time of little more than data transfer time for all read and write i/o activity.

Conventional rotating disks usually disconnect from the channel while performing relatively slow mechanical movements, and may fail to reconnect for data transfer if the selected path is in use

(370 SCP) or all available paths are in use (XA SCP with DPS/DPR devices).

This is known as a rotation position sensing miss, and involves a penalty of one complete disk revolution each time it occurs, with consequent deterioration of device response time.

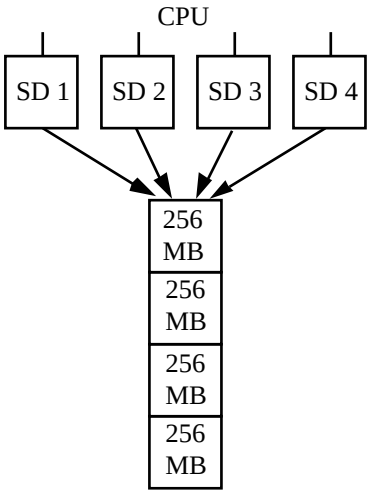


Figure 1: Maximal configuration of 1 GB with 4-path support

Semiconductor storage subsystems do not disconnect from the channel between command sequences, and so maintain excellent performance characteristics under conditions of very heavy loading. Semiconductor storage therefore eliminates disconnect time (latency + seek + rps miss) and offers a device service time of only connect time (protocol time plus transfer time) plus 0.3 milliseconds. This is illustrated with some typical device performance statistics in figure 2.

Clearly, it is feasible to reduce conventional device service time, and hence device utilisation, by a factor of 10. This means that at conventional i/o rates, device response times of around 2 milliseconds are achievable, and even at the very high i/o rate of 300 i/o/second device response times of around 6 milliseconds can be supported.

The implication for online applications is that transaction rates can be increased while simultaneously offering reduced response time. Similarly, for batch applications, throughput volume may be increased while simultaneously reducing turnaround time.

Data transfer time is dependent on datablock size and channel speed. With small blocks and standard 3 Mb/second channels, data transfer time will be low. With large blocks, however, transfer time becomes more significant and 4.5 Mb/second channels should be considered.

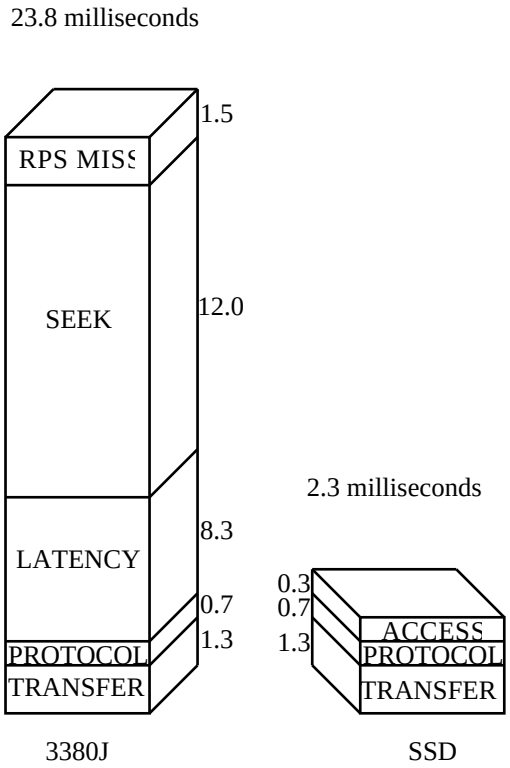


Figure 2: Device service time components (Not drawn to scale)

The first generation of semiconductor storage devices were volatile in that power loss resulted in data loss. This ensured that the device was suitable only for data which could be re-created easily.

Present generation devices incorporate battery back-up in conjunction with integral Winchester disk storage. This circumvents volatility in all circumstances of scheduled or unscheduled power-off, and ensures suitability for all performance-critical data.

It is worth observing that scheduled power-off data is written to the integral Winchester disk storage before the power-off process is allowed to complete. In circumstances of unscheduled power-off or power loss, the battery back-up powers the semiconductor memory and the Winchester disks and so data is written to the stable medium as soon as practicable and does not have to wait for power restitution. On power-up the data is read

from Winchester disk into semiconductor memory. Unload time is 8 minutes maximum, load time is 4 minutes maximum.

Note that data is written to the Winchester disks only during power-off processing. In normal operation data is stored only in semiconductor storage and so performance is truly at semiconductor speed. The Winchester disks are not defined to the operating system and cannot be accessed directly.

Many workloads feature high activity data files which are a serious constraint for online response/transaction rates or batch throughput/volume. Semiconductor storage can be a cost-effective solution to critical application performance.

Many DB/DC systems, particularly when used with 4GLs, can make very heavy usage of scratch pad areas. For various performance reasons concerning virtual and real storage these areas are usually on disk, but conventional devices often cannot support adequate transaction rates. Bottlenecks such as these are eliminated easily by semiconductor storage.

Database indexes of all types, including the Adabas associator, are ideal candidates for semiconductor storage because storage requirements tend to be low and access rates high.

Semiconductor storage always has had a role as a high performance paging device. In the past, esoteric fixed-head devices were simulated in order to optimise behaviour of the auxiliary storage manager. However, modern paging algorithms automatically prefer the devices offering the best performance so elaborate configuration of the paging subsystem is unnecessary.

Semiconductor storage can have a dramatic impact upon a virtual storage system which is constrained on real storage and hence paging heavily. As a result of the very fast device response time offered by semiconductor storage page, delay time can be reduced. This in turn will improve application response time and so transaction residency in real storage will reduce. This means that real storage contention will decline and paging rates may even fall.

NOTE : We don't suppose he *really* wants to reduce device utilisation, but that's what he said.

COMPARE :

Lane and Mooney[INT3] : Sections 8.3, 9.4; Silberschatz and Galvin[INT4] : Sections 2.1 and 2.2.

REFERENCE.

INT5 : S.E. Madnick, J.J. Donovan : *Operating systems* (McGraw-Hill, 1974).

IMP34 : M. Donnelly : "Making I/O go faster", *Computing*, 5 January 1989, page 14.