Chapter 29: Operating Systems

Author's Note: *This chapter is copied almost verbatim from the material in Chapter 21 of the textbook by Peter Abel. It is used by permission.*

This chapter introduces material that is suitable for more advanced assembler programming. The first section examines general operating systems and the various support programs. Subsequent sections explain the functions of the program status word and the interrupt system. Finally, there is a discussion of input/output channels, physical IOCS, and the input/output system.

These topics provide an introduction to systems programming and the relationship between the computer hardware and the manufacturer's software. A knowledge of these features can be a useful asset when serious bugs occur and when a solution requires an intimate knowledge of the system.

In an installation, one or more systems programmers, who are familiar with the computer architecture and assembler language, provide .support for the operating system. Among the software that IBM supplies to support the system are language translators such as assembler, COBOL, and PL/I and utility programs for cataloging and sorting files.

OPERATING SYSTEMS

Operating systems were developed to minimize the need for operator intervention during the processing of programs. An operating system is a collection of related programs that provide for the preparation and execution of a user's programs. The system is stored on disk, and part of it, the supervisor program, is loaded into the lower part of main storage.

You submit job control commands to tell the system what action to perform. For example, you may want to assemble and execute a source program. To this end, you insert job control commands before and after the source program and submit it as a job to the system. In simple terms, the operating system performs the following steps:

1. Preceding the source program is a job control command that tells the operating system to assemble a program. The system loads the assembler program from a disk library into storage and transfers control to it for execution.

2. The assembler reads and translates the source program into an object program and stores it on disk.

3. Another job control command tells the system to link–edit the object program. The system loads the linkage editor from a disk library into storage and transfers control to it for execution.

4. The linkage editor reads and translates the object program, adds any required input/output modules, and stores it on disk as an executable module.

5. Another job control command tells the system to execute the executable module. The system loads the module into storage and transfers control to it for execution.

6. The program executes until normal or abnormal termination, when it returns processing control to the system.

7. A job command tells the system that this is the end of the job, since a job may consist of any number of execution steps. The system then terminates that job and prepares for the next job to be executed.

Throughout the processing, the system continually intervenes to handle all input/output, interrupts for program checks, and protecting the supervisor and any other programs executing in storage.

IBM provides various operating systems, depending on users' requirements, and they differ in services offered and the amount of storage they require. These are some major operating systems:

DOS	Disk Operating System	Medium-sized systems
DOS/VSE	Disk Operating System	Medium-sized systems with virtual storage
OS/VSl	Operating System	Large system
OS/VS2	Operating System	Large system
OS/MVS	Operating System	Large system

Systems Generation

The manufacturer typically supplies the operating system on reels of magnetic tape, along with an extensive set of supporting manuals. A systems programmer has to tailor the supplied operating system according to the installation's requirements, such as the number and type of disk drives, the number and type of terminals to be supported, the amount of processing time available to users, and the levels of security that are to prevail. This procedure is known as ***systems generation***, abbreviated as ***sysgen***.

Operating System Organization

Figure 29–1 shows the general organization of Disk Operating System (DOS), on which this text is largely based. The three main parts are the control program, system service programs, and processing programs.

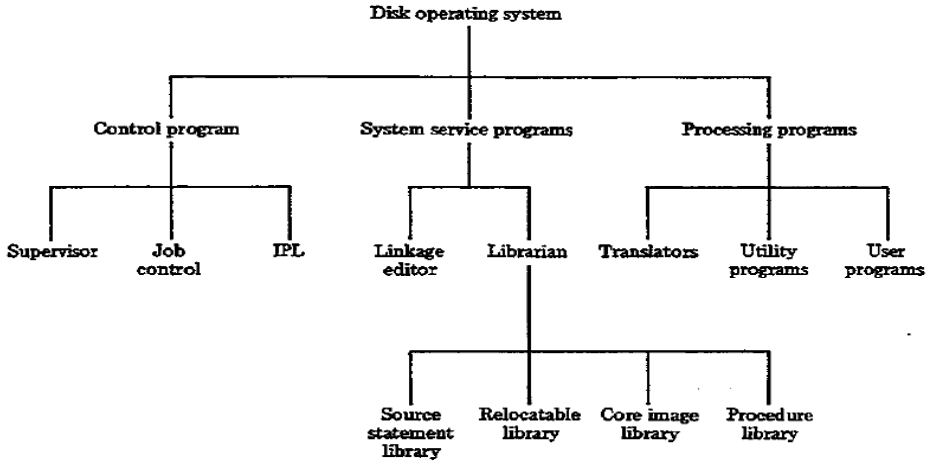


Figure 29–1 Disk Operating System Organization

Control Program
The control program, which controls all other programs being processed, consists of initial program load (**IPL**), the supervisor, and job control. Under OS, the functions are task management, data management, and job management.

IPL is a program that the operator uses daily or whenever required to load the supervisor into storage. On some systems, this process is known as booting the system.

Job control handles the transition between jobs run on the system. Your job commands tell the system what action to perform next.

The supervisor, the nucleus of the operating system, resides in lower storage, beginning at location X'200'. The system loads user (problem) programs in storage following the supervisor area, resulting in at least two programs in storage: the supervisor program and one or more problem programs. Only one is executing at any time, but control passes between them.

The supervisor is concerned with handling interrupts for input/output devices, fetching required modules from the program library, and handling errors in program execution. An important part of the supervisor is the input/output control system (IOCS), known under OS as data management.

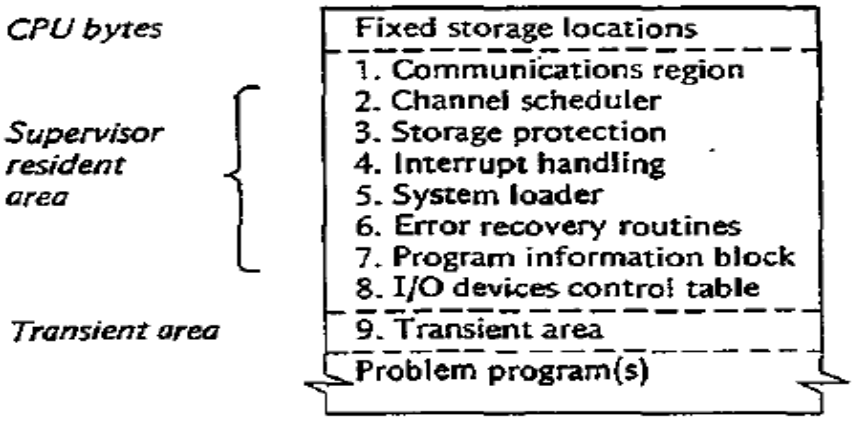


Figure 29–2 Supervisor areas

Figure 29–2 (not an exact representation) illustrates the general layout of the supervisor in main storage. Let's examine its contents.

1. Communication Region. This area contains the following data:

LOCATION	CONTENTS
00 – 07	The current date, as mm/dd/yy or dd/mm/yy
08 – 11	Reserved
12 – 22	User area, set to zero when a JOB command is read to provide communication within a job step or between job steps
23	User program status indicator (UPSI)
24 – 31	Job name, entered from job control
32 – 35	Address: highest byte of problem program area
36 – 39	Address: highest byte of current problem program phase
40 – 43	Address: highest byte of phase with highest ending address
44 – 45	Length of label area for problem program

2. Channel Scheduler. The channels provide a path between main storage and the input/output devices for all I/O interrupts and permit overlapping of program execution with I/O operations. If the requested channel, control unit, and device are available, the channel operation begins. If they are busy, the channel scheduler places its request in a queue and waits until the device is available. The channel notifies the scheduler when the I/O operation is complete or that an error has occurred.

3. Storage Protection. Storage protection prevents a problem program from erroneously moving data into the supervisor area and destroying it. Under a multiprogramming system, this feature also prevents a program in one partition from erasing a program in another partition.

4. Interrupt Handling. An interrupt is a signal that informs the system to interrupt the program that is currently executing and to transfer control to the appropriate supervisor routine. A later section on the program status word covers this topic in detail.

5. ***System Loader.*** The system loader is responsible for loading programs into main storage for execution.

6. Error Recovery Routines. A special routine hancl1es error recovery for each I/O device or class of devices. When an error is sensed, the channel scheduler invokes the required routine, which attempts to correct the error.

7. Program Information Block (PIB). The PIB contains information tables that the supervisor needs to know about the current programs in storage.

8. ***I/O Devices Control Table***. This area contains a table of I/O devices that relate physical unit addresses (X'nnn') with logical addresses (SYSxxx).

9. Transient Area. This area provides temporary storage for less used routines that the supervisor loads as required, such as OPEN, CLOSE, DUMP, end–of– job handling, some error recovery, and checkpoint routines.

System Service Programs
System service programs include the linkage editor and the librarian.

Linkage editor. The linkage editor has two main functions:

1. To include input/output modules. An installation catalogs I/O modules in the system library (covered next). When you code and assemble a program, it does not yet contain the complete instructions for handling input/output. On completion of assembly, the linkage editor includes all the required I/O modules from the library.

2. To link together separately assembled programs. You may code and assemble a number of subprograms separately and link-edit these subprograms into one executable program. The linkage editor enables data in one subprogram to be recognized in another and facilitates transfer of control between subprograms at execution time.

Librarian. The operating system contains libraries on a disk known as SYSRES to catalog both IBM programs and the installation's own commonly used programs and subroutines. DOS/VS supports four libraries:

1. The source statement library (SSL) catalogs as a book any program, macro, or subroutine still in source code. You can use the assembler directive COPY to include cataloged code into your source program for subsequent assembling.

2. The relocatable library (RL) catalogs frequently used modules that are assembled but not yet ready for execution. The assembler directs the linkage editor to include I/O modules automatically, and you can use the INCLUDE command to direct the linkage editor to include your own cataloged modules with your own assembled programs.

3. The core image library (CIL) contains phases in executable machine code, ready for execution. The CIL contains; for example, the assembler, COBOL, PL/I, and other translator programs, various utility programs such as LINK and SORT, and your own production programs ready for execution. To request the supervisor to load a phase from the CIL into main storage for execution, use the job control command
// EXEC phasename.

4. The procedure library (PL) contains cataloged job control to facilitate automatic processing of jobs.

The OS libraries vary by name according to the version of OS, but basically the OS libraries equivalent to the DOS source statement, relocatable, and core image are, respectively, source library, object library, and load library, and they serve the same functions.

Processing Programs
Processing programs are cataloged on disk in three groups:

1. Language translators that IBM supplies with the system include assembler, PL/I, COBOL, and RPG.

2. Utility programs that IBM supplies include such special-purpose programs as disk initialization, copy file–to–file, and sort/merge.

3. User–written programs that users in the installation write and that IBM does not support. All the programs in this text are user–written programs. For example, the job command // EXEC ASSEMBLY causes the system to load the assembler from the CIL into an available area ("partition") in storage and begins assembling a program. The job command // OPTION LINK directs the assembler to write the assembled module on SYSLNK in the relocatable library.

Once the program is assembled and stored on SYSLNK, the job command // EXEC LNKEDT tells the linkage editor to load the module from SYSLNK into storage, to complete addressing, and to include I/O modules from the RL. Assuming that there was no job command to catalog it, the linkage editor writes the linked phase in the CIL in a non–catalog area. If the next job command is // EXEC with no specified phase name, the supervisor loads the phase from the non–catalog area into storage for execution. The next program that the linkage editor links overlays the previous one in the CIL non–catalog area.

The job command // OPTION CATAL instead of // OPTION LINK tells the system both to link the program and to catalog the linked phase in the catalog area of the CIL. You normally catalog production programs in the CIL and for immediate execution use the job command // EXEC phase name.

MULTIPROGRAMMING

Multiprogramming is the concurrent execution of more than one program in storage. Technically, a computer executes only one instruction at a time, but because of the fast speed of the processor and the relative slowness of I/O devices, the computer's ability to service a number of programs at the same time makes it appear that processing is simultaneous. For this purpose, an operating system that supports multiprogramming divides storage into various ***partitions*** and is consequently far more complex than a single–job system.

The number and size of partitions vary according to the requirements of an installation. One job in each partition may be subject to execution at the same time, although only one program is actually executing. Each partition may handle jobs of a particular nature. For example, one partition handles relatively short jobs of high priority, whereas another partition handles large jobs of lower priority.

The job scheduler routes jobs to a particular partition according to its class. Thus a system may assign class A to certain jobs, to be run in the first partition.

In Fig. 29–4, the job queue is divided into four classes, and main storage is divided into three user partitions. Jobs in class A run in partition 1, jobs in classes B and C run in partition 2, and jobs in class P run in partition 3.

Depending on the system, storage may be divided into many partitions, and a job class may be designated to run in anyone of the partitions. Also, a partition may be designated to run any number of classes.

When an operator uses the IPL procedure to boot the system, the supervisor is loaded from the CIL into low storage. The supervisor next loads job control from the CIL into the various partitions. The supervisor then scans the system readers and terminals for job control commands.

FIXED STORAGE LOCATIONS

As mentioned earlier, the first X'200' (decimal 512)bytes of storage are reserved for use by the CPU. Figure 29–3 lists the contents of these fixed storage locations.

AREA, dec.	Hex addr	EC only	Function
0– 7	0		Initial program loading PSW, restart new PSW
8– 15	8		Initial program loading CCW1, restart old PSW
16– 23	10		Initial program loading CCW2
24– 31	18		External old PSW
32– 39	20		Supervisor Call old PSW
40– 47	28		Program old PSW
48– 55	30		Machine-check old PSW
56– 63	38		Input/output old PSW
64– 71	40		Channel status word (see diagram)
72– 75	48		Channel address word (0–3 key, 4–7 zeros, 8–31 CCW address)
80– 83	50		Interval timer
88– 95	58		External new PSW
96–103	60		Supervisor Call new PSW
104–111	68		Program new PSW
112–119	70		Machine-check new PSW
120–127	78		Input/output new PSW
132–133	84		CPU address assoc'd with external interruption, or unchanged
132–133	84	X	CPU address assoc'd with external interruption, or zeros
134–135	86	X	External interruption code
136–139	88	X	SVC interruption (0–12 zeros, 13–14 ILC, 15:0, 16–31 code)
140–143	8C	X	Program interrupt (0–12 zeros, 13–14 ILC, 15:0, 16–31 code)
144–147	90	X	Translation exception address (0–7 zeros, 8–31 address)
148–149	94		Monitor class (0–7 zeros, 8–15 class number)
150–151	96	X	PER interruption code (0–3 code, 4–15 zeros)
152–155	98	X	PER address (0–7 zeros, 8–31 address)
156–159	9C		Monitor code (0–7 zeros, 8–31 monitor code)
168–171	A8		Channel ID (0–3 type, 4–15 model, 16–31 max. IOEL length)
172–175	AC		I/O extended logout address (0–7 unused, 8–31 address)
176–179	B0		Limited channel logout (see diagram)
185–187	B9	X	I/O address (0–7 zeros, 8–23 address)
216–223	D8		CPU timer save area
224–231	E0		Clock comparator save area
232–239	E8		Machine-check interruption code
248–251	F8		Failing processor storage address (0–7 zeros, 8–31 address)
252–255	FC		Region code*
256–351	100		Fixed logout area*
352–383	160		Floating-point register save area
384–447	180		General register save area
448–511	1C0		Control register save area
512†	200		CPU extended logout area (size varies)

*May vary among models: see system library manuals for specific model

†Location may be changed by programming (bits 8–28 of CR 15 specify address)

Figure 29–3 Fixed Storage Locations

When a job completes processing, the job scheduler selects another job from the queue to replace it. For example, if partition 1 is free, the job scheduler in Fig. 29–4 selects from the class A queue either the job with the highest priority or, if all jobs have the same priority, the first job in the queue.

The system has to provide a more or less equitable arrangement for processing jobs in each partition. Under time slicing, each partition is allotted in turn a time slice of so many milliseconds of execution. Control passes to the next partition when the time has expired, the job is waiting for an I/O operation to complete, or the job is finished.

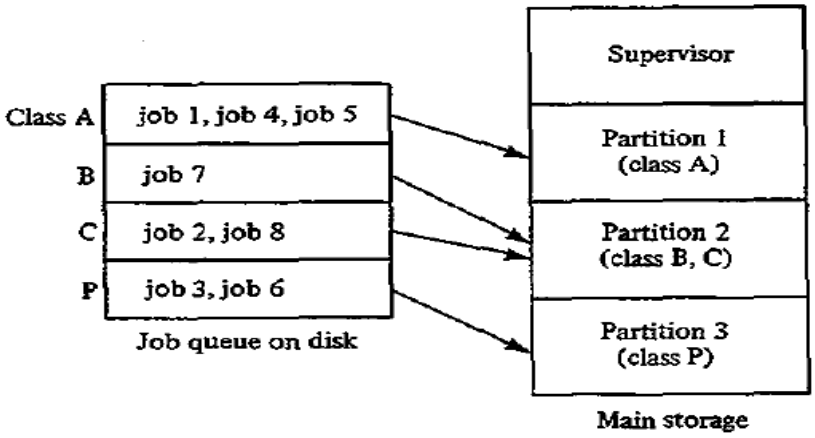


Figure 29–4 Job queue and partitions

VIRTUAL STORAGE

In a multiprogramming environment, a large program may not fit entirely in a partition. As a consequence, both *DOS/VS* and *OS/VS* support a virtual storage system that divides programs into segments of 64K bytes, which are in turn divided into pages of 2K or (usually) 4K bytes. On disk, the entire program is contained as pages in a page data set, and in storage VS arranges a page pool for as much of the program as it can store, as shown in Fig. 29–5. As a consequence, a program that is 100K in size could run in a 64K partition. If the executing program references an address for a part of the program that is not in storage, VS swaps an unneeded page into the page data set on disk and pages in the required page from disk into the page pool in storage. (Actually, VS swaps onto disk only if the program has not changed the contents of the page.) The 16 control registers handle much of the paging operations. Since a page from disk may map into any page in the pool, VS has to change addresses; this process is known as dynamic address translation (DAT).

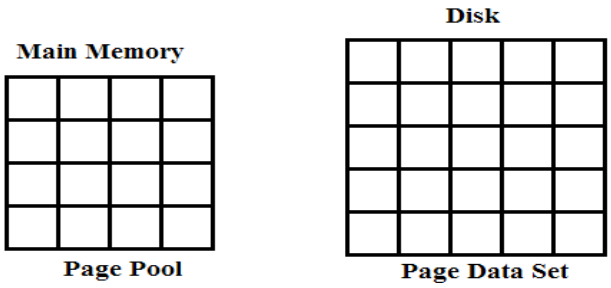


Figure 29–5: Page Pool

When running a real–time application such as process control, a data communications manager, or an optical scan device, you may not want VS to page it out. It is possible to assign an area of nonpageable (real) storage for such jobs or use a "page fix" to lock certain pages into real storage.

PROGRAM STATUS WORD: PSW

The PSW is a doubleword of data stored in the control section of the CPU to control an executing program and to indicate its status. The two PSW modes are *basic control (BC) mode* and *extended control (EC) mode.* A 0 in PSW bit 12 indicates BC mode, and a 1 indicates EC mode. EC mode provides an extended control facility for virtual storage. One of the main features of the PSW is to control the state of operation.

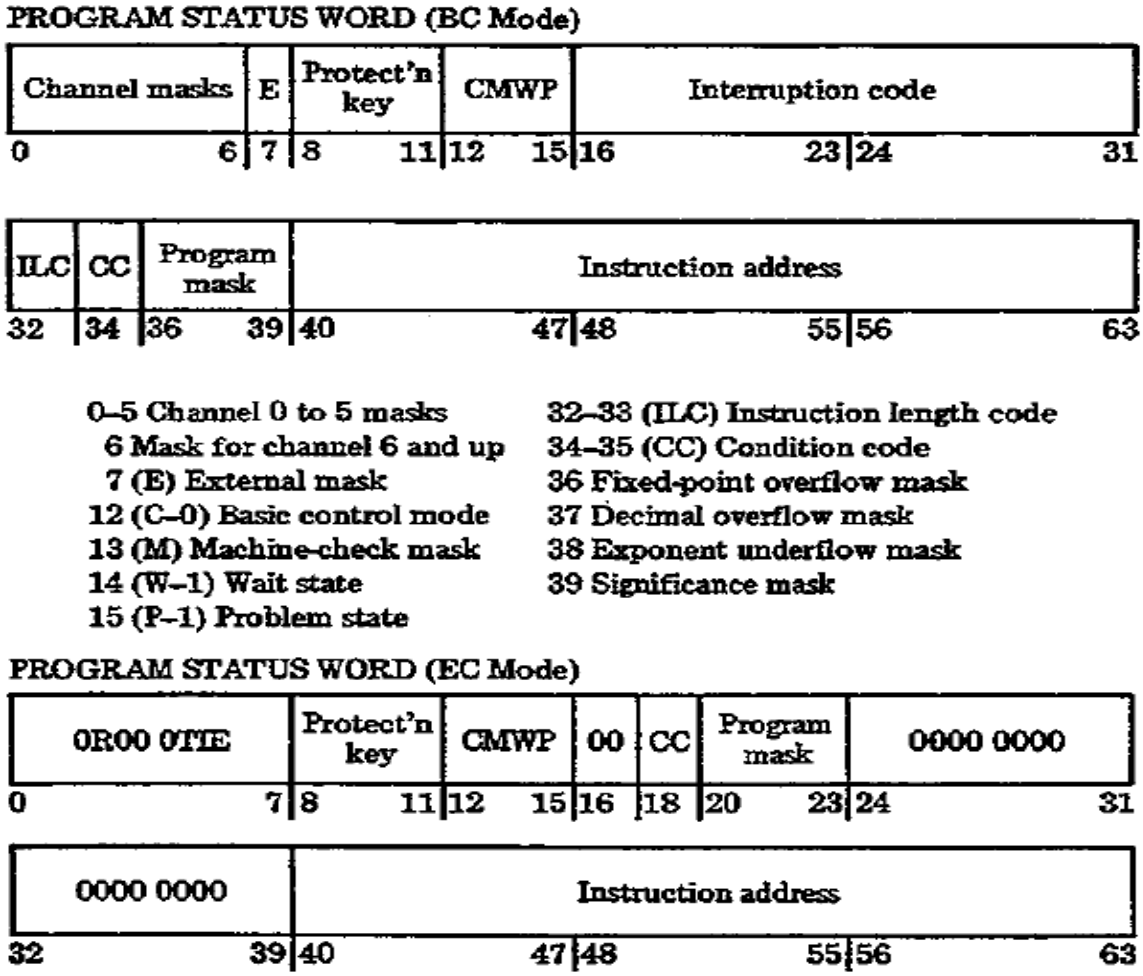


Figure 29–6: Two Variants of the PSW (Program Status Word)

[Note by ELB: In the original S/360 design, bit 12 of the PSW was the ASCII bit, with settings to allow running the computer with either EBCDIC characters ($PSW_{12} = 0$) or ASCII characters $PSW_{12} = 1$.) The ASCII option was so little used that most system managers were completely unaware that it existed. When the S/370 design with support for virtual memory was introduced, the designers needed a bit to indicate what was essentially "S/370 mode" rather than the older "S/360 mode". Bit 12 was reassigned.]

Users of the system have no concern with certain operations such as storage management and allocation of I/O devices, and if they were allowed access to every instruction, they could inadvertently access other users' partitions or damage the system. To provide protection, certain instructions, such as Start I/O and Load PSW, are designated as privileged.

The PSW format is the same in only certain positions for .each mode. Figure 29–6, just above, illustrates the two modes, in which the bits are numbered 0 through 63 from left to right. Some of the more relevant fields are explained next.

Bit 14: Wait state. When bit 14 is 0, the CPU is in the running state and executing instructions. When bit 14 is 1, the CPU is in wait state; which involves waiting for an action such as an I/O operation to be completed.

Bit 15: State. For both modes, 0 = supervisor state and 1 = problem state. When the computer is executing the supervisor program, the bit is 0 and all instructions are valid. When in the problem state, the bit is 1 and privileged instructions cannot be executed.

Bits 16–31: Program interrupt code (BC mode). When a program interrupt occurs, the computer sets these bits according to the type. The following list shows the interrupt codes in hex format:

0001 Operation exception
0002 Privileged operation exception
0003 Execute exception
0004 Protection exception
0005 Addressing exception
0006 Specification exception
0007 Data exception
0008 Fixed–point overflow exception
0009 Fixed–point divide exception
000A Decimal overflow exception
000B Decimal divide exception
000C Exponent overflow exception
000D Exponent underflow exception
000E Significance exception
000F Floating–point divide exception
0010 Segment translation exception
0011 Page translation exception
0012 Translation specification exception
0013 Special operation exception
0040 Monitor event
0080 Program event (may be combined with another code)

Bits 34–35: Condition code. BC mode only; the condition code under EC mode is in bits 18–19. Comparisons and certain arithmetic instructions set this code.

Bits 40–63: Instruction address [Often called the PC, or Program Counter]. This area contains the address of the next instruction to be executed. The CPU accesses the specified instruction from main storage, decodes it in the control section, and executes it in the arithmetic/logic section. The first 2 bits of a machine instruction indicate its length. The CPU adds this length to the instruction address in the PSW, which now indicates the address of the next instruction. For a branch instruction, the branch address may replace the PSW instruction address.

INTERRUPTS

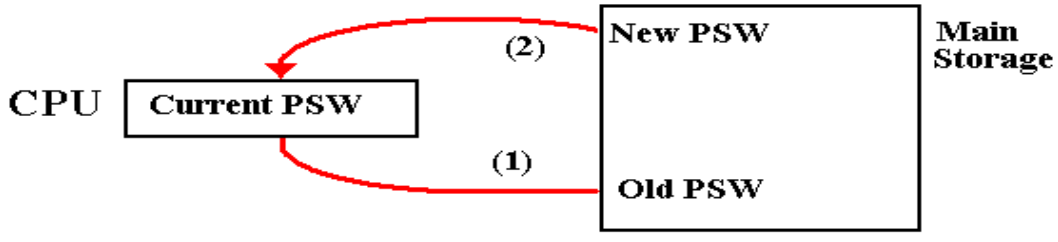
An interrupt occurs when the supervisor has to suspend normal processing to

perform a special task. The six main classes of interrupts are as follows:

1. *Program Check Interrupt.* This interrupt occurs when the computer cannot execute an operation, such as performing arithmetic on invalid packed data.
   This is the common type of interrupt when a program terminates abnormally.
2. *Supervisor Call Interrupt.* A problem program may issue a request for input/output or to terminate processing. A transfer from the problem program to the supervisor requires a supervisor call (SVC) operation and causes an interrupt.
3. *External Interrupt.* An external device may need attention, such as the operator pressing the request key on the console or a request for communications.
4. *Machine Check Interrupt.* The machine–checking circuits may detect a hardware error, such as a byte not containing an odd number of on bits (odd parity). [Note by ELB: This refers to parity memory, in which an 8–bit byte is stored as 9 bits in memory, with the extra bit (not transferred to the CPU) being the parity bit.]
5. *Input/Output Interrupt.* Completion of an I/O operation making a unit available or malfunction of an I/O device (such as a disk head crash) cause this interrupt.
6. *Restart Interrupt.* This interrupt permits an operator or another CPU to invoke execution of a program.

The supervisor region contains an interrupt handler for each type of interrupt. On an interrupt, the system alters the PSW as required and stores the PSW in a fixed storage location, where it is available to any program for testing.

The PSW discussed to this point is known as the current PSW. When an interrupt occurs, the computer stores the current PSW and loads a new PSW that controls the new program, usually the supervisor. The current PSW is in the control section of the CPU, whereas the old and new PSWs are stored in main storage, as the following indicates:



The interrupt replaces the current PSW in this way. (1) It stores the current PSW into main storage as the old PSW, and (2) it fetches a new PSW from main storage, to become the current PSW. The old PSW now contains in its instruction address the location following the instruction that caused the interrupt. The computer stores the Program Status Words in 12 doubleword locations in fixed storage; 6 are for old PSWs and 6 are for new PSWs, depending on the class of interrupt. There are eight bytes allocated for each PSW; for this reason the following addresses appear to be decimal numbers.

Interrupt Type	Old PSW	New PSW
Restart	0008	0000
External	0024	0088
Supervisor Call	0032	0096
Program Old PSW	0040	0104
Machine Check	0048	0112
Input/Output	0056	0120

Let's trace the sequence of events following a supervisor interrupt. Assume that the supervisor has stored the address of each of its interrupt routines as bits 40–63 of the PSW that is stored in the address associated with its interrupt type. Loading the CPU Program Status Word with the "New PSW" associated with an interrupt type essentially starts the interrupt handler on the next instruction.

Remember also that when an instruction executes, the computer updates the instruction address and the condition code in the current PSW (in the CPU) as required.

1. A program requests input from disk. The GET or READ macro contains a SVC (**S**uper**v**isor **C**all) to link to the supervisor [ELB: a part of the Operating System] for input/output. This is a supervisor interrupt.

2. The instruction address in the current PSW contains the address in the program immediately following the SVC that caused the interrupt. The CPU stores this current PSW in the old PSW for supervisor interrupt, location 32.

   The new PSW for supervisor interrupt, location 96, contains supervisor state bit = 0 and the address of the supervisor interrupt routine. The CPU moves this new PSW to the current PSW and is now in the supervisor state.

3. The PSW instruction address contains the address of the supervisor I/O routine, which now executes. The channel scheduler requests the channel for disk input.

4. To return to the problem program, the supervisor loads the old PSW from location 32 back into the current PSW. The instruction links to the PSW instruction address, which is the address in the program following the original SVC that caused the interrupt. The system switches the PSW from supervisor state back to problem state.

[ELB Note: This design reflects some older strategies that had yet to take full advantage of dynamic memory organizations, based on use of the stack and heap.]

In the event of a program check interrupt, the computer sets its cause on PSW bits 16-31, the program interrupt code. For example, if the problem program attempts arithmetic on invalid data, the computer senses a data exception and stores X'0007' in PSW bits 16-31. The computer then stores the current PSW in old PSW location 0040 and loads the new PSW from 0104 into the current PSW. This PSW contains the address of the supervisor's program check routine, which tests the old PSW to determine what type of program check caused the interrupt.

The supervisor displays the contents of the old PSW in hexadecimal and the cause of the program check (data exception), flushes the interrupted program, and begins processing the next job. Suppose that the invalid operation is an MP [Multiply Packed] at location X'6A320'. Since MP is 6 bytes long, the instruction address in the PSW and the one printed will be X'6A326'. You can tell from the supervisor diagnostic message that the error is a data exception and that the invalid operation immediately precedes the instruction at X'6A326'.

CHANNELS

A channel is a component that functions as a separate computer operated by channel commands to control I/O devices. It directs data between devices and main storage and permits attaching a great variety of I/O devices. The more powerful the computer model, the more channels it may support. The two types of channels are multiplexer and selector.

1. ***Multiplexer channels*** are designed to support simultaneous operation of more than one device by interleaving blocks of data. The two types of multiplexer channels are byte-multiplexer and block-multiplexer. A byte-multiplexer channel typically handles low-speed devices, such as printers and terminals.

A block-multiplexer can support higher-speed devices, and its ability to interleave blocks of data facilitates simultaneous I/O operations.

2. ***Selector channels***, no longer common, are designed to handle high–speed devices, such as disk and tape drives. The channel can transfer data from only one device at a time, a process known as burst mode.

Each channel has a 4–bit address coded as in the following example:

CHANNEL	ADDRESS	TYPE
0	0000	byte-multiplexer
1	0001	block-multiplexer
2	0010	block-multiplexer
3	0011	block-multiplexer
4	0100	block-multiplexer
5	0101	block-multiplexer
6	0110	block-multiplexer

A ***control unit***, or controller, is required to interface with a channel. A channel is basically device–independent, whereas a control unit is device–dependent.

Thus a block-multiplexer channel can operate many type of devices, but a disk drive control unit can operate only a disk drive. Figure 29–7 illustrates a typical configuration of channels, control units, and devices.

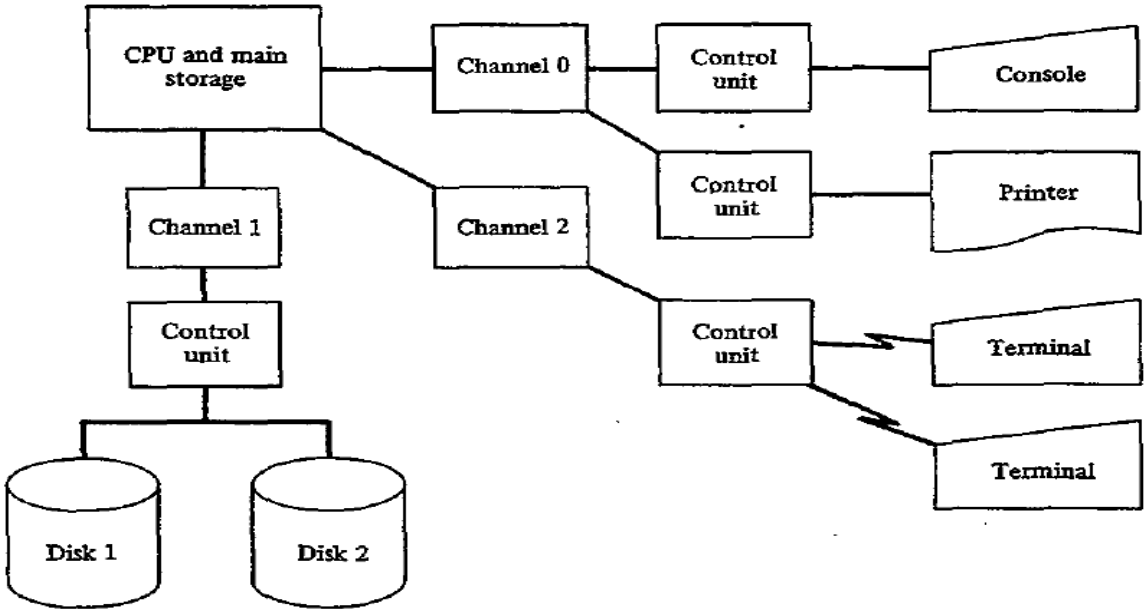


Figure 29–7: Channels, Control Units, and Devices

For example, a computer uses a multiplexer channel to connect it to a printer's control unit. The control unit has a 4–bit address. Further, each device has a 4–bit address and is known to the system by a physical address. The device address is therefore a 12–bit code that specifies:

DEVICE	**CODE**
Channel	**0CCC**
Control unit	**UUUU**
Device	**DDDD**

If the printer's device number is 1110 (X'E') and it is attached to channel 0, control unit 1, then to the system its physical address is 0000 00011110, or X'01E'. Further, if two disk devices are numbered 0000 and 0001 and they are both attached to channel 1, control unit 9, their physical addresses are X'190' and X'191', respectively.

This physical address permits the attaching of 2^8 , or 256 devices.

Symbolic Assignments

Although the supervisor references *IJO* devices by their physical numbers, your programs use symbolic names. You may assign a symbolic name to any device temporarily or (more or less) permanently, and a device may have more than one symbolic name assigned. The operating system uses certain names, known as system logical units, that include the following.

In addition, you may reference programmer logical units, SYS000-SYSnnn.

SYSIPT
The terminal, system reader, or disk device used as input for programs

SYSRDR
The terminal, system reader, or disk device used as input for job control for the system

SYSIN
The system name to assign both SYSIPT and SYSRDR to the same terminal, system reader, or disk device

SYSLST
The printer or disk used as the main output device for the system

SYSPCH
The device used as the main unit for output

SYSOUT
The system name to assign both SYSLST and SYSPCH to the same output device

SYSLNK
The disk area used as input for the linkage editor

SYSLOG
The console or printer used by the system to log operator messages and job control statements

SYSRES
The disk device where the operating system resides

SYSRLB
The disk device for the relocatable library

SYSSLB
The disk device for the system library

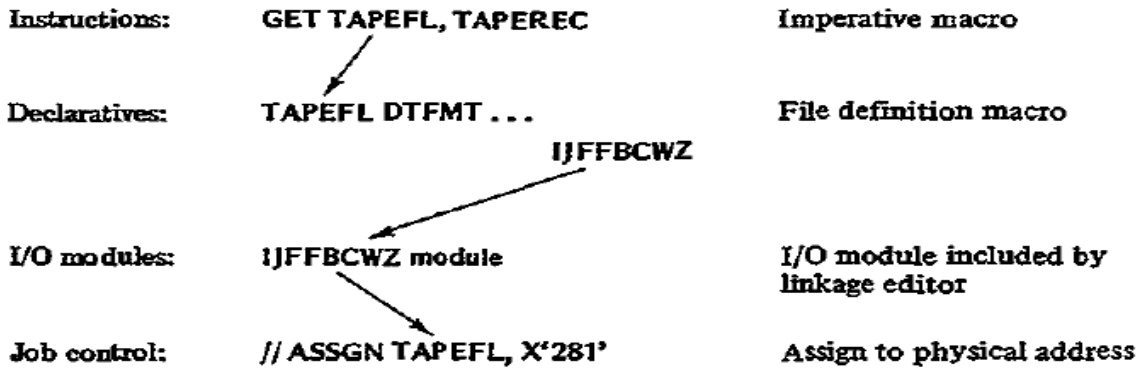
For example, you may assign the logical address SYS025 to a disk drive with physical address X'170'. The supervisor stores the physical and logical addresses in an I/O devices control table in order to relate them. A simplified table could contain the following:

I/O Device	Physical Address	Logical Units
Reader	X‘00C’	SYSIPT, SYSRDR
Printer	X‘00E’	SYSLST
Disk Drive	X‘170’	SYSLNK, SYSRES, SYS025
Tape Drive	X‘280’	SYS031, SYS035

A reference to SYSLST is to the printer, and a reference to SYSLNK, SYSRES, or SYS025, depending on its particular use, is to disk device X'170'. You may assign a logical address permanently or temporarily and may change logical addresses from job to job. For instance, you could use an ASSGN job control command to reassign SYS035 for a program from a disk device X'170' to another disk device X'l72'.

I/O LOGIC MODULES

Consider a program that reads a tape file named TAPEFL. The program would require a DTFMT or DCB file definition macro to define the characteristics of the file and tape device to generate a link to an I/O logic module. The assembler determines which particular logic module, based on (1) the kind of DTF and (2) the specifications within the file definition, such as device number, an input or output file, the number of buffers, and whether processing is in a work area (WORKA) or a buffer (IOREG). In the following example, the assembler has generated a logic module named DFFBCWZ (the name would vary depending on specifications within the DTFMT).



When linking a program, the linkage editor searches for addresses in the external symbol dictionary that the assembler generates. For this example, the ESD would contain entries at least for the program name and UFFBCWZ. The linker accesses the named module cataloged on disk (provided it was ever cataloged) and includes it at the end of the assembled object program. One role of a system programmer is to define and catalog these I/O modules.

On execution of the program, the GET macro links to the specified file definition macro, DTFMT. This macro contains the address of the I/O logic module at the end of the object program where the linker included it. The module, combined with information from the DTFMT, contains all the instructions necessary to notify the supervisor as to the actual type of I/O operation, device, block size, and so forth.

The only remaining information is to determine which tape device; the supervisor derives it from the job control entry, which in this example assigns X'281' as the physical address. The supervisor then (at last) delivers the physical request for input via a channel command.

For example, the printer module, PRMOD, consists of three letters (IJD) and five option letters (abcde), as IJDabcde. The options are based on the definitions in the DTFPR macro, as follows:

a RECFORM: FlXUNB (F), VARUNB (V), UNDEF (U)
b CILCHR: ASA (A), YES (Y), CONTROL (C)
c PRINTOV=YES and ERROPT=YES (B), PRINTOV=YES and ERROPT not specified (Z), plus 14 other options
d IOAREA2: defined (I), not defined (Z)
e WORKA: YES (W), YES and RDONLY= YES (V), neither specified (Z)

A common printer module for IBM control character; two buffers, and a work area would be IJDFYZIW. For one buffer, the module is IJDFYZZW.

PHYSICAL IOCS

Physical I0CS (PIOCS), the basic level of I0CS, provides for channel scheduling, error recovery, and interrupt handling.. When using PIOCS, you write a channel program (the channel command word) and synchronize the program with completion of the I/O operation. You must also provide for testing the command control block for certain errors, for checking wrong–length records, for switching between I/O areas where two are used, and, if records are blocked, for blocking and deblocking.

PIOCS macros include CCW, CCB, EXCP, and WAIT.

Channel Command Word (CCW)

The CCW macro causes the assembler to construct an 8–byte channel command word that defines the I/O command to be executed.

Name	Operation	Operand
[LABEL]	CCW	command–code, data–address, flags, count–field

- command-code defines the operation to be performed, such as 1 = write, 2 = read, X'09' = print and space one line.
- data-address provides the storage address of the first byte where data is to be read or written.
- flag bits determine the next action when the channel completes an operation defined in a CCW. You can set flag bits to 1 to vary the channel's operation (explained in detail later).
- count-field provides an expression that defines the number of bytes in the data block that is to be processed.

Command Control Block (CCB)

You define a CCB macro for each I/O device that PIOCS macros reference. The CCB comprises the first 16 bytes of most generated DTF tables. The CCB communicates information to PIOCS to cause required I/O operations and receives status information after the operation.

Name	Operation	Operand
[LABEL]	CCB	SYSnnn, command–list–name

- blockname is the symbolic name associated with the CCB, used as an old PSW for the EXCP and WAIT macros.
- SYSnnn is the symbolic name of the 110 device associated with the CCB.
- command-list-name is the symbolic name of the first CCW used with the CCB.

Execute Channel Program (EXCP)

The EXCP macro requests physical I0CS to start an I/O operation, and PIOCS relates the block name to the CCB to determine the device. When the channel and the device become available, the channel program is started. Program control then returns to your program.

Name	Operation	Operand
[LABEL]	EXCP	block–name or (1)

The operand gives the symbolic name of the CCB macro to be referenced.

The WAIT Macro

The WAlT macro synchronizes program execution with completion of an I/O operation, since the program normally requires its completion before it can continue execution. (When bit 0 of byte 2 of the CCB for the file is set to 1, the WAlT is completed and processing resumes.) For example, if you have issued an EXCP operation to read a data block, you now WAlT for delivery of the entire block before you can begin processing it.

Name	Operation	Operand
[LABEL]	WAIT	block–name or (1)

CCW Flag Bits

You may set and use the flag bits in the CCW as follows:

• Bit 32 (chain data flag), set by X'80', specifies ***data chaining***. When the CCW has processed the number of bytes defined in its count field, the I/O operation does not terminate if this bit is set. The operation continues with the next CCW in storage. You may use data chaining to read or write data into or out of storage areas that are not necessarily adjacent.

In the following three CCWs, the first two use X'80' in the flag bits, operand 3, to specify data chaining. An EXCP and CCB may then reference the first CCW, and as a result, the chain of three CCWs causes the contents of an 80–byte input record to be read into three separate areas in storage: 20 bytes in NAME, 30 bytes in ADDRESS, and 30 bytes in CITY.

```
DATCHAIN CCW 2,NAME,X'80',20     Read 20 bytes into NAME,
                                 and chain.
         CCW  ,ADDRESS,X'80',30  Read 30 bytes to ADDRESS,
                                 and chain.
         CCW  ,CITY,X'00',30     Read 30 bytes into CITY,
                                 and terminate.
```

• Bit 33 (chain command flag), set by X'40', specifies ***command chaining*** to enable the channel to execute more than one CCW before terminating the I/O operation.
Each CCW applies to a separate I/O record.

The following set of Channel Command Words could provide for reading three input blocks, each 100 bytes long:

```
COMCHAIN  CCW 2,INAREA,X'40',100       Read record-l into
                                       INAREA, chain.
          CCW 2,INAREA+100,X'40',100   Read record-2 into
                                       INAREA+l00, chain.
          CCW 2,INAREA+200,X'00',100   Read record-3 into
                                       INAREA+200, stop.
```

• Bit 34 (suppress length indication flag), set by X'20', is used to suppress an error indication that occurs when the number of bytes transmitted differs from the count in the CCW.

• Bit 35 (skip flag), set by X'10', is used to suppress transmission of input data. The device actually reads the data, but the channel does not transmit the record.

• Bit 36 (program controlled interrupt flag), set by X'08', causes an interrupt when this CCW's operation is complete. (This is used when one supervisor SIO instruction executes more than one CCW.)

• Bit 37 (indirect data address flag), as well as other features about physical IOCS, is covered in the IBM Principles of Operation manual and the appropriate supervisor manual for your system.

Sample Physical IOCS Program

The program in Fig. 29–8 illustrates many of the features of physical IOCS we have discussed. It performs the following operations:

• At initialization, prints three heading lines by means of command chaining (X'40').

• Reads input records one at a time containing salesman name and company.

• Prints each record.

• Terminates on reaching end–of–file.

Note that the program defines a *CCB/CCW* pair for each type of record, and the EXCP/WAIT operations reference the CCB name – INDEVIC for the reader, OUTDEV1 for heading lines, and OUTDEV2 for sales detail lines. Each CCB contains the name of the I/O device, SYSIPT or SYSLST, and the name of an associated CCW: INRECD, TITLES, and DETAIL, respectively.

```
  LOC   OBJECT CODE      STMT   SOURCE STATEMENT
                            1          PRINT NODATA,NOGEN
000000                      2 PIOCSPRG START 0                INITIALIZE
000000 0530                 3          BALR  3,0              *
                            4          USING *,3              *

                            6          EXCP  OUTDEV1          PRINT TITLES
                           10          WAIT  OUTDEV1          *

                           17 A100READ EXCP  INDEVIC          READ RECORD
                           21          WAIT  INDEVIC          *

00002A D501 3076 3332      28          CLC   RECORD(2),=C'/*' END FILE?
000030 4780 305C           29          BE    A900END            YES
000034 D213 32B8 3076      31          MVC   SURNOUT,SURNAME  LOAD
00003A D213 32CD 308A      32          MVC   GIVENOUT,GIVENAME * PRINT
000040 D21D 32E2 309E      33          MVC   COMPOUT,COMPANY  * LINE

                           35          EXCP  OUTDEV2          PRINT
                           39          WAIT  OUTDEV2          *
00005A 47F0 3014           45          B     A100READ         RETURN

                           47 A900END  EOJ                    END OF JOB

                           51 *        ----------------------
                           52 *        D E C L A R A T I V E S
                           53 *        ----------------------
                           54 INDEVIC  CCB   SYSIPT,INRECRD   I/P DEVICE
000070 0200007820000050    65 INRECRD  CCW   X'02',RECORD,X'20',80

000078                     67 RECORD   DS    0CL80            I/P RECORD
000078                     68 SURNAME  DS    CL20             *
00008C                     69 GIVENAME DS    CL20             *
0000A0                     70 COMPANY  DS    CL40             *
```

```
                          72 OUTDEV1  CCB   SYSLST,TITLES     O/P DEVICE
0000D8 8B0000D860000001   84 TITLES   CCW   X'8B',*,X'60',1
0000E0 110000F840000085   85           CCW   X'11',PRIMARY,X'40',133
0000E8 1900017D40000085   86           CCW   X'19',SECONDRY,X'40',133
0000F0 1100020200000085   87           CCW   X'11',TERTIARY,X'00',133

0000F8                    89 PRIMARY  DS    0CL133            TITLE #1
0000F8 4040404040404040   90           DC    CL37' '
00011D E340D640D7404040   91           DC    CL16'T O P   S A L E'
00012D E240D440C540D540   92           DC    CL80'S M E N   O F'

00017D                    94 SECONDRY DS    0CL133            TITLE #2
00017D 4040404040404040   95           DC    CL34' '
00019F E340C840C5404040   96           DC    CL14'T H E   W E S'
0001AD E340C540D940D540   97           DC    CL14'T E R N   R E'
0001BB C740C940D640D540   98           DC    CL71'G I O N'

000202                   100 TERTIARY DS    0CL133            TITLE #3
000202 4040404040404040  101           DC    CL26' '
00021C E2E4D9D5C1D4C540  102           DC    CL21'SURNAME'
000231 C7C9E5C5D540D5C1  103           DC    CL21'GIVEN NAME'
000246 C3D6D4D7C1D5E840  104           DC    CL65'COMPANY'
                         106 OUTDEV2  CCB   SYSLST,OUTRECRD   O/P DEVICE
000297 00
000298 090002A020000085  118 OUTRECRD CCW   X'09',DETAIL,X'20',133

0002A0                   120 DETAIL   DS    0CL133            DETAIL
0002A0 4040404040404040  121           DC    CL26' '           * LINE
0002BA                   122 SURNOUT  DS    CL20
0002CE 40                123           DC    CL01' '
0002CF                   124 GIVENOUT DS    CL20
0002E3 40                125           DC    CL01' '
0002E4                   126 COMPOUT  DS    CL30
000302 4040404040404040  127           DC    CL35' '

000328                   129           LTORG ,
000328 000000C8          130                 =A(OUTDEV1)
00032C 00000060          131                 =A(INDEVIC)
000330 00000287          132                 =A(OUTDEV2)
000334 615C              133                 =C'/*'
                         134           END   PIOCSPRG
```

Output:-

```
                T O P   S A L E S M E N   O F

            T H E   W E S T E R N   R E G I O N

SURNAME              GIVEN NAME           COMPANY

RUTH                 GEORGE HERMAN        LASER CORP.
JOHNSON              WALTER               AMX ELECTRONICS
COLLINS              EDDIE                B M I
COBB                 TYRUS RAYMOND        AUDIO SHACK
SPEAKER              TRIS                 PACKLETT HEWARD
SIMMONS              AL                   VIDEO DUMP
SISLER               GEORGE               COMPUTER HEAP
WAGNER               HANS                 DIGITAL CORP.
```

Figure 29–8: Physical IOCS

KEY POINTS

- Systems generation (sysgen) involves tailoring the supplied operating system to the installation's requirements, such as the number and type of disk drives, the number and type of terminals to be supported, the amount of process time available to users, and the levels of security that are to prevail.
- The control program, which controls all other programs being processed, consists of initial program load (IPL), the supervisor, and job control. Under OS, the functions are task management, data management, and job management.
- Initial program load (IPL) is a program that the operator uses daily or whenever required to load the supervisor into storage. The system loader is responsible for loading programs into main storage for execution.
- The supervisor resides in lower storage, beginning at location X'200'. The supervisor is concerned with handling interrupts for input/output devices, fetching required modules from the program library, and handling errors in program execution.
- Channels provide a path between main storage and the input/output devices and permit overlapping of program execution with I/O operations. The channel scheduler handles all I/O interrupts.
- Storage protection prevents a problem program from erroneously moving data into the supervisor area and destroying it.
- An interrupt is a signal that informs the system to interrupt the program that is currently executing and to transfer control to the appropriate supervisor routine.
- The source statement library (SSL) catalogs as a book any program, macro, or subroutine still in source code.
- The relocatable library (RL) catalogs frequently used modules that are assembled but not yet ready for execution.
- The core image library (CIL) contains phases in executable machine code, ready for execution.
- Multiprogramming is the concurrent execution of more than one program in storage. An operating system that supports multiprogramming divides storage into various partitions. One job in each partition may be subject to execution at the same time, although only one program is actually executing.
- The PSW is stored in the control section of the CPU to control an executing program and to indicate its status. The two PSW modes are basic control mode (BC) and extended control (EC) mode.
- Certain instructions such as Start I/O and Load PSW are privileged to provide protection against users' accessing the wrong partitions.
- An interrupt occurs when the supervisor has to suspend normal processing to perform a special task. The supervisor region contains an interrupt handler for each type of interrupt.

- A channel is a component that functions as a separate computer operated by channel commands to control I/O devices. It directs data between devices and main storage and permits the attachment of a variety of I/O devices.

  The two types are multiplexer and selector.

- The operating system uses certain names, known as system logical units, such as SYSIPT, SYSLST, and SYSLOG. Programmer logical units are referenced as SYS000-SYSnnn.

- Physical IOCS (PIOCS), the basic level of IOCS, provides for channel scheduling, error recovery, and interrupt handling. When using PIOCS, you write a channel program (the channel command word) and synchronize the program with completion of the I/O operation. .

- The CCW macro causes the assembler to construct an 8–byte channel command word that defines the I/O command to be executed.