GC20-1755-0

Systems

A Guide to the IBM System/370 Model 168

IBM

GC20-1755-0

Systems

A Guide to the IBM System/370 Model 168

This guide presents hardware, programming systems, and other pertinent information about the IBM System/370 Model 168 that describes its significant new features and advantages. Knowledge of the IBM System/370 Model 165 is assumed. Features common to Models 165 and 168 are indicated but not discussed in detail. The contents of the guide are intended to acquaint the reader with the Model 168 and to be of benefit in planning for its installation.

Associated with this guide are three optional supplements that describe operating systems for the Model 168 that support a virtual storage environment. Each supplement has its own form number and must be ordered individually, if required. Optional supplements are the following:

- *OS/Virtual Storage 1 Features Supplement* (GC20-1752)
- *OS/Virtual Storage 2 Features Supplement* (GC20-1753)
- *Virtual Machine Facility/370 Features Supplement**

*Availability to be announced

IBM

First Edition (August 1972)

This guide is intended for planning purposes only. It will be updated from time to time; however, the reader should remember that the authoritative sources of system information are the system library publications for the Model 168, its associated components and its programming support. These publications will first reflect any changes.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604. Comments become the property of IBM.

PREFACE

It is assumed that the reader of this publication is familar with System/370 Model 165 hardware features, channels, I/O devices, and programming support as described in *A Guide to the IBM System/370 Model 165*, GC20-1730, and/or system library publications concerning Model 165 hardware and programming systems support. This guide discusses in detail only the hardware features of the Model 168 that are different from those of the Model 165 and the programming support provided for new features of the Model 168. The Model 168 is not compared with a Model 165 II, which is a purchased Model 165 (storage model J, K, or KJ) with the optional Dynamic Address Translation Facility installed. However, functional descriptions of Model 168 features that are also part of the Dynamic Address Translation Facility of the Model 165 II apply to the Model 165 II as well, unless otherwise noted. This publication applies to systems with 60-cycle power.

The total Model 168 guide consists of this base publication (Sections 01 to 70), which covers virtual storage concepts and Model 168 hardware and I/O devices, and from one to three optional supplements (Sections 90 to 110). The optional supplements describe the facilities of the IBM operating systems that support a virtual storage environment using the dynamic address translation hardware of the Model 168. Each optional supplement has its own unique form number and each supplement desired must be ordered separately and inserted in this base publication, which is distributed without the automatic inclusion of any optional supplements.

The following optional supplements can be inserted in this base publication:

- OS/Virtual Storage 1 Features Supplement (GC20-1752) - assumes knowledge of OS MFT

- OS/Virtual Storage 2 Features Supplement (GC20-1753) - assumes knowledge of OS MVT

- Virtual Machine Facility/370 Features Supplement*

All optional supplements also assume knowledge of virtual storage, dynamic address translation, and other new Model 168 features as described in this base publication or appropriate system library documents. However, no optional supplement requires knowledge of the contents of any other optional supplement.

This base publication, as well as each optional supplement, begins with page 1 and includes its own table of contents and index. The base publication or supplement title is printed at the bottom of each page as a means of identification.

*Availability to be announced

The optional programming systems supplements contain System/370 model-independent information, unless otherwise noted, and are designed to be included in the guides for System/370 Models 135, 145, 158, and 168 as shown below.

	Supplements			
Base Publications	DOS/VS Features Supplement (*)	OS/VS1 Features Supplement (GC20-1752)	OS/VS2 Features Supplement (GC20-1753)	VM/370 Features Supplement (*)
A Guide to the IBM System/370 Model 135 (GC20-1738-4 or later editions)	X	X		X
A Guide to the IBM System/370 Model 145 (GC20-1734-2 or later editions)	X	X	X	X
A Guide to the IBM System/370 Model 158 (GC20-1754)	X	X	X	X
A Guide to the IBM System/370 Model 168 (GC20-1755)		X	X	X

*Availability to be announced

CONTENTS

Base Publication Sections (Sections 01 to 70)

Optional Sections (See each supplement for detailed contents and index)

Note: This guide does not have a Section 80. DOS/Virtual Storage features are discussed in the Section 80 supplement and the Model 168 is not supported by DOS/VS.

FIGURES (Sections 01 to 70)

TABLES (Sections 01 to 70)

SECTION 01: SYSTEM HIGHLIGHTS

The System/370 Model 168 is an advanced function growth system for System/360 Models 65, 67, and 75 and System/370 Models 155, 158, and 165. The Model 168 provides major new functions that are not basic to System/360 architecture. The Model 168 has new features and new programming systems support that are designed to facilitate application development and maintenance. In addition, a Model 168 and its new programming support can ease entry into, and expansion of, online data processing operations.

The Model 168 makes new functions available to Model 65, 75, and 165 users without requiring a major conversion effort as the Model 168 is upward compatible with these models. Existing System/360 operating systems that support these models, namely OS MFT and MVT, support the Model 168. However, the Model 168 has standard features that are designed to support a virtual storage environment and new versions of OS are provided that use these features.

Compatible growth from a System/360 operating system to a Model 168 virtual storage environment can be achieved using the new System/370 operating systems: OS/Virtual Storage 1 (OS/VS1) and OS/Virtual Storage 2 (OS/VS2), which are based on OS MFT and OS MVT, respectively. These operating systems will run only on System/370 models with extended System/370 functions, namely on those with extended control mode of system operation and dynamic address translation facilities. They cannot operate on System/360 models. In addition to implementing virtual storage, the System/370 operating systems offer many other new capabilities and performance-oriented enhancements that are not provided by OS MFT or MVT.

A virtual machine environment is supported by Virtual Machine Facility/370 (VM/370), the successor to CP/67 for System/370. While CP/67 is available only to Model 67 System/360 users, VM/370 operates on System/370 Models 135, 145, 155 II, 158, 165 II, and 168. Model 67 users who have CP/67 installed can use VM/370 on a Model 168 with some conversion effort.

Transition with little or no reprogramming is also provided for Model 65, 67, and 165 users who are emulating 7070-, 7080-, and 7090-series systems under OS MFT or MVT and for users with these systems installed, since the integrated emulators for 7000-series systems are also supported by OS/VS1 and OS/VS2.

Highlights of the Model 168, when compared with a Model 165, are as follows:

- A basic control (BC) mode and an extended control (EC) mode of system operation are standard. Only BC mode is provided in the Model 165. EC mode of operation provides additional system control and supports new functions that are not provided in System/360 or a Model 165.

- Internal performance of a Model 168 operating in BC mode is faster than that of a Model 165. The instruction execution rate of the Model 168 is generally in the range of 10 to 30 percent faster than that of the Model 165 when identical system configurations, identical programs, and the same operating system are used. Increased internal performance results primarily from the significantly faster cycle times of processor storage in the Model 168.

- Dynamic address translation (DAT) is a standard facility that can be made operative only when the Model 168 is in EC mode. It provides hardware translation of addresses during program execution. One virtual storage of up to 16 million bytes or multiple virtual storages of up to 16 million bytes each can be supported using DAT hardware. (The amount of virtual storage that can be efficiently supported by a Model 168 depends on the hardware configuration and job stream characteristics.) The optional channel indirect data addressing feature must be installed on 2860, 2870, and 2880 channels when dynamic address translation is used. Channel indirect data addressing enables the channels to access an I/O buffer that is contained in noncontiguous processor storage areas.

- Program event recording (PER) is standard and can be made operative when the Model 168 is in EC mode. It is designed to be used as a problem determination aid. This feature includes hardware that monitors the following during program execution: successful branches, the alteration of general registers, and instruction fetches from and alterations of specified areas of processor storage.

- A monitoring feature is standard that can be used to trace user-defined program events for the purpose of debugging or statistics gathering.

- A CPU timer and clock comparator are standard. The CPU timer provides an interval timing capability similar to that of the interval timer at location 80 but it is updated every microsecond, as is the time of day clock. The clock comparator can be used to cause an interruption when the time of day clock passes a specified value. These items provide higher resolution timing facilities than the interval timer and enable more efficient timing services routines to be written.

- New instructions that support dynamic address translation, the new timing hardware, and system control facilities are added to the System/370 instructions available for the Model 165.

- Processor storage is implemented using monolithic technology instead of discrete ferrite cores, and a Model 168 can have one million more bytes than a Model 165. Processor storage sizes of 1024K, 2048K, 3072K, and 4096K are available for the Model 168. Monolithic storage for the Model 168 is faster and more compact than core storage for the Model 165.

  As in a Model 165, processor storage in a Model 168 is four-way interleaved. However, each logical storage in Model 168 processor storage has a 480 nanosecond read/write cycle time for eight bytes on a doubleword boundary. A CPU fetch of a doubleword from processor storage (to a CPU register) requires 800 nanoseconds in the Model 168, which compares with 1440 nanoseconds in the Model 165.

  The physical size of a Model 168 CPU is not a function of the amount of processor storage installed. A Model 168 is smaller than a Model 165 with 512K and, therefore, is significantly smaller than Model 165 CPU's with more than 512K installed.

- The maximum aggregate channel data rate a Model 168 can support is significantly increased over that supported by a Model 165 because of the faster cycle time of processor storage and the new channel dual I/O bus that is used to transfer data from the channels to the storage control unit. A Model 168 configuration can handle a maximum aggregate data rate of 16 megabytes per second (MB). The maximum aggregate data rate possible on a Model 165 is 9.4 MB.

- 3330-series disk storage can be attached to a 2880 channel on a Model 168 via the Integrated Storage Control (ISC) feature as well as via 3830 Storage Control (Models 1 and 2). The optional ISC feature provides dual direct access storage control functions equivalent to two 3830 Storage Control Model 2 units, with the exception of four-channel switching. Two strings of from two to eight drives each can be attached to each of the two logical storage controls for a total of four 3330-series strings (32 drives) attached via the ISC feature.

The Model 168 is designed primarily to support a virtual storage environment that allows programmers to write and execute programs that are larger than the processor storage available to them. When virtual storage is supported, restraints normally imposed by the amount of processor storage actually available in a system are eased. The removal of certain restraints can enable applications to be installed more easily, and can be valuable in the installation and operation of online applications. While some of the new hardware features of the Model 168 and some of the new facilities supported by System/370 operating systems are designed to improve performance, a virtual storage environment is designed primarily to help improve the productivity of data processing personnel and enhance the operational flexibility of the installation.

The System/370 Model 168 is shown in Figure 10.1. The physical size of a Model 168 CPU does not depend on the amount of processor storage installed, and processor storage is contained within the CPU frames of a Model 168. All Model 168 systems (excluding the I/O configuration) are the same size, which is smaller than the size of a 512K Model 165. The physical size of a Model 168 is smaller than the size of a Model 165 as a result of the implementation of monolithic, instead of magnetic core, processor storage. Like a Model 165 CPU, a Model 168 CPU is water-cooled.

A Model 168 configuration consists of (1) a Model 168 CPU with integrated monolithic processor storage and, optionally, the Integrated Storage Control feature for 3330-series disk storage, (2) a stand-alone 3066 Model 2 System Console, (3) a stand-alone 3067 Model 2 Power and Coolant Distribution Unit, (4) stand-alone 2860, 2870, and 2880 channels (up to twelve channels maximum), and (5) a motor generator set. Field conversion of 3066 Model 1 and 3067 Model 1 units to Model 2 units is possible. The same motor generator set that is used to supply power to a Model 165 can be used with a Model 168 configuration. The motor generator supplies power to the Model 168 CPU but not to the integrated monolithic processor storage. A Model 165 CPU cannot be converted to a Model 168.

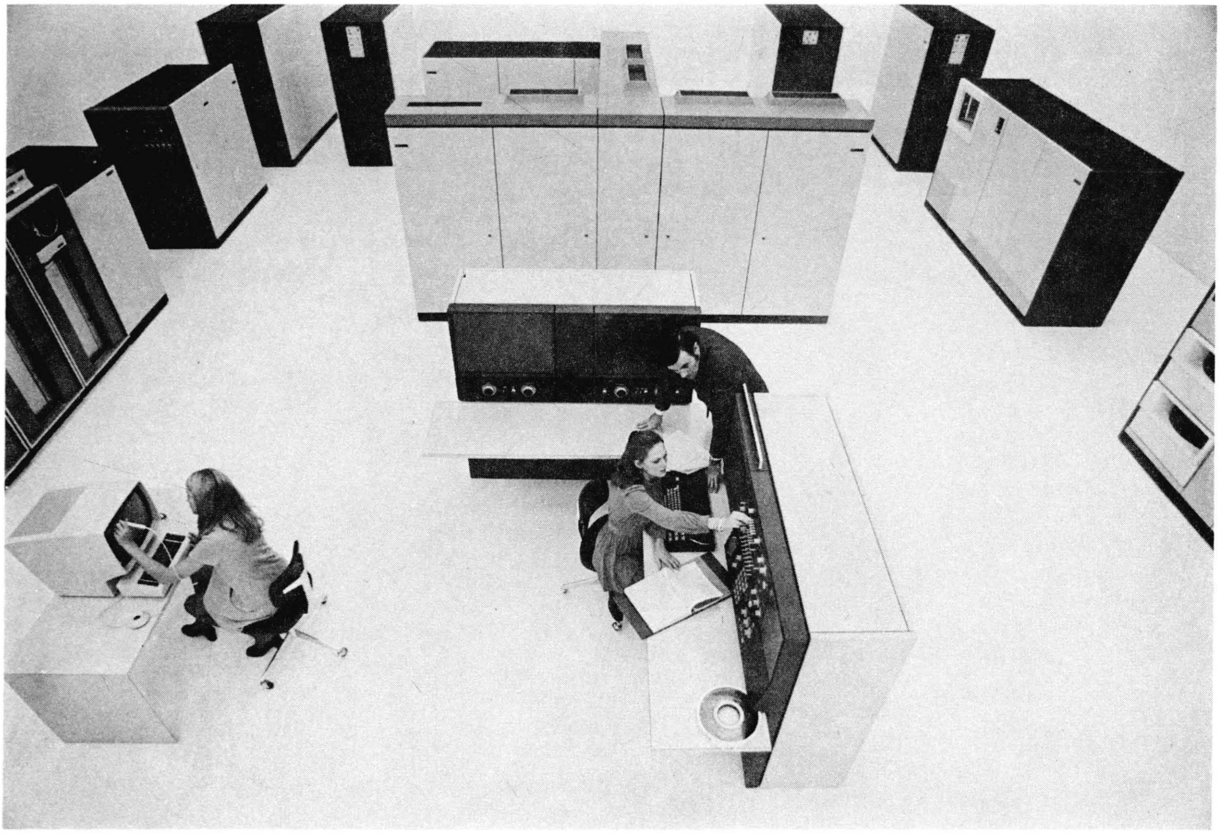


Figure 10.1. System/370 Model 168 (design model)

Monolithic technology is used to implement nearly all logic and all storage (processor, local, writable control, read-only control, and buffer) in the Model 168. Use of monolithic technology for processor storage, as well as for logic, represents a significant technological advance in storage implementation. The monolithic storage implemented in the Model 168 provides several advantages over the wired, discrete ferrite core storage implemented in the Model 165.

Monolithic storage is similar in design to monolithic logic circuitry, the latter representing a technological advance over the solid logic technology (SLT) introduced with the announcement of System/360. Since the technology associated with monolithic storage is like that used to produce monolithic logic, monolithic storage can be batch-fabricated.

Solid Logic Technology (SLT)

Monolithic technology is a breakaway from the hybrid circuit design concept of SLT and can best be explained by comparison with SLT. As shown in Figure 10.2, SLT circuits were implemented on half-inch ceramic squares called substrates. Metallic lands on the substrate formed interconnections onto which the components were soldered. These components consisted of transistors and diodes, which were integrated on silicon chips about the size of a pinhead, and thin film resistors. An SLT chip usually contained one component, and several chips and resistors were needed to form a circuit. In general, an SLT substrate contained a single circuit.

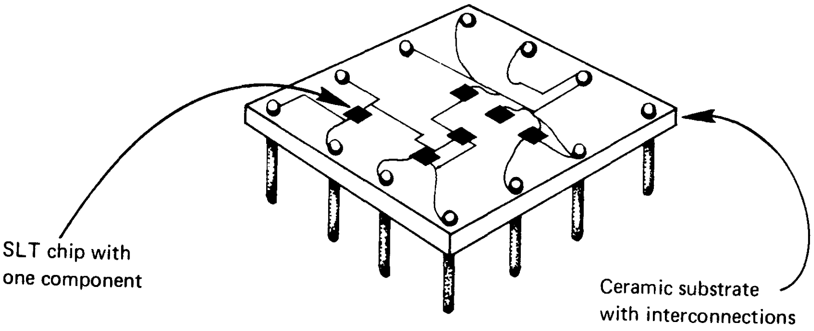

SLT chip with one component
Ceramic substrate with interconnections

Figure 10.2. SLT substrate

Monolithic System Technology (MST)

Monolithic system technology also makes use of a half-inch-square ceramic substrate with metal interconnections onto which chips are placed. However, in monolithic logic circuitry, large numbers of elementary components, such as transistors and resistors, are integrated on a single chip. An MST logic chip usually contains several interconnected logic circuits instead of only one component, as does an SLT chip. MST logic modules, each consisting of one substrate, are mounted on circuit cards, which are in turn mounted on circuit boards (as in SLT logic).

MST logic offers the following advantages over SLT:

- MST logic circuitry is intrinsically more reliable because many circuit connections are made on the chip, significantly reducing the number of external connections.

- Faster circuit speeds can be obtained because the path between circuits is considerably shorter.

- Space requirements for logic circuitry are reduced by the significantly higher density of components per chip.

Monolithic Storage

Monolithic storage design incorporates the same concepts described for monolithic logic. However, storage cells that are used to contain storage bits instead of logic circuits are implemented on a metal oxide semiconductor chip. In the Model 168, a monolithic storage array chip is approximately 1/8 by 3/16 of an inch in size and contains a large number of interconnected circuits. These circuits form storage bits and support circuitry (decoding, addressing, and sensing) on the chip.

Since power is required to maintain a one or zero state in a monolithic storage bit, data is lost when power is turned off, and monolithic storage is, therefore, said to be volatile. This is not true of core storage, which retains a magnetized state when power is removed.

The following are the advantages of monolithic over core storage:

- Faster storage speeds are obtained, first, because of the shorter paths between storage circuitry and, second, because of the nondestructive read-out capability of monolithic storage. Since core storage read-out is destructive, a regeneration cycle is required after a read and a readout cycle is required prior to a write. These types of regeneration cycles are not required for monolithic storage.

- Storage serviceability is enhanced because storage is implemented in accessible, easily replaceable cards. Diagnostic routines need only identify the failing storage card, which can be replaced in a matter of minutes.

- Space requirements for system storage are reduced. Dense bit packaging per chip is achieved by the use of monolithic technology and by the fact that the regularity of a storage pattern lends itself to such packaging.

SECTION 20: ARCHITECTURE DESIGN AND SYSTEM COMPONENTS

20:05 ARCHITECTURE DESIGN

Extended System/370 architecture embodies two different modes of system operation, basic control (BC) mode and extended control (EC) mode, as determined by bit 12 of the current PSW. When a Model 168 operates in BC mode, the contents, layout, and function of permanently assigned processor storage locations 0 to 127 are identical to these locations in System/360 Models 22 and up (except 44 and 67) with the exception of the use of PSW bit 12. BC mode essentially is the System/360-compatible mode of System/370 operation.

When EC mode is operative in the Model 168, the format of the PSW is altered and the number of permanently assigned locations extends beyond processor storage address 127. Changes to the PSW consist of the removal of certain fields to create space for additional mode and mask bits that are required for new functions, such as dynamic address translation and program event recording. The removed fields are assigned to locations above 127 and to a control register.

EC mode is effective when PSW bit 12 is a one. BC mode is effective when this bit is a zero. BC mode is established during initial program reset. Therefore, a control program must turn on bit 12 of the PSW in order to cause EC mode to become operative. As a result, control and processing programs written for System/360 (Models 22 and up except 44 and 67) will run without modification in BC mode on a System/370 Model 168 that has a comparable hardware configuration, with the following exceptions:

1. Time-dependent programs. (They may or may not execute correctly.)

2. Programs that use machine-dependent data such as that which is logged in the machine-dependent logout area. (OS SER error-logging routines for System/360 models will not execute correctly.)

3. Programs that use the ASCII mode bit in the PSW (bit 12). ASCII mode is not implemented and this bit is used in System/370 to specify BC or EC mode of operation.

4. Programs that depend on the nonusable lower processor storage area being smaller than 1938 bytes. This area can be reduced to 512 bytes by moving the CPU extended logout area.

5. Programs deliberately written to cause certain program checks.

6. Programs that depend on devices or facilities not implemented in the Model 168.

7. Programs that use model-dependent operations of the System/370 Model 168 that are not necessarily compatible with the same operations on System/360 models.

8. Programs that depend on the validity of storage data after system power has been turned off and then on.

Only BC mode is implemented in the Model 165. Hence, control and processing programs that currently operate on a Model 165 will run without modification in BC mode on a Model 168 that has a comparable hardware configuration, with the following exceptions:

1. Time-dependent programs. (They may or may not execute correctly.)

2. Programs that depend on the nonusable lower processor storage area being smaller than 1938 bytes. (The nonusable area in the Model 165 is 1504 bytes.)

3. Programs that use machine-dependent data such as that which is logged in the machine-dependent logout area.

4. Programs deliberately written to cause certain program checks.

5. Programs that depend on the validity of storage data after system power has been turned off and then on.

OS control programs are designed to support either BC or EC mode of system operation. OS PCP, MFT, and MVT control programs generated for a Model 65, 67, or 75 support BC mode operations on a Model 168. OS control and processing programs being used on a Model 65, 67, or 75 are subject to the eight compatibility restrictions in the first list. If an OS MFT or MVT control program that was generated for a Model 65, 67, or 75 is used on a Model 168, the system should be set to check stop on machine checks. (Section 60:30 in A Guide to the IBM System/370 Model 165, GC20-1730, discusses the reason.)

OS MFT and MVT support for the Model 168 in BC mode will be provided in Release 21.6. OS MFT and MVT control programs generated for a Model 165 using OS Release 21.6 will also operate on a Model 168 to support BC mode of system operation. Processing programs that are used on the Model 165 will operate under OS MFT or MVT control on a Model 168 in BC mode subject to the five compatibility restrictions in the second list.

Support of Model 168 systems operating in EC mode is provided by OS/VS1, OS/VS2, and VM/370, each of which is designated as system control programming (SCP). All of these operating systems support a virtual storage environment using dynamic address translation, which operates only when the system is in EC mode. VM/370 supports a virtual machine environment. User-written processing programs that operate on a Model 165 or 168 under OS MFT or MVT control can operate under OS/VS1 or OS/VS2, respectively, on a Model 168 with little or no modification, as discussed in the optional programming systems supplements (Sections 90 and 100). Hence, compatible growth from a System/360 or a BC mode nonvirtual storage environment to an EC mode virtual storage environment is provided.

The following are standard features of the Model 168 that are functionally identical to the same features of the Model 165:

- Instruction set that includes System/360 instructions and the following System/370 instructions:

  COMPARE LOGICAL CHARACTERS UNDER MASK
  COMPARE LOGICAL LONG
  INSERT CHARACTERS UNDER MASK
  LOAD CONTROL
  MOVE LONG
  SET CLOCK
  SHIFT AND ROUND DECIMAL
  START I/O FAST RELEASE
  STORE CHANNEL ID
  STORE CHARACTERS UNDER MASK
  STORE CLOCK
  STORE CONTROL
  STORE CPU ID

- Extended precision floating point
- Overlap of instruction fetching and preparation with instruction execution (implementation of the instruction and execution units is enhanced in the Model 168*)

- Store and fetch protection
- Multiple control registers (more registers are implemented in the Model 168 than in the Model 165*)
- Interval timer (3.3 millisecond resolution)
- Time of day clock
- Byte-oriented operands
- Extended external interruption masking
- Expanded machine check interruption class (additional facilities are provided in the Model 168*)
- Extended channel logout
- Instruction retry, ECC on processor storage, and command retry (RAS features)
- Writable monolithic control storage
- High-speed buffer storage - 8K
- Direct Control

The following are optional features of the Model 168 that are functionally identical to the same features on the Model 165:

- High-speed multiply (increases speed of fixed- and floating-point multiply operations by a factor of two to three)
- Buffer Expansion for the addition of 8K of buffer storage (the 16K buffer has a slightly different organization in the Model 168*)
- 7070/7074 Compatibility
- 7080 Compatibility
- 709/7090/7094II Compatibility
- 2870 Multiplexer Channels and attachment feature, 2860 Selector Channels and attachment feature, and 2880 Block Multiplexer Channels (one 2860, one 2880, or one 2870 with one selector subchannel is required)
- Extended Channels (for up to twelve channels)
- Channel-to-Channel Adapter on 2860
- 3066 Model 2 System Console (required) - a few new items are implemented

The following are new standard features of the Model 168:

- EC mode of system operation*
- Dynamic address translation*
- Reference and change recording*
- CPU timer and clock comparator*
- Program event recording*
- Monitoring feature*
- Program interruption for SET SYSTEM MASK instruction*
- Store status function*
- New instructions*
  LOAD REAL ADDRESS
  MONITOR CALL
  PURGE TLB
  RESET REFERENCE BIT
  SET CLOCK COMPARATOR
  SET CPU TIMER
  STORE CLOCK COMPARATOR
  STORE CPU TIMER
  STORE THEN AND SYSTEM MASK
  STORE THEN OR SYSTEM MASK
- Monolithic read-only control storage (instead of capacitor read-only*)
- Monolithic processor storage (instead of core storage)
- Channel dual I/O bus

*Part of the Dynamic Address Translation Facility of a Model 165 II. The functional descriptions of these items in this publication apply to their implementation in both the Model 168 and the Model 165 II, unless otherwise indicated.

The following are new optional features of the Model 168:

- Channel Indirect Data Addressing for 2860, 2870, and 2880 channels (required by the virtual storage operating systems and available for the Model 165 II)

- Integrated Storage Control for attachment of 3330-series disk storage

- Two-Channel Switch for Integrated Storage Control

All the new features of the Model 168 except Integrated Storage Control and those related to implementing virtual storage (such as dynamic address translation and reference and change recording) are discussed in the remainder of this section.

20:10 THE CENTRAL PROCESSING UNIT

Like the Model 165, the Model 168 has a CPU cycle time of 80 nanoseconds and an internal data path that is eight bytes wide. The implementation of local storage, read-only and writable control storage, expanded external interruption masking, and parity checking is the same in the two models. Control registers in addition to the four implemented in the Model 165 are implemented in the Model 168 in order to support new EC-mode-only functions. Additional control registers are implemented in the Model 165 II as well.

Implementation of the instruction and execution units in Models 168 and 165 differs in several aspects in order to provide better overlap of instruction preparation with instruction execution and to provide functions required by new Model 168 hardware features, such as dynamic address translation. (This new implementation is also provided in a Model 165 II.) Significant differences are the following:

- In the Model 168, up to four instructions can be prepared and await execution while one instruction is being executed. The Model 165 can prepare and hold up to three instructions.

- When an incorrect estimate of the success of a conditional branch has been made, the Model 168 can access the correct instruction one cycle sooner than can the Model 165.

- In the Model 168, a doubleword from a given instruction stream can be placed in the instruction buffers every machine cycle. This can be done every other cycle in a Model 165.

- In the Model 168, two registers are provided to hold data that is awaiting placement in processor storage. Each can hold up to eight bytes. The Model 165 has only one such register.

- The instruction unit in the Model 168 includes an instruction pretest function (explained under "Instruction Nullification" in Section 30:10).

- Imprecise interruptions do not occur in a Model 168. In a Model 165, an imprecise interruption occurs if an attempt is made to store data at an invalid storage address or at a storage protected location. The Model 168 implementation of pretesting (for the dynamic address translation function) also ensures that such conditions do not cause imprecise interruptions in the Model 168.

EXTENDED CONTROL MODE

Extended control mode, unlike basic control mode, is exclusively a System/370 mode and is not implemented in System/360. In a Model 168, the optional Channel Indirect Data Addressing feature must be installed on all stand-alone channels in order for the channels to operate with EC mode enabled. Note that IBM-supplied operating systems do not support System/370 models operating in EC mode without dynamic address translation operative also. Facilities that depend on which mode is in effect are discussed below. Any item not covered operates identically in BC and EC modes. (The discussion of EC/BC mode differences applies to the Model 165 II also.)

Change in PSW Format

When a System/370 operates in EC mode, the format of the PSW differs from the BC mode format. Both PSW formats are shown in Figure 20.10.1. In EC mode, the PSW does not contain individual channel mask bits, an instruction length code, or the interruption code for a supervisor call, external, or program interruption. The channel masks are contained in control register 2, and the other fields are allocated permanently assigned locations in fixed processor storage above address 127.

Removal of the fields indicated provides room in the EC mode PSW for control of new features that are unique to EC mode (such as PER and DAT) and for the addition of summary mask bits (such as channel and I/O masks). Use of a single mask bit to control the operation of an entire facility (such as program event recording) or an entire interruption class (such as I/O and external) simplifies the coding required to enable and disable the system for these interruptions.

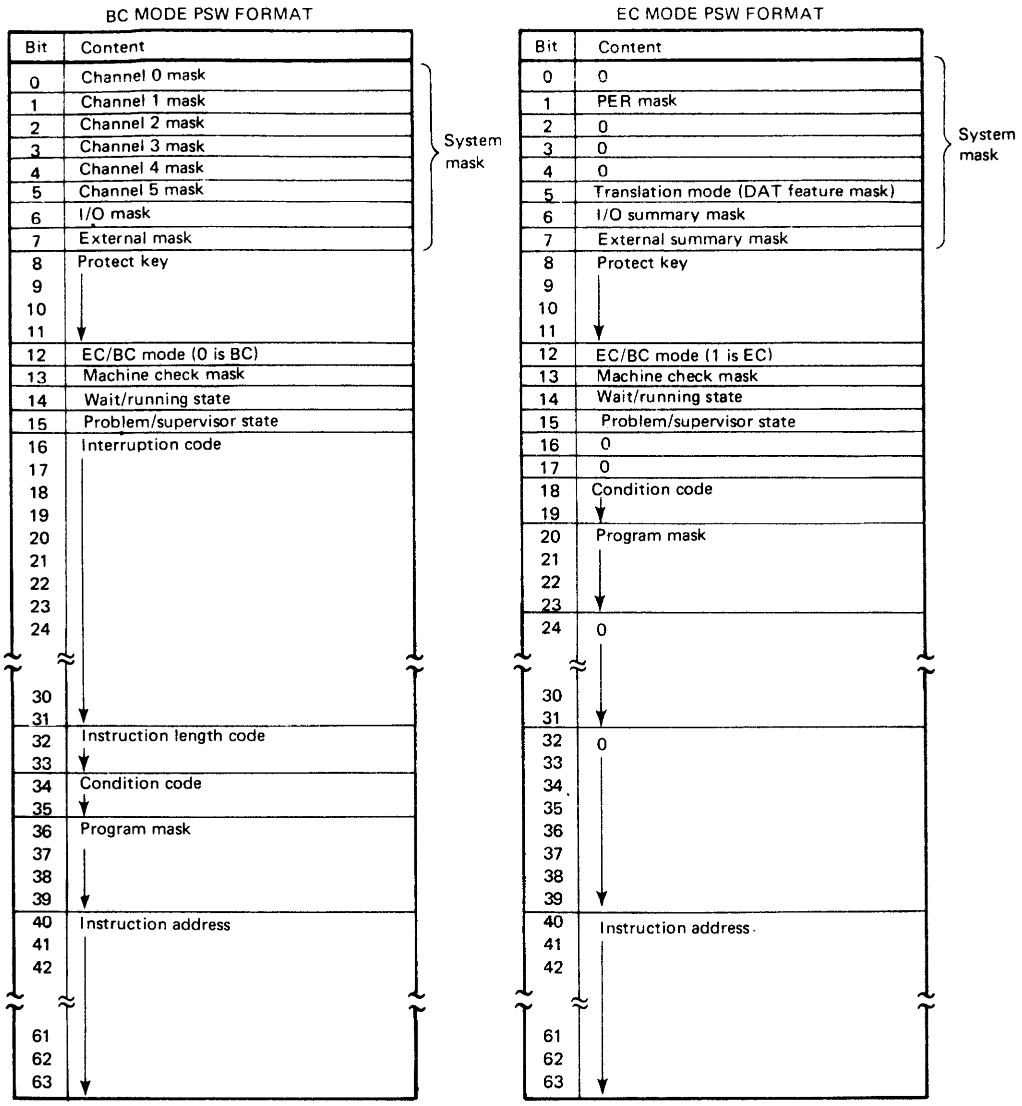
Change in Permanently Assigned Processor Storage Locations

When a System/370 operates in EC mode, the number of permanently assigned locations in lower processor storage is increased to include fields for storing instruction length codes, interruption codes (for supervisor call, external, and program interruptions), program event recording data, the I/O device address for an I/O interruption, and an exception address for the DAT feature. The model-independent BC mode and EC mode fixed storage areas for System/370 models are shown in Figure 20.10.2. The balance of the fixed area for the Model 168, that which has model-dependent fields, is shown in Figure 20.10.3. This model-dependent area is not affected by whether EC or BC mode is in effect except for locations 185 to 187, which contain the I/O address after an I/O interruption and an IPL only when EC mode is in effect.

The machine check interruption procedure and the format of the data logged on a machine check are the same in EC and BC modes, except for differences in the PSW format and the permanently assigned locations previously discussed.

Channel Masking Changes

When a System/370 operates in EC mode, interruptions from each channel are controlled by the summary I/O mask bit in the current PSW (bit 6) and an individual channel mask bit in control register 2. In the Model 168, bits 0 to 11 in control register 2 are assigned to control channels 0 to 11, respectively. Both the summary mask bit and the appropriate individual channel mask bit must be on in order for an interruption from a given channel to occur. In BC mode, only interruptions from channels 6 to 11 are controlled by individual channel mask bits in control register 2 and the I/O mask bit in the PSW. Interruptions from channels 0 to 5 are controlled only by channel mask bits in the current PSW (bits 0 to 5) in BC mode.



BC MODE PSW FORMAT

Bit	Content	
0	Channel 0 mask	System mask
1	Channel 1 mask	
2	Channel 2 mask	
3	Channel 3 mask	
4	Channel 4 mask	
5	Channel 5 mask	
6	I/O mask	
7	External mask	
8	Protect key	
9		
10		
11		
12	EC/BC mode (0 is BC)	
13	Machine check mask	
14	Wait/running state	
15	Problem/supervisor state	
16	Interruption code	
17		
18		
19		
20		
21		
22		
23		
24		
30		
31		
32	Instruction length code	
33		
34	Condition code	
35		
36	Program mask	
37		
38		
39		
40	Instruction address	
41		
42		
61		
62		
63		

EC MODE PSW FORMAT

Bit	Content	
0	0	System mask
1	PER mask	
2	0	
3	0	
4	0	
5	Translation mode (DAT feature mask)	
6	I/O summary mask	
7	External summary mask	
8	Protect key	
9		
10		
11		
12	EC/BC mode (1 is EC)	
13	Machine check mask	
14	Wait/running state	
15	Problem/supervisor state	
16	0	
17	0	
18	Condition code	
19		
20	Program mask	
21		
22		
23		
24	0	
30		
31		
32	0	
33		
34		
35		
36		
37		
38		
39		
40	Instruction address	
41		
42		
61		
62		
63		

Figure 20.10.1. BC mode and EC mode PSW formats

Expansion of Storage Protect Key Size

The size of the storage protect key associated with each 2K storage block is expanded from five to seven bits in the Model 168. The two additional bits (reference and change) are included for use with dynamic address translation and are discussed in Section 30:10. The SET STORAGE KEY instruction sets a seven-bit key regardless of the mode, BC or EC, in effect. The INSERT STORAGE KEY loads a five-bit or a seven-bit key into the register indicated depending on whether BC or EC mode, respectively, is in effect.

BC MODE FIXED AREA 0-159

Decimal locations					
0	IPL PSW				
8	IPL CCW 1				
16	IPL CCW 2				
24	External old PSW				
32	Supervisor call old PSW				
40	Program old PSW				
48	Machine check old PSW				
56	I/O old PSW				
64	Channel status word — CSW				
72	Channel address word — CAW	76	Unused		
80	Interval timer	84	Unused		
88	External new PSW				
96	Supervisor call new PSW				
104	Program new PSW				
112	Machine check new PSW				
120	I/O new PSW				
128	0	132	0		
136	0	140	0		
144	0	148	0	Monitor class	0
152	0	156	0	Monitor code	

- Model independent among System/360 and System/370 models in BC mode except for PSW bit 12
- Processed by the control program

EC MODE FIXED AREA 0–159

0	IPL PSW					
8	IPL CCW 1					
16	IPL CCW 2					
24	External old PSW					
32	Supervisor call old PSW					
40	Program old PSW					
48	Machine check old PSW					
56	I/O old PSW					
64	Channel status word — CSW					
72	Channel address word — CAW	76	Unused			
80	Interval timer	84	Unused			
88	External new PSW					
96	Supervisor call new PSW					
104	Program new PSW					
112	Machine check new PSW					
120	I/O new PSW					
128	0	132	0	External int. code		
136	0	ILC	SVC int. code	140 0	ILC	Program int. code
144	0	Translation excp. addr.	148 0	Monitor class	PER code	0
152	0	PER address	156 0	Monitor code		

- Model independent among System/370 models in EC mode
- PSW format is different from that of BC mode PSW
- Processed by the control program

Figure 20.10.2. Model 168 model-independent fixed storage locations for BC and EC modes

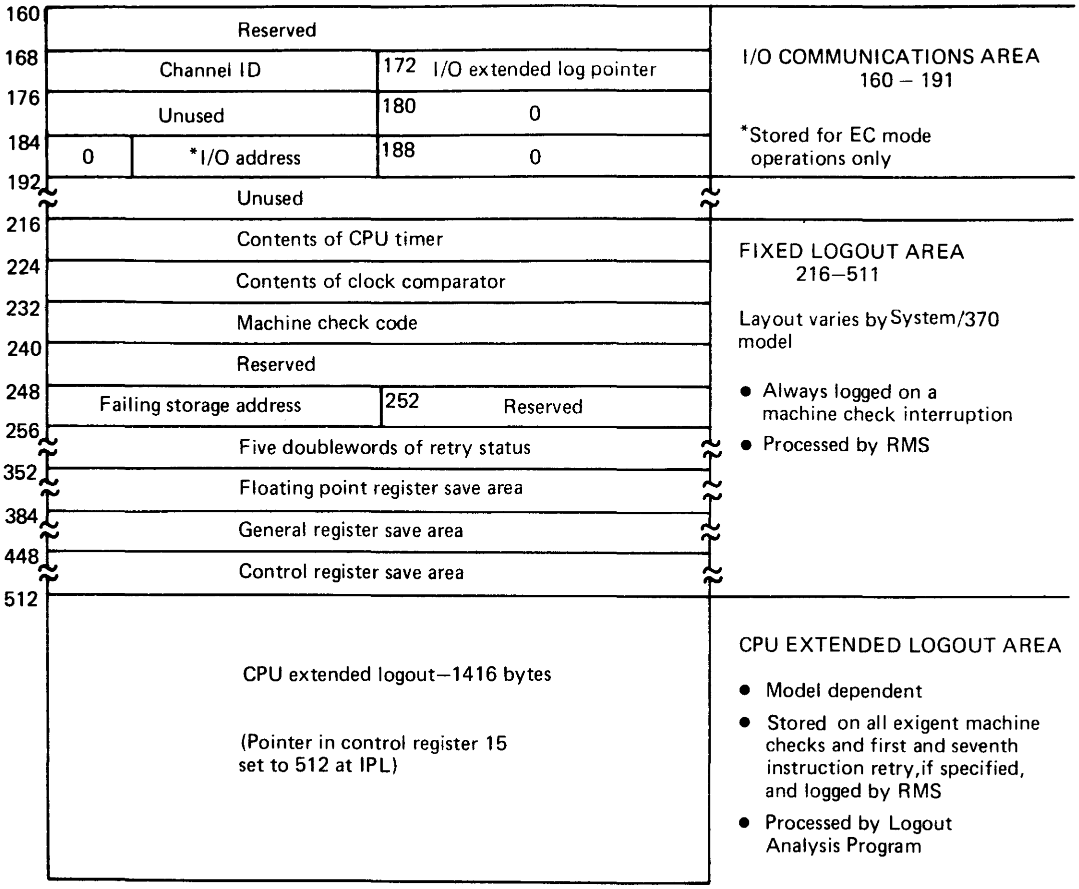

160
168
176
184
192
216
224
232
240
248
256
352
384
448
512
Reserved
Channel ID
172 I/O extended log pointer
Unused
180 0
0
*I/O address
188 0
Unused
Contents of CPU timer
Contents of clock comparator
Machine check code
Reserved
Failing storage address
252 Reserved
Five doublewords of retry status
Floating point register save area
General register save area
Control register save area
CPU extended logout—1416 bytes
(Pointer in control register 15 set to 512 at IPL)
I/O COMMUNICATIONS AREA 160 – 191
*Stored for EC mode operations only
FIXED LOGOUT AREA 216–511
Layout varies by System/370 model
• Always logged on a machine check interruption
• Processed by RMS
CPU EXTENDED LOGOUT AREA
• Model dependent
• Stored on all exigent machine checks and first and seventh instruction retry, if specified, and logged by RMS
• Processed by Logout Analysis Program

Figure 20.10.3. Model 168 model-dependent fixed storage locations

Changes to Certain System/370 Instruction Definitions

All Model 168 instructions are valid in BC and EC modes. However, because of differences between the PSW format and the permanently assigned storage locations in EC and BC modes, the definition of certain instructions is affected. Instructions provided for both System/360 and System/370 whose definition is altered for EC mode are:

BRANCH AND LINK (RR, RX)	SET STORAGE KEY
INSERT STORAGE KEY	SET SYSTEM MASK
LOAD PSW	SUPERVISOR CALL
SET PROGRAM MASK	

Revised definitions of these instructions to include BC/EC mode differences are contained in System/370 Principles of Operation (GA22-7000-2, or later editions). Programs that operate in BC mode and that use LOAD PSW and/or SET SYSTEM MASK (SSM) instructions must be modified in order to operate correctly in EC mode. The eight-byte PSW to be loaded by LPSW instructions and the eight-bit system mask to be set by SSM instructions must be changed to EC mode format. (Programs that use SSM instructions and that are executed in an OS/VS1 or OS/VS2 environment need not be modified because the interruption for SSM instructions and an SSM simulation routine, described next, are supported.)

Programs that use the other instructions listed do not have to be changed in order to operate correctly in EC mode, unless they use other facilities that are mode dependent. Programs that operate in BC mode and that use the STORE THEN OR SYSTEM MASK and STORE THEN AND SYSTEM MASK instructions (not provided in System/360) must also be modified in order to operate correctly in EC mode.

Program Interruption for Set System Mask Instruction

When a System/370 is operating in EC mode, execution of the SET SYSTEM MASK instruction is under the control of the SSM mask in control register 0. When the SSM mask bit is a one, an attempt to execute an SSM instruction causes a program interruption without execution of the SSM instruction. When the SSM mask bit is a zero, SSM instructions are executed as usual.

This interruption is implemented to enable existing programs that were written for System/360 models or for System/370 BC mode of operation to execute correctly in EC mode without modification of the system mask field addressed by existing SSM instructions. When an interruption occurs for an SSM instruction, the contents of the BC mode format system mask indicated by the SSM instruction can be inspected, and the appropriate EC mode mask bits can then be set by an SSM simulation routine.

Program Event Recording

Program event recording (PER), a standard feature of the Model 168, is designed to assist in program debugging by enabling a program to be alerted to any combination of the following events via a program interruption:

- Successful execution of any type of branch instruction

- Alteration of the contents of the general registers designated by the user

- Fetching of an instruction from a processor storage area defined by the user

- Alteration of the contents of a processor storage area defined by the user

The PER feature can operate only when EC mode is in effect and the PER mask, bit 1 of the current PSW, is on. Control register 9 (bits 0 to 3) is used to specify which of the four PER event types are to be monitored. A PER program interruption is taken after the occurrence of an event only if both the PER mask bit and the respective event mask bit in control register 9 are on. Control register 9 (bits 16 to 31) also specifies which of the 16 general registers are to be monitored if monitoring of this event is specified. Control registers 10 and 11 indicate the beginning address and the ending address, respectively, of the contiguous processor storage area that is to be monitored for instruction fetching and/or alteration.

When an event that is being monitored is detected, PER hardware causes a program interruption, if the PER mask bit is on, and identification of the type of event is stored in the fixed processor storage area (location 150). The address of the instruction associated with the event is also stored (locations 153 to 155). Program event interruptions are lost if they occur when the PER mask bit or the particular event mask bit is off.

If dynamic address translation mode is also specified when PER is active, virtual storage addresses instead of real storage addresses (discussed in Section 30) are placed in the control registers to monitor references to a contiguous virtual storage area.

MONITORING FEATURE

The monitoring feature is standard on the Model 168 (and on the Model 165 II). This feature provides the capability of monitoring the occurrence of programmed events. For example, monitoring can be used to perform measurement functions (how many times a routine was executed) or tracing functions for the purpose of program debugging (which routines were executed).

The MONITOR CALL instruction is provided with the monitoring feature. Execution of this instruction indicates the occurrence of one of the events being monitored. The operands of the MONITOR CALL instruction permit specification of up to 16 classes of events, each class with up to 16 million unique types of events. The 16 monitor classes are individually maskable via mask bits in control register 8. A program interruption occurs when a MONITOR CALL instruction is executed, if the monitor class indicated is specified in control register 8, and the event identification (class and type) is stored in the fixed storage area.

Both the PER facility and the monitoring feature are provided for debugging purposes. The two features differ from one another in (1) the number of events that can be defined, (2) whether events are defined by the hardware or the programmer, and (3) whether hardware or the programmer checks for the events and causes the interruptions. When PER is used, once the events to be monitored have been designated by the user, CPU hardware checks for the occurrence of the events and causes the interruption. When the monitoring feature is used, the user defines the events to be monitored (up to 16 classes with up to 16 million monitor codes each instead of only four events), determines when the events occur, and causes program interruptions by issuing MONITOR CALL instructions.

NEW INSTRUCTIONS

Two of the new instructions provided in the Model 168 enable a program to directly manipulate the system mask. Other new instructions provided are related to specific features (such as monitoring, dynamic address translation, the clock comparator, and the CPU timer) and are discussed with these features.

STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK are two new privileged instructions that affect the system mask (bits 0 to 7 in the current PSW). The STORE THEN AND SYSTEM MASK instruction provides, via a single instruction, the capability of storing the current system mask for later restoration, while selectively zeroing certain system mask bits. The STORE THEN OR SYSTEM MASK provides system mask storing and selective setting of system mask bits to ones. These two instructions simplify the coding required to alter the system mask, particularly when the existing settings must be saved. (These instructions are implemented in the Model 165 II also.)

CLOCK COMPARATOR AND CPU TIMER

These timing facilities are standard on the Model 168. (They are also provided in a Model 165 II.) The clock comparator provides a means of causing an external interruption when the time of day clock has passed a time specified by a program. This feature can be used to initiate an action, terminate an operation, or inspect an activity, for example, at specific clock times during system operation.

The clock comparator has the same format as the time of day clock and is set to zero during initial program reset. The SET CLOCK COMPARATOR privileged instruction is provided to place a value that represents a time of day in the clock comparator. When clock comparator interruptions are specified via the external interruption summary mask bit in the current PSW and the clock comparator subclass mask bit in control register 0, an external interruption occurs when the time of day clock value is greater than the clock comparator value. Bits 0 to 51 of the time of day clock and the clock comparator are compared. If clock comparator interruptions are masked when this condition occurs, the interruption remains pending only as long as the time of day clock value remains higher than the value in the clock comparator. The STORE CLOCK COMPARATOR privileged instruction can be used to obtain the current value of the clock comparator.

The use of a clock comparator to cause an interruption when a specified time is passed, instead of the interval timer at location 80, offers two advantages. First, the time of day clock increments when the system is in the stopped state while the interval timer does not. Hence, if a system stop occurs during processing and the system is restarted, the clock comparator can still cause an interruption at the time requested. The interruption caused by the interval timer in such a situation is late. Second, implementing the time of day clock and the clock comparator in the same doubleword format eliminates having to convert doubleword time of day clock units to single word interval timer units.

The CPU timer provides a means of causing an external interruption when an interval of time specified by a program has elapsed. The CPU timer is implemented as a binary counter with a format identical to that of the time of day clock; however, bit 0 of the CPU timer is considered to be a sign. The CPU timer has a maximum time period half as large as that of the time of day clock and the same resolution of one microsecond. When both the CPU timer and the time of day clock are running, the stepping rates of the two are synchronized such that they are stepped at exactly the same rate.

The CPU timer is set to zero at initial program reset and the SET CPU TIMER privileged instruction is provided to place an interval of time in the CPU timer. The STORE CPU TIMER privileged instruction can be used to obtain the current CPU timer value. The CPU timer decrements every microsecond. If the external interruption summary mask bit in the current PSW and the CPU timer subclass mask bit in control register 0 are on, an external interruption occurs whenever the CPU timer value is negative (not just when the timer goes from positive to negative), indicating that the time interval has elapsed. The CPU timer decrements when the CPU is executing instructions (including instruction retry operations) and while the CPU is in the wait state. It is not decremented when the system is in the stopped state.

While providing essentially the same function as the interval timer at location 80, the CPU timer provides advantages over the interval timer as follows. Task processing intervals of less than 3.3 milliseconds are accurately measured because of the one microsecond resolution of the CPU timer. A pending CPU timer interruption is reset when a SET CPU TIMER instruction is issued to set a positive value in the CPU timer, eliminating the need to take an interruption in order to reset the CPU timer, as is required for the interval timer. In addition, the amount of timing facilities processing required during a task switch can be reduced. This can result from the fact that the format of the time of day clock and the CPU timer are the same. Conversion of doubleword time of day clock values to single-word interval timer values is eliminated, and timer queues can be structured such that little of the processing currently required during a task switch, when the interval timer is used, is necessary.

RELIABILITY, AVAILABILITY, AND SERVICEABILITY FEATURES

The following hardware RAS features implemented in the Model 168 are identical to those provided in the Model 165:

- Automatic retry of most failing CPU operations by hardware

- ECC checking on processor storage to correct all single-bit and detect all double-bit errors

- I/O operation retry facilities, including the storing of channel retry data during an I/O interruption that results from an error, and channel/control unit command retry procedures to correct certain failing I/O operations

Implementation of machine check interruption facilities is expanded in the Model 168 to provide more definitive logout information when a machine check interruption is taken, and a new buffer row deletion function is implemented. Machine check interruption facilities are the same in Models 168 and 165 except for the following (which also applies to a Model 165 II):

- The instruction processing damage subclass of machine check interruption, not implemented in the Model 165, is implemented in the Model 168. Instruction processing damage will be indicated in the machine check code (shown in Figure 20.10.4) when a CPU error occurs that is not retryable or that was unsuccessfully retried, unless an LPSW instruction or an interruption was in process at the time of the failure or the failure was a hang detect. In these cases, system damage is indicated. In the Model 165, system damage is indicated for all CPU and storage errors that cannot be retried or that are unsuccessfully retried. Implementation of the instruction processing damage subclass in the Model 168 is designed to identify errors that can be associated with a specific task so that only that task need be abnormally terminated. Code is included in the Model 165 MCH routine that attempts, when a system damage error is indicated, to distinguish system damage from damage that can be associated with a task. This code will not be required for the Model 168.

- Whenever a machine check interruption is taken to record information about a correctable or an uncorrectable processor storage error, the failing processor storage address is placed in locations 248-251. The machine check code will indicate the type of processor storage error and whether or not the stored failing storage address is valid.

- In the Model 168, each block in the high-speed buffer has a delete bit associated with it in the address array for the buffer, as in the Model 165. However, in the Model 168 each row within the buffer also has a row delete trigger associated with it. (There are four rows in the 8K buffer and eight rows in the 16K buffer, as shown later in Figure 20.15.2.) Whenever certain buffer errors occur and the Model 168 CPU is enabled for system recovery machine check interruptions, hardware determines the buffer row in which the error occurred. The row delete trigger is turned on for that row. This indicates that the buffer row is disabled and that the CPU can no longer fetch data from or store data in the deleted buffer row. The machine check code stored during the interruption that occurs after a buffer row is deleted indicates a degradation error condition.

The mode bit implemented in the Model 165 that can be set by a DIAGNOSE instruction to cause the entire buffer to be disabled is not implemented in the Model 168. However, the Model 168 contains a mode bit that can be set to cause the buffer row deletion mechanism to be disabled. This selective buffer deletion facility allows only one-quarter of an 8K buffer or one-eighth of a 16K buffer to be automatically disabled by hardware at the time certain buffer errors occur and avoids total buffer disabling after an error.

- The time of day clock damage interruption, maskable by the external mask bit and PSW bit 13, is expanded to include clock comparator and CPU timer errors. Its name is changed to "Timing Facilities Damage". When a STORE CLOCK COMPARATOR or a STORE CPU TIMER instruction is issued and the addressed timing facility has an error, or when the CPU timer or the clock comparator develops an error, a timing facilities damage interruption occurs if the timing facilities damage mask bit is a one.

- Whenever a machine check interruption occurs in a Model 168, the current value of the CPU timer is stored in locations 216 to 223 and the current value of the clock comparator is stored in locations 224 to 231. Bits 46 and 47 of the machine check code, shown in Figure 20.10.4, are used to indicate whether or not these values were stored correctly. Whenever a machine check interruption occurs as a result of a processor storage error, the address of the error location is stored in locations 249 to 251. Bit 24 of the machine check code is used to indicate whether or not the value was correctly stored.

- The size of the CPU extended logout area in the Model 168 is 1416 bytes instead of 992 bytes as in the Model 165 in order to log additional status information when a machine check occurs.

Model 168 recovery management routines (machine check and channel check handlers) that operate in BC mode will be included in OS MFT and OS MVT as of Release 21.6. They will provide the same recovery functions as are provided for the Model 165 and support of new Model 168 machine check facilities. Model 168 RMS routines will provide recovery for system damage errors similar to that provided by Model 165 RMS routines (attempt to repair damaged control program storage areas by loading a refreshable module). In addition, an instruction processing damage interruption will be recognized in the Model 168, and RMS will attempt to identify the affected task and abnormally terminate it. Model 168 RMS will also recognize a degradation interruption that indicates buffer row deletion by the hardware, and the operator will be notified of this hardware action.

These recovery routines are also included in OS/VS1 and OS/VS2 and are modified to operate correctly when the Model 168 is operating in EC and dynamic address translation modes. A discussion of how these recovery routines differ from those provided for BC mode operations is contained in each optional programming systems supplement, which also discusses the programmed repair facilities (OLTEP, OLT's, Logout Analysis program) provided.

Fixed Logout Area Locations 232–239

	0 – 8 Machine Check Types											16 – 18 Storage Error				20 – 31, 46, 47 Validity Bits															48 – 63 CPU Extended Log Length
	SD	PD	SR	UNUSED	CD	ED	UNUSED	DG	UNUSED	UNUSED	BACKED UP	SE	SC	KE	UNUSED						UNUSED	UNUSED						UNUSED			Zero if no logout or 1416 bytes
Bit	0	1	2	3	4	5	6	7	8	9-14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32-45	46	47	48 63

Bit	Interrupt Type
0	SD – System Damage
1	PD – Instruction Processing Damage
2	SR – System Recovery
4	CD – Timing Facilities Damage
5	ED – External Damage
7	DG – Degradation

Bit	Error
15	Delayed Interruption
16	Storage Error Uncorrected
17	Storage Error Corrected
18	Protection Key Error

Bit	Valid Fixed Area Data
20-23	Machine Check Old PSW (48-55) 20 AMWP 21 Masks and Protect Key 22 Program Mask and Condition Code 23 Instruction Address
24	Failing Storage Address
27	Floating Point Registers (352-383)
28	General Registers (384-447)
29	Control Registers (448-511)
30	CPU Extended Logout
31	Storage (Validity of storage being processed by instructions when interruption occurred.)
46	CPU Timer Value
47	Clock Comparator Value

Figure 20.10.4. Model 168 machine check code

20:15 STORAGE

PROCESSOR (MAIN) STORAGE

Like the Model 165, the Model 168 has a two-level storage system in which large high-speed monolithic processor storage backs up small, higher speed buffer storage. A maximum of 4096K of processor storage can be installed in a Model 168. The Model 165 can have a maximum of 3072K. Processor storage is available for the Model 168 in 1024K increments as follows:

Model	Capacity
J	1024K
K	2048K
KJ	3072K
L	4096K

Processor storage in a Model 168 is four-way doubleword interleaved, as it is in a Model 165. The processor storage installed in a Model 168 is divided into four logical storages, each of which can operate independently from the other three logical storages. Logical storages can be selected at 80 nanosecond intervals. Consecutively addressed doublewords are spread across logical storages, as shown in Figure 20.15.1, so that access to four doublewords can be overlapped.

The data path to and from processor storage is eight bytes wide. As in a Model 165, the storage control unit (SCU) provides the interface to the logical storages. A logical storage in the Model 168 has a read/write cycle time of 480 nanoseconds for eight bytes on a doubleword boundary. A cycle time of 800 nanoseconds is required for a partial write (any number of bytes less than eight). Once selected, a logical storage remains busy for the storage cycle and cannot be selected again until 480 or 800 nanoseconds have passed.

The access time of processor storage is 560 nanoseconds for eight bytes. That is, once the SCU sends a storage selection, 560 nanoseconds are required to fetch eight bytes of data and make them available in the storage control unit, from which they can be sent to the instruction unit, the buffer, etc. A CPU fetch of eight bytes from processor storage in the Model 168 requires 800 nanoseconds. This is the time between request acceptance and availability of the eight bytes in a CPU register. A CPU fetch of eight bytes in the Model 165 requires 1440 nanoseconds. Table 20.15.1 summarizes cycle and access times for the Model 168.

As in a Model 165, processor storage in a Model 168 can be accessed concurrently by any combination of one or more channels and the CPU for a total of four unique logical storage requests. When simultaneous requests for the same logical storage are received, the storage control unit schedules the requests according to a priority scheme. This priority is the same in Models 168 and 165. That is, the channels have priority over the CPU and the priority among channels is definable at channel installation time. Processor storage access times stated are obtainable assuming the logical storages are free at the time a request is directed to them.

Processor Storage Reconfiguration

As shown in Figure 20.15.1, the processor storage present in the Model 168 is divided into from one to four segments of 1024K bytes each. Segment numbers 0 to 3 are used. If an uncorrectable processor storage error occurs, the segment containing the malfunctioning location(s) can be manually configured out of the system by the operator.

The configuration panel on the 3066 Model 2 System Console is used to enable storage segments, assign a one megabyte range of addresses to each enabled segment, and establish four-way interleaving or serial operations. The configuration panel is shown in Figure 20.15.1. The operator selects a configuration by inserting pins in the appropriate hubs. The storage configuration indicated by the panel is made effective during a system reset.

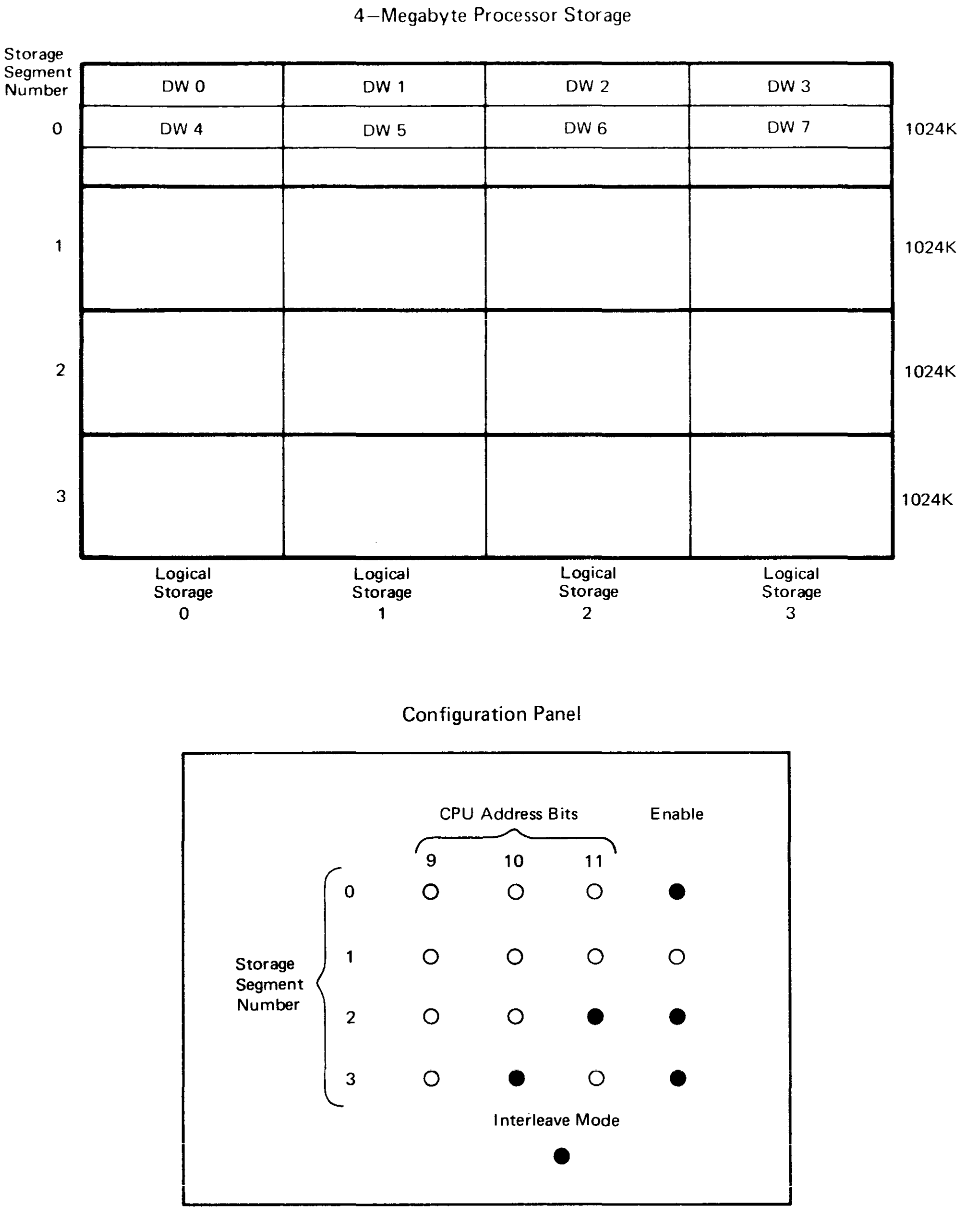


Figure 20.15.1. Model 168 processor storage organization and configuration panel

Table 20.15.1. Model 168 cycle and access times

Cycle or Access Time	Time in Nanoseconds
CPU cycle time	80
Local storage cycle time	80
Control storage cycle time	80
Processor (logical) storage read/write cycle time (for eight bytes on a doubleword boundary)	480
Processor (logical) storage cycle time for a partial write (fewer than 8 bytes)	800
Minimum time between successive selects to processor storage	80
Processor storage access time (from time of SCU select to availability of data in the storage control unit)	560
CPU fetch of 8 bytes from processor storage (from time of request acceptance to availability of data in a CPU register)	800
CPU fetch of eight bytes from buffer (from time of request acceptance to availability of data in a CPU register, as in the Model 165)	160
Minimum time between successive buffer requests	80

When a pin is inserted in the interleave mode hub, four-way interleaving is selected. The absence of a pin selects serial (noninterleaved) operations. The presence of a pin in an enable hub indicates the associated segment is to be included in the active storage configuration. The three CPU address bit hubs for a segment are used to indicate the range of processor storage addresses that are to be assigned to the segment. When none of the three address bit hubs for a segment contains a pin, address range 0 to 1024K is selected. When only the address bit 11 hub for a segment contains a pin, addresses 1024K to 2048K are selected. The presence of a pin only in the address bit 10 hub selects addresses 2048K to 3072K, and a pin only in the address bit 9 hub selects addresses 3072K to 4096K.

The storage configuration selected by the control panel shown in Figure 20.15.1 is the following:

- Segments 0, 2, and 3 are enabled and segment 1 is disabled.
- Segment 0 is assigned addresses 0 to 1024K, segment 2 is assigned addresses 1024K to 2048K, and segment 3 is assigned addresses 2048K to 3072K.
- Four-way interleaving is enabled.

Storage ripple functions are provided in the Model 168 for read-only control storage, writable control storage, local storage, and processor storage, as for the Model 165. The inline ripple facility of the Model 165 is not implemented in the Model 168.

HIGH-SPEED BUFFER STORAGE

As in the Model 165, an 8K buffer is standard in the Model 168 and installation of the optional Buffer Expansion feature permits inclusion of an additional 8K of buffer storage. Buffer storage provides high-speed data access for CPU fetches. In a Model 168, as in a Model 165, the CPU can obtain eight bytes from the buffer in 160 nanoseconds (two CPU cycles) and a request can be initiated every cycle. This is the time between request acceptance and availability of the data in a CPU register. If the buffer does not contain the data required, the data must be obtained from processor storage. A CPU to processor storage fetch requires 800 nanoseconds.

Use of the high-speed buffer in Models 168 and 165 is almost identical. (This description of the buffer in the Model 168 also applies to the Model 165 II.) When a data fetch request is made by the CPU, a determination is made of whether or not the requested data is in the high-speed buffer by the interrogation of the address array of the buffer's contents. If the data requested is present in the buffer, it is sent directly to the CPU without a processor storage reference. If the requested data is not currently in the buffer, a processor storage fetch is made and the data obtained is sent to the CPU. The data is also assigned a buffer location and stored in the buffer. When data is stored by the CPU, both the buffer and processor storage are updated if the contents of the processor storage location being altered are currently being maintained in the buffer.

The channels never access the buffer directly. They read into and write from processor storage using a eight-byte-wide path between the CPU and processor storage that bypasses the buffer. When a channel stores data in processor storage, the address array is inspected. If the data from the affected processor storage address is being maintained in the buffer, appropriate bits are set in the address array to indicate that this buffer data is no longer valid. In a Model 165, the buffer is updated instead of invalidated when a channel stores data in a processor storage location whose contents are currently in the buffer.

As in a Model 165, the entire buffer in a Model 168 can be disabled manually by a system console switch. When the buffer is disabled, all CPU fetches are made directly to processor storage and effective system execution speed is reduced. Selective buffer disabling by row performed by hardware, as described previously, is also provided for the buffer in the Model 168.

The 8K and 16K buffers are shown in Figure 20.15.2 together with their address arrays. The 8K buffer is organized in the same way in Models 168 and 165. The 8K buffer contains 64 columns of 128 bytes each. Every buffer column is subdivided into four blocks. A block is 32 bytes and can contain 32 consecutive bytes from processor storage that are on a 32-byte boundary. The 8K buffer can contain a maximum of 256 different blocks of processor storage data (four blocks per column times 64 columns). A valid trigger is associated with each buffer block and is set to indicate whether or not the block contains valid data. All valid triggers are set off during an initial program reset. There are four rows in the 8K buffer. The first row consists of block 0 of each column (64 blocks). The last row consists of block 3 of each column.

The organization of the 16K buffer in Models 168 and 165 is slightly different. In the Model 168, the 16K buffer still contains 64 columns but each column has eight blocks instead of four. In a Model 165, the 16K buffer has 128 columns of four blocks each. The approach taken in the Model 168 enables bits 21 to 31 of the storage address in an instruction to be used to address the index array for the buffer regardless of whether the storage address is virtual or real. This enables interrogation of the index array to be performed simultaneously with interrogation of the translation lookaside buffer, which is part of the dynamic address translation facility. (See Section 30:10 for more details.) There are eight rows in the 16K buffer. The first row consists of block 0 of each column (64 blocks). The last row consists of block 7 of each column.

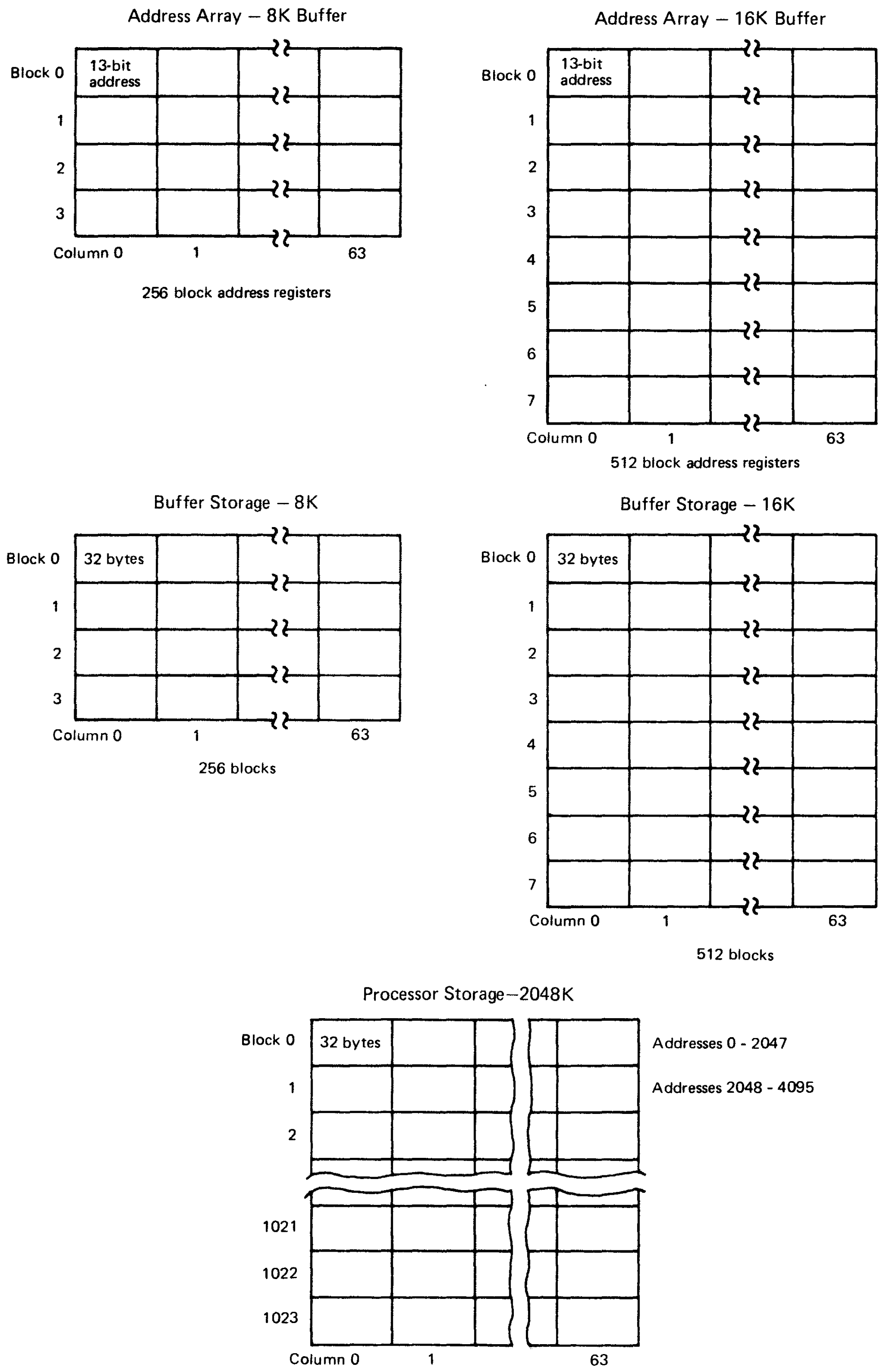


Figure 20.15.2. 8K and 16K buffer organization

Processor storage is logically divided into the same number of columns as buffer storage, which is always 64 in the Model 168. While there are four or eight blocks in a buffer column, depending on buffer size, the number of blocks in a processor storage column varies with the size of processor storage. When buffer storage is assigned, bits 21-26 of the processor storage address determine which one of the 64 columns in buffer storage is to be used. The organization of 2048K bytes of processor storage is shown in Figure 20.15.2. Any of the 1024 blocks in a given processor storage column can be placed in any one of the four (8K buffer) or eight (16K buffer) blocks in a corresponding buffer column.

Figure 20.15.2 also shows the organization of the address array for the 8K and the 16K buffer. The address array contains the processor storage addresses of the data that is currently in the buffer. A least-recently-used algorithm, similar to that used in the Model 165, is implemented in the Model 168 to determine which block within a buffer column is to be assigned when data is placed in the buffer.

Buffer and processor storage components and controls in the Model 168 are shown in Figure 20.15.3.

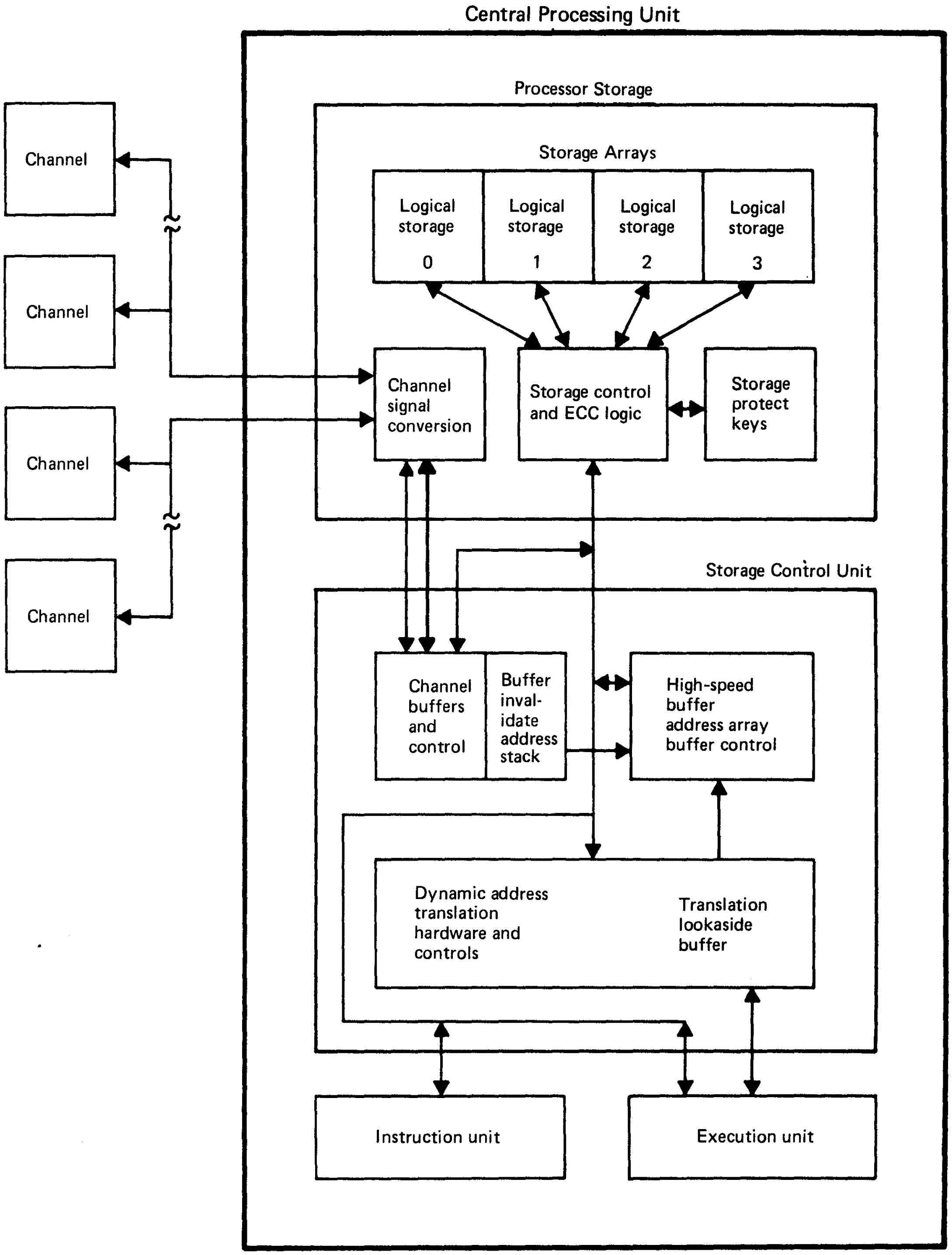

Central Processing Unit
Processor Storage
Storage Arrays
Logical storage 0
Logical storage 1
Logical storage 2
Logical storage 3
Channel
Channel
Channel
Channel
Channel signal conversion
Storage control and ECC logic
Storage protect keys
Storage Control Unit
Channel buffers and control
Buffer invalidate address stack
High-speed buffer address array buffer control
Dynamic address translation hardware and controls
Translation lookaside buffer
Instruction unit
Execution unit

Figure 20.15.3. Model 168 components and controls

20:20 CHANNELS

The number and types of channels that can be attached to Models 165 and 168 are the same. The capability of attaching up to seven standalone channels to the Model 168 is standard. Any combination of one or two 2870 Multiplexer, up to six 2860 Selector, and up to six 2880 Block Multiplexer channels can be attached to a Model 168, up to the limit of seven channels. Installation of the optional Extended Channels feature permits attachment of a maximum of twelve channels. Any combination of one (with address 0) or two (with an address from 1 to 6) 2870's, six 2860's (with addresses 1 through 6), and eleven 2880's (with addresses 1 through 11), up to the limit of twelve, can be installed.

As for a Model 165 channel configuration, the addresses and priorities of the channels present in a Model 168 configuration are established at channel installation time as indicated by the user, within the restraints specified for the Model 168. The channel buffering scheme implemented in the storage control unit is the same for Models 168 and 165.

The 2870, 2860, and 2880 channels that attach to the Model 168 are functionally and physically identical to those that attach to a Model 165. The same attachment feature that must be installed on a 2870 or a 2860 channel in order to attach the channel to a Model 165 must be installed on 2870 and 2860 channels that are to be attached to a Model 168.

While the data rates of channels that attach to the Model 168 are the same as for the Model 165, the maximum aggregate data rate that a Model 168 can sustain with minimal overrun exposure is significantly higher than that of the Model 165. The Model 168 can also have more high-speed I/O devices, such as the 2305, operating concurrently. The increased data rate is made possible by the use of a channel dual I/O bus to transfer data between the channels and the storage control unit so that the faster cycle time of Model 168 processor storage can be utilized to advantage.

The channel dual I/O bus in the Model 168 consists of bus A and bus B. Each bus provides a path between from one to six channels and a register in the storage control unit. A channel is connected to one bus or the other (not to both). Data can be transferred simultaneously on the two buses. This facility is used for input operations to transfer simultaneously data from two different channels to registers in the storage control unit.

At the time channels are installed, each is assigned to one of the two buses. A maximum of four channel frames (with up to six channels) can be attached to a bus. A maximum of seven channel frames (for a maximum of twelve channels) can be attached to a Model 168 system. Channel priorities 1, 2, 3, 9, 10, and 11 are assigned to bus A. Bus B has priorities 4, 5, 6, 7, 12, and 13 assigned. Priority 8 is not used. Simulation shows that each bus can handle a maximum data rate of 8 MB. Thus, the maximum aggregrate data rate a Model 168 can sustain is 16 MB.

The following must be observed when channel priorities are assigned:

- Priority positions 1, 2, 4, and 5 can sustain a 3-megabyte data rate concurrently. Hence, 2305 Fixed Head Storage Files (Models 1 and 2) and 2301 drums attached to 2880 channels can be assigned these priorities.

- 2301 drums attached to 2860 channels must be assigned priority positions 1 and/or 2.

- The maximum data transfer capability of a 2870 channel can be supported when it is assigned priority 1, 2, or 3.

The presence of the channel dual I/O bus in the Model 168 permits greater flexibility in the physical layout of Model 168 components since the channel frames are attached to two separate cables instead of only one, as for a Model 165. Greater flexibility in the cable lengths between channel frames attached to the same I/O bus is also provided by the Model 168.

20:25 SYSTEM CONSOLE

The 3066 Model 2 System Console for the Model 168 has the same features as the 3066 Model 1 System Console for the Model 165: a cathode ray tube and keyboard, a microfiche indicator viewer, a microfiche document viewer, a processor storage configuration panel, a system activity monitor, and a device for loading microcode and diagnostics. In addition, the store status function is implemented. (The store status function is implemented in a Model 165 II as well.)

The operator can cause the contents of the following to be placed in processor storage by pressing the new store status button on the control panel:

CPU timer - locations 216-223

Clock comparator - locations 224-231

Current PSW - locations 256-263

Floating-point registers - locations 352-383

General registers - locations 384-447

Control registers - locations 448-511

In addition to the store status button, the control panel on the 3066 Model 2 has system clear and cooling reset alarm pushbuttons, and a switch associated with the dynamic address translation feature.

20:30 STANDARD AND OPTIONAL SYSTEM FEATURES

STANDARD FEATURES

Standard features for the System/370 Model 168 are:

- BC and EC mode of operation
- Instruction set that includes binary, decimal, floating-point, and extended precision floating-point arithmetic, and System/370 instructions. Standard System/370 instructions for the Model 168 are:

  COMPARE LOGICAL CHARACTERS UNDER MASK
  COMPARE LOGICAL LONG
  INSERT CHARACTERS UNDER MASK
  *LOAD CONTROL
  *LOAD REAL ADDRESS
  MONITOR CALL
  MOVE LONG
  *PURGE TLB
  *RESET REFERENCE BIT
  *SET CLOCK

*SET CLOCK COMPARATOR
*SET CPU TIMER
SHIFT AND ROUND DECIMAL
*START I/O FAST RELEASE
*STORE CHANNEL ID
STORE CHARACTERS UNDER MASK
STORE CLOCK
*STORE CLOCK COMPARATOR
*STORE CONTROL
*STORE CPU ID
*STORE CPU TIMER
*STORE THEN AND SYSTEM MASK
*STORE THEN OR SYSTEM MASK

- Dynamic Address Translation
- Reference and Change Recording
- Instruction retry
- Interval timer (3.3 ms resolution)
- Time of day clock
- Clock comparator and CPU timer
- Monitoring feature
- Program Event Recording
- Program interruption for SSM instruction
- Expanded machine check interruption class
- ECC on processor storage
- Byte-oriented operands
- Store and fetch protection
- High-speed buffer storage - 8K bytes
- Attachment for up to seven channels
- Channel dual I/O bus
- Channel retry data in extended channel logout area
- Writable and read-only control storage
- Store status function
- Direct Control

*Privileged instructions

OPTIONAL FEATURES

Optional features for the System/370 Model 168, which can be field installed unless indicated otherwise, are:

- 3066 Model 2 System Console (required in all configurations)
- High-Speed Multiply **
- Buffer Expansion for inclusion of a 16K buffer
- 7070/7074 Compatibility **
- 7080 Compatibility **
- 709/7090/7094/7094II Compatibility **
- 2870 Byte Multiplexer Channels, 2860 Selector Channels, and 2880 Block Multiplexer Channels
- Channel Indirect Data Addressing for 2870, 2860, and 2880 channels (required when OS/VS1, OS/VS2, or VM/370 is used)
- Extended Channels (for up to twelve channels)
- Channel-to-Channel Adapter
- Integrated Storage Control
- Two-Channel Switch for Integrated Storage Control

**Not recommended for field installation

Note: Compatibility features are mutually exclusive

SECTION 30: VIRTUAL STORAGE AND DYNAMIC ADDRESS TRANSLATION

The first subsection, 30:05, discusses the needs that virtual storage and dynamic address translation in System/370 are designed to address. No previous understanding of these facilities is assumed. In this discussion, an address space is defined as a consecutive set of addresses that can be used in programs to reference data and instructions. System operation in IBM-supplied virtual storage environments is explained conceptually, without use of all the terminology new to such an environment.

The general advantages of IBM-supplied virtual storage operating systems are presented also. Included in this subsection are those that apply to OS/VS1 and OS/VS2. Additional advantages of virtual storage that are specific to a particular IBM-supplied operating system are discussed in the optional supplement for that operating system.

The last portion of subsection 30:05 defines the terminology associated with virtual storage and dynamic address translation hardware. The terminology included is that common to the IBM-supplied operating systems that support a virtual storage environment for System/370. However, specific references to DOS/VS are not made where a difference between DOS and OS exists, since DOS/VS does not support the Model 168. Terms unique to a particular operating system are defined in the optional supplement that describes that operating system.

Subsection 30:10 describes in detail the implementation and operation of dynamic address translation and channel indirect data addressing hardware in the Model 168. Other hardware items associated with dynamic address translation, such as reference and change recording, are discussed as well.

The last subsection, 30:15, discusses the new factors that affect system performance in a virtual storage environment. The information presented is related to efficient installation and utilization of an IBM-supplied virtual storage operating system.

The optional programming systems supplements (Sections 90 to 110) assume knowledge of the entire contents of Section 30. This entire section applies to the Model 165 II as well as the the Model 168, except where differences are noted.

30:05 VIRTUAL STORAGE CONCEPTS, ADVANTAGES, AND TERMINOLOGY

THE NEED FOR LARGER ADDRESS SPACE

The past and present rapid growth in the number and types of data processing applications being installed has led to an increasing demand for more freedom to design applications without being concerned about, or functionally constrained by, the physical characteristics of a particular computer system--system architecture, I/O device types, and processor storage size. As program design and implementation become easier, they can enable more rapid installation of applications so that the benefits of data processing can be achieved sooner.

The design of System/360 and OS MFT and MVT allowed programmers to be less concerned than before about specific CPU architecture and I/O device types when designing and implementing applications by (1) providing a compatible set of CPU models ranging in size from small to large scale, (2) providing a variety of high-level languages with greatly expanded capabilities, including a new language (PL/I), (3) providing comprehensive data management functions, including support of I/O device independence where data organization and the physical characteristics of devices permitted, and (4) supporting dynamic allocation of system resources (channels, I/O devices, direct access space, and processor storage).

While System/360 and OS represented major steps toward giving programmers a larger measure of system configuration independence, constraints that resulted from the necessity to design applications to fit within the amount of processor storage available still existed. In addition, although System/360 models provided more and less costly processor storage than was previously available, increasingly larger amounts of processor storage began to be required as the use of high-level languages increased, the usage and level of multiprogramming increased, the functions supported by operating system control programs expanded, and applications that require relatively larger amounts of processor storage (such as teleprocessing and data base) were designed and installed more frequently.

The requirement for more processor storage is still growing. The new applications being developed and installed tend to have larger and larger storage design points in order to provide the functions desired. More processor storage is also required for I/O buffer areas to achieve maximum capacity and performance for sequential operations using new System/370 direct access devices with significantly larger track capacities. Larger blocking of tape records, which requires larger I/O buffers, also results in increased tape reel capacity and decreased tape processing time. As a result, System/370 models provide significantly more processor storage than their predecessor System/360 models and offer it for a lower cost.

The availability of more processor storage, however, has not relieved all the constraints associated with processor storage. Applications still must be tailored to the amount of processor storage actually available in a given system even though storage design points (partition and region sizes) can be larger than they were previously.

Consider the following situations that can occur in installations:

1. An application is designed to operate in a 50K processor storage area that is adequate to handle current processing needs and that provides room for some expansion. Some time after the application is installed, however, maintenance changes and the addition of new functions cause one of the programs in the application to require 51K and another to require 52K. Installation of the next processor storage increment cannot be justified on the basis of these two programs, so time must be spent restructuring and retesting the programs to fit within 50K.

2. An existing application has programs with a planned overlay structure. The volume of transactions processed by these programs has doubled and better performance is now required. Additional processor storage is installed. However, the overlay programs cannot automatically use the additional storage. Therefore, reworking of the overlay programs is required to take them out of planned overlay structure and, thereby, achieve the better performance desired.

3. A low-volume, terminal-oriented, simple inquiry program that will operate for three hours a day is to be installed. If the program is written without any type of overlay structure, it will require 60K of processor storage to handle all the various types of

inquiries. However, because of a low inquiry rate, only 8K to 12K of the total program will be active at any given time. In order to justify its operational cost, considerable additional program development time is spent designing the inquiry program to operate with a dynamic overlay structure so that only 12K of processor storage is required for its execution.

4. A multiprogramming installation has a daily workload consisting primarily of long-running jobs. There are also certain jobs that require a relatively small amount of time to execute. The times at which these jobs must be executed is unpredictable; however, when they are to be run, they have a high completion priority. While it is desirable to be able to initiate these high-priority jobs as soon as the request to execute them is received, this cannot be done because long-running jobs are usually in operation. Hence, a certain time of day is established for initiating high-priority jobs and the turnaround time for these jobs is considerably longer than is desired.

5. A series of new applications are to be installed that require additional computing speed and twice the amount of processor storage available in the existing system. The new application programs have been designed and are being tested on the currently installed system until the new one is delivered. However, because many of the new application programs have storage design points that are much larger than those of existing applications, testing has to be limited to those times when the required amount of processor storage can be made available. Although another smaller scale model is also installed that has time available for program testing, it cannot be used because it does not have the amount of processor storage required by the new application programs. In addition, although the smaller scale model now provides backup for the currently installed larger scale model, the smaller scale model cannot be used to back up the new system because of processor storage size limitations.

6. A large terminal-oriented application is to be operative during one entire shift. During times of peak activity, four times more processor storage is required than during low-activity periods. Peak activity is experienced about 20 percent of the time and low activity about 40 percent. The rest of the time, activity ranges from low to peak. Allocation of the peak activity processor storage requirement for the entire shift cannot be justified and a significantly smaller storage design point is chosen. As a result, a dynamic program structure must be used, certain desired functions are not included in the program, and response times during peak and near-peak activity periods are increased above that originally planned.

   In this installation, most of the batched jobs are processed during the second shift. However, there is also a need to operate the large terminal-oriented application for a few hours during second shift. This cannot be done because the system does not have the amount of processor storage required for concurrent operation of the batched jobs and the terminal program (which must have its storage design point amount allocated even though that amount of processor storage would not be required during second shift operations). The large amount of additional processor storage required to operate the terminal program for only a portion of the second shift cannot be justified.

7. An application program with a very large storage design point is executed only once a day as a batched job. A significant benefit would result from putting the program online to a few terminals during the morning hours. However, the program continues to be

run as a batched job because it is very large and would be made larger by putting it online. The large amount of additional processor storage required to operate the program concurrently with the existing morning workload cannot be justified.

8. A terminal-based application has been installed on a full production basis for several months. During this period, the benefits accrued from the online application have encouraged the gradual addition of several more terminals, and peak activity is considerably higher than it was initially. Because growth has been gradual, much additional programming time (significantly more than is required to maintain batch-oriented applications) has to be spent periodically restructuring the terminal-based application program to handle the increasing volume of activity.

9. An online application is currently active during an entire shift and operates concurrently with batched jobs. It would be advantageous to install a second terminal-oriented application that would operate concurrently with the existing workload during the entire shift. However, the amount of processor storage that would have to be dedicated to each online application for the entire shift in order to handle its peak activity is very large, and times of peak activity for the two applications do not completely overlap. Because so much processor storage would be unused during a large portion of the shift if both online applications were always active, installation of the second online application is difficult to justify.

In the situations described, processor storage is a constraining factor in one way or another and the constraints highlighted can apply in some degree to all systems regardless of their scale (small, intermediate, large) or processor storage size. The fact that larger, less expensive processor storage is now available on System/370 models does not remove these constraints for two major reasons.

First, once a storage design point has been chosen for an application, whether the design point is relatively large or small, the application is dependent on that processor storage size for its operation. The application cannot execute in less than its design point storage amount, nor can it take advantage of additional available processor storage without being modified (unless it has been specifically structured to use additional storage as, for example, are most IBM-supplied language translators).

Second, although processor storage has become less costly, it still is a resource that should be used efficiently because of its importance in the total system operation. Thus, when storage design points are chosen, tradeoffs among processor storage cost, application function, and system performance are often made. Making applications fit within the storage design points selected becomes the responsibility of application designers and programmers. This situation is made more difficult by the fact that for many applications an optimum storage design point cannot be determined until the application is written and tested using expected transaction volumes.

The significance of processor storage restraints should be evaluated in light of the following trends evidenced by new types of applications: (1) the total amount of storage required to support their new facilities continues to grow larger, (2) the storage they actually require for operation during their execution is tending to become more variable, and (3) it is becoming as desirable to install many of these new applications on smaller scale systems with relatively small maximum processor storage sizes and low volume requirements as it is to install them on larger scale systems. Reduction of the constraining factors currently imposed by processor storage is, therefore, a necessary step in making new applications easier and less costly to install and available to a wider range of data processing installations.

Given the existing processor storage restraints on application design and development and the storage requirements that are becoming increasingly more characteristic of many of the new types of applications, it becomes advantageous to allow programmers to design and code applications for a larger address space than they currently have. That is, programmers should be able to use as much address space as an application requires so that special program structures and techniques are not required to fit the application into a given storage size. Programmers can then concentrate more on the application and less on the techniques of programming. In addition, the size of the address space provided should not be determined by processor storage size, as it is in OS MFT and MVT, so that the address space can be larger than the processor storage available.

A larger address space should be provided, therefore, by a means other than making processor storage as large as the address space desired. This requirement can be satisfied by providing programmers with an address space (called virtual storage) that is supported using online direct access storage and dynamic address translation hardware. This approach also offers the advantage of supporting a larger address space for a lower cost than if larger processor storage is used, since direct access storage continues to be significantly less expensive per bit than processor storage. In addition, dynamic address translation hardware offers functional capabilities that large processor storage alone cannot provide.

VIRTUAL STORAGE AND DYNAMIC ADDRESS TRANSLATION CONCEPTS

Virtual storage is an address space the maximum size of which is determined by the addressing scheme of the computing system that supports it rather than by the actual number of physical processor storage locations present in the computing system. In System/370, for example, which uses a 24-bit binary address, a virtual storage as large as 16,777,216 bytes can be supported. When virtual storage is implemented, the storage that can be directly accessed by the CPU, normally called processor or main storage, is referred to as real storage.

The concept of virtual storage is made possible by distinguishing between the names of data and instructions and their physical location. In a virtual storage environment, there is a distinction between address space and real storage space. Address space (virtual storage) is a set of identifiers or names (virtual storage addresses) that can be used in a program to refer to data and instructions. Real storage space is a set of physical storage locations in the computer system in which instructions and data can be placed for processing by the CPU. The number of addresses in the two spaces need not be the same, although both spaces begin with address zero and have consecutive addresses. The programmer refers to data and instructions by name (virtual storage address) without knowing their physical location.

When virtual storage is not implemented, there is, in effect, no differentiation between address space and real storage space. The address space that can be used in programs is identical in size to the real storage space available and the address in an instruction represents both the name and the location of the information it references.

In a virtual storage environment, therefore, the address space available to programmers is that provided by the virtual storage size implemented by a given system--not the address space provided by the real storage available in the given system configuration. In OS/VS1 and OS/VS2, virtual storage rather than real storage is divided into consecutively addressed partitions or dynamically allocated regions for allocation to problem programs. The fact that storage addresses in executable programs are virtual rather than real does not affect the way in which the programmer handles addressing. In System/370, for example, an Assembler Language programmer assigns and loads base registers and manipulates virtual storage addresses in a program just as if they were real storage addresses.

Virtual storage is so named because it represents an "image of storage" rather than physical processor storage. Since virtual storage does not actually exist as a physical entity, the instructions and data to which its virtual storage addresses refer, which are the contents of virtual storage, must be contained in some physical location.

In OS/VS1 and OS/VS2 environments, the contents of virtual storage are divided into a portion that is always present in real storage, namely, part of the control program, and another portion that is not always present in real storage. The instructions and data that are not always present in real storage must be placed in locations from which they can be brought into real storage for processing by the CPU during system operation. This requirement is met by using direct access storage to contain this portion of the contents of virtual storage (see Figure 30.05.1). The amount of direct access storage required to support a given amount of virtual storage varies by operating system, depending on how direct access storage is organized and allocated.

In addition, a mechanism is required for associating the virtual storage addresses of instructions and data contained in direct access storage with their actual locations in real storage when the instructions and data are being processed by the CPU. This requirement is met by using dynamic address translation (DAT) hardware in the CPU to associate virtual storage addresses with appropriate real storage addresses.

With this design, a system can support an address space that is larger than the actual size of the real storage present in the system. This is accomplished by bringing instructions and data from direct access storage into real storage only when they are actually required by an executing program, and by returning altered instructions and data to direct access storage when the real storage they occupy is needed and they are no longer being used. At any given time, real storage contains only a portion of the total contents of virtual storage.

Such a design is made practical by the fact that the logical flow of processing within the majority of programs is such that the entire program need not be resident in real storage at all times during execution of the program. For example, initialization and termination routines are executed only once during the operation of a program. Any exception-handling procedure, such as an error routine, is required only if the exception condition occurs. A program that handles a variety of transaction types (whether batch or online oriented) need have resident at any given time only the transaction routine required to process the current transaction type. It is this property of programs that has enabled planned overlay and other dynamic program structures to be used successfully in nonvirtual storage environments when the amount of processor storage available was not large enough. As indicated previously, this variable storage requirement characteristic of programs tends to be even more pronounced in new types of applications and in online environments in which processing is event-driven.

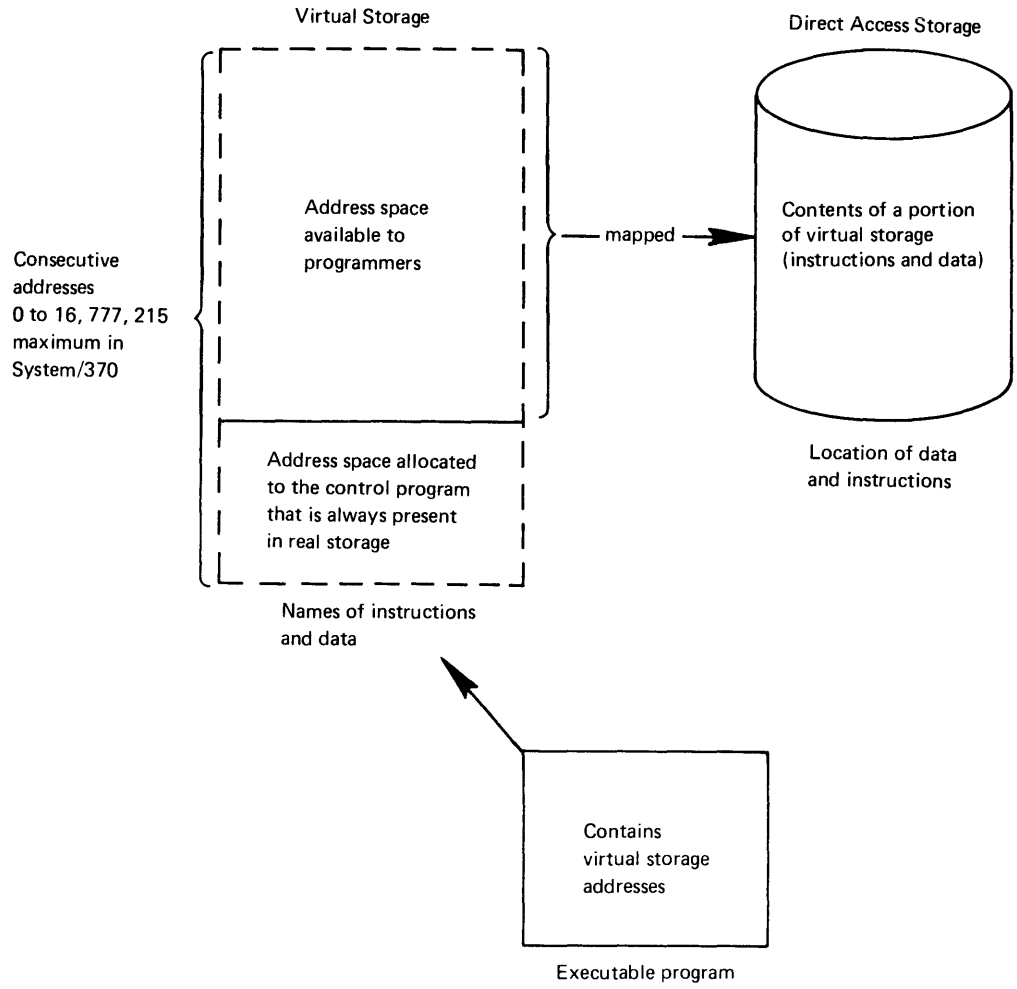

Virtual Storage
Direct Access Storage
Address space available to programmers
mapped
Contents of a portion of virtual storage (instructions and data)
Consecutive addresses 0 to 16, 777, 215 maximum in System/370
Address space allocated to the control program that is always present in real storage
Location of data and instructions
Names of instructions and data
Contains virtual storage addresses
Executable program

Figure 30.05.1. Names and location of instructions and data in a virtual storage environment

For the purpose of resource management in a virtual storage environment, virtual storage and its contents, direct access storage used to contain a portion of the contents of virtual storage, and real storage are divided into contiguous fixed-length sections of equal size. Once a program has been fetched from a program library and initiated, instructions and data within a program are transferred between real storage and direct access storage a section at a time, during program execution. A section of an executing program is brought into a real storage section only when it is required, that is, only when a virtual storage address in the section is referenced by the executing program. A program section that is present in real storage is written in a direct access storage section only when the real storage assigned to it is required by another program section and only if the section has been changed.

A virtual storage operating system control program monitors the activity of the sections of all executing programs and attempts to keep the most active sections in real storage, leaving the least active sections in direct access storage. Figure 30.05.2 illustrates the relationship of virtual storage, direct access storage, and real storage without regard to a specific virtual storage operating system implementation.

The division of a program and its data into sections, and the transfer of these sections between direct access storage and real storage during program execution is handled entirely by the virtual storage operating system without any effort by the programmer. When a planned overlay or dynamic overlay program structure is used, the programmer is responsible for dividing the program and its data into phases, determining which phases can be present at the same time in the amount of real storage available (partition or region), and indicating when phases are to be loaded into real storage during processing.

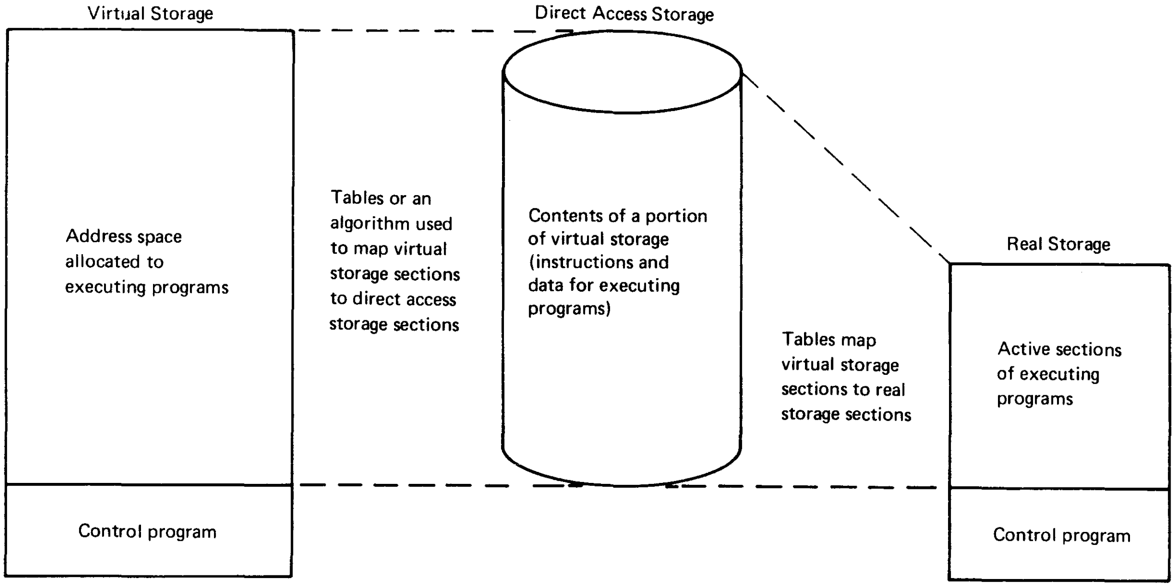

Virtual Storage
Direct Access Storage
Address space allocated to executing programs
Tables or an algorithm used to map virtual storage sections to direct access storage sections
Contents of a portion of virtual storage (instructions and data for executing programs)
Tables map virtual storage sections to real storage sections
Real Storage
Active sections of executing programs
Control program
Control program

Figure 30.05.2. Relationship of virtual storage, direct access storage, and real storage

While a virtual storage up to 16 million bytes in size can be addressed by any System/370 model with DAT hardware, the virtual storage size that can be effectively implemented by a given system is affected by (1) the amount of real storage present, (2) the amount of direct access storage space available to contain the contents of virtual storage, (3) the speed of the direct access storage devices containing virtual storage contents and contention for these devices or the channels to which they are attached, (4) the speed of the CPU, and (5) the characteristics of the programs operating concurrently. Hence, the amount of real storage required to effectively implement a specific amount of virtual storage can vary by system, depending on the characteristics of the applications in the workload and the performance desired, as is discussed in Section 30:15.

Once a program section has been loaded into real storage, its virtual storage addresses can be translated when they are referenced. Dynamic address translation hardware is the mechanism that translates the virtual storage addresses contained in instructions into real storage addresses during instruction execution. Address translation is accomplished in System/370 using a hardware-implemented table lookup procedure that accesses tables contained in real storage. These tables, which are maintained by control program routines, (1) define the amount of virtual storage supported and allocated, (2) indicate whether or not any given program section is currently present in real storage, and (3) contain the addresses of real storage sections allocated to the program sections that are currently present in real storage.

During the execution of each instruction, address translation is performed on any virtual storage address in the instruction that refers to data or to an instruction. Translation occurs after the 24-bit effective virtual storage address has been computed by adding base, displacement, and, if any, index values together, as usual. The result of the address translation is a 24-bit real storage address designating the location containing the data or instruction referenced by the virtual storage address in the instruction. The virtual storage addresses in channel programs (CCW lists) are not translated by channel hardware during channel program execution; therefore, programmed translation is required prior to initiation of a channel operation.

In reality, DAT hardware provides dynamic relocation of the sections of a program during its execution. This capability is not provided by OS MFT and OS MVT, which support program relocation at link-edit and program load time only. Once a program has been loaded into an area of real storage by the program fetch routine, these operating systems cannot relocate the program to another area of real storage during its execution. Thus, an entire program or a portion of a program cannot be written on direct access storage during execution and later reloaded into different real storage locations to continue execution. Once loaded, therefore, a program is bound during its execution to its initially allocated real storage addresses. In a virtual storage environment, a program is bound only to the virtual storage addresses it was assigned during loading.

The dynamic relocation provided by DAT hardware eliminates, for most programs, the need for allocating and dedicating a contiguous area of real storage to an entire program for the duration of its execution, a requirement for all programs in MFT and MVT. (As discussed later in this subsection, some programs cannot operate in the manner being described, that is, with sections transferred only as required between direct access storage and real storage.) In a virtual storage environment, real storage is no longer divided into contiguously addressed partitions or dynamically allocated regions that can contain one executing job step (program) at a time.

Further, when real storage is allocated to a section of an executing program, the real storage is not dedicated to that program section for the duration of program execution. Concurrently executing programs can dynamically share the same real storage sections. That is, is general, the real storage available for allocation to executing programs can be allocated to any program section as needed. When a section of an executing program must be loaded, any available section of real storage can be assigned (subject to certain restrictions imposed by operating-system-dependent real storage organizations). When the program section is no longer required, it can be written in direct access storage, if it has been altered, and the real storage assigned to it can be made available for allocation to another section of the same program or to a section of another program.

The assignment of real storage sections is handled entirely by the operating system, which also keeps account of which sections of concurrently operating programs are the most active. The operating system does not attempt to allocate a given amount of real storage to each executing program. It merely allocates real storage to those sections it determines are the most active, without taking into account the particular program to which the active section belongs.

DAT hardware, therefore, provides more than translation from address space (virtual storage) to real storage space. It provides the capability of implementing dynamic real storage management that requires no effort on the part of the programmer and significantly less CPU time than programmed address translation during program execution. (The large amount of CPU time required to translate addresses during program execution using programmed means has precluded implementation by IBM of an operating system that supports such programmed dynamic address translation.) Much of the real storage utilization preplanning required for OS MVT and MVT environments in order to use real storage effectively can be eliminated in a virtual storage environment. Dynamic real storage management capability is another advantage the technique of using direct access storage and DAT hardware to support a larger address space has over using larger real storage to provide a larger address space.

Another capability made available by the implementation of large address space using direct access storage and dynamic address translation is that of supporting more than one virtual storage with only one system. Multiple virtual storages can be used to support multiple virtual machines. A discussion of virtual machines is contained in Virtual Machine Facility/370 (VM/370): Introduction (GC20-1800).

The use of virtual storage and DAT hardware to enable programs to operate in less real storage than the total storage requirement of the programs can also offer better performance potential than the technique of using a planned overlay program structure. When a planned overlay program executes in MFT or MVT, considerable time can be spent executing the overlay supervisor in order to perform programmed address translation (relocation) when a program phase is loaded. In addition, more efficient real storage utilization may be achieved in a virtual storage environment, since the control program reacts to changing processing needs and only portions of the program that are actually required are loaded (all phases of an overlay program may not be the same size and all code within a phase may not be used when the phase is loaded). Once a planned overlay program has been structured to handle the currently required set of program phases efficiently, it cannot automatically adapt to a change in the set of program phases required or to a change in the activity of the required set of phases.

In a virtual storage environment, the performance of the system can be directly affected by the amount of time spent transferring program sections between direct access storage and real storage. Satisfactory system performance is achieved when each of the concurrently executing programs has enough real storage dynamically allocated to it such that the need for transferring program sections into and out of real storage is kept at an acceptable level.

As previously mentioned, most programs can be structured such that processing activity is localized in one area of the program or another during time intervals rather than equally spread over the entire program. In other words, at any given time period during execution of the program, only a subset of the entire program need be referenced. This is sometimes called the "locality of reference" characteristic of programs. Therefore, a program achieves satisfactory performance when its most active sections in any given time interval remain in real storage and there is a limited amount of program section transfer activity.

Most programs require a certain minimum amount of real storage in which to execute in order to achieve satisfactory performance. If such programs operate with less than their minimum requirement dynamically allocated, program section transfer activity increases and performance degradation can occur. The minimum real storage requirement of a program is related to the amount of real storage required by the most active sections of the program. Because of the locality of reference characteristic of most programs, the minimum real storage requirement of a program for satisfactory operation frequently can be less than its total storage requirement. This fact enables an operating system to efficiently support a virtual storage that is larger than the real storage actually present in the computing system.

A virtual storage environment, therefore, enables most programs to be independent of real storage size to a large degree. A program can execute using varying amounts of dynamically available real storage without being modified. The amount of real storage dynamically available to a program during its execution primarily affects its performance, to the extent that program section transfer activity is affected, rather than its capability to be executed. For example, while a given 200K language translator might be able to operate with an average of 100K of real storage dynamically available to it during its operation, the time required to compile a program under these conditions might be unacceptable. Alternatively, the performance desired might be achieved if an average of 130K is dynamically available to the language translator while it operates. Without a virtual storage operating system, the 200K language translator might not be used at all because of its design point size.

In addition to the requirement for larger address space, there is still a requirement for larger real storage sizes in order to meet the functional and performance needs of the larger, more complex, multiprogramming environments. The availability of large lower cost real storage for the Model 168 and the real storage independence that a virtual storage environment offers provide new flexibility in tradeoffs among real storage cost, function, and individual program or total system performance.

GENERAL ADVANTAGES OFFERED BY IBM OPERATING SYSTEMS THAT SUPPORT A VIRTUAL STORAGE ENVIRONMENT

Each of the IBM operating systems that supports a virtual storage environment for System/370 models using dynamic address translation offers the capability of using address space that is larger than that provided by available real storage, and each supports dynamic real storage management that is transparent to the user. As a result, these operating systems offer certain general potential advantages that do not depend on their unique features. The implementation of virtual storage also provides benefits that are specific to each of these operating systems because of their design and the particular functions they support. The following discusses the potential advantages of virtual storage and dynamic address translation that are common to OS/VS1 and OS/VS2 environments.

The general advantages of virtual storage operating systems are the potential they offer for:

- Increased application development

- Expanded operational flexibility

- System performance improvement

A virtual storage operating system can facilitate more rapid development of new applications because, by removing most existing real storage restraints on application design, it can help improve the productivity of programmers. Specifically, a virtual storage operating system has characteristics that can be used to reduce the effort, time, and cost associated with application design, coding, testing, and maintenance. This makes the installation of new applications more readily justifiable and encourages the addition of new functions to existing applications. The potential advantage of improved operational flexibility is made possible by the greater independence of applications from real storage size. Enhanced system performance can result from improved real storage utilization. While these latter two benefits have their own individual value, they too, either indirectly or directly, ease the installation of new applications.

Potential for Increased New Application Development

The following capabilities are characteristic of a virtual storage operating system environment:

- Greater flexibility in the design of applications is possible.

  Larger programs can be written without the necessity of using planned overlay techniques or other dynamic program structures designed to fit programs into the amount of real storage available. The need for intermediate (or working) data sets is reduced or eliminated because tables, relatively small data groups, etc., that are placed on direct access storage because of real storage limitations, can become part of the program and will be brought into real storage automatically as required. Program planning, coding, and testing time can be reduced by elimination of the use of these programming techniques and other real storage management facilities, which also require additional programming knowledge and skill. Also avoided is the restructuring of application programs after they have been written, because they are larger than the real storage available for their execution. Hence, applications can become operational more quickly.

  Open-ended, straightforward application design is possible and more comprehensive programs can be written. An application can be segmented into a series of programs according to its logical flow instead of according to the functions that can be performed in the specific amount of real storage available to each step in the application. Programming and processing duplication inherent in the approach of using two or more job steps to perform one logical process is thereby avoided.

  Additional programming facilities can become available that otherwise could not be used because of real storage limitations. Specifically, full function high-level language translators, which offer more capabilities than their subset versions (such as additional debugging facilities and performance options) but which also have larger storage design points, can be used because they can operate in a virtual storage environment using less real storage than their design point requirement.

- Preproduction testing of larger than average application programs can be increased if enough virtual storage can be made available to enable them to run during normal testing periods. Turnaround time during testing can be reduced.

  In a nonvirtual storage environment such programs are usually grouped together and executed only at certain times when their larger design point storage requirements can be made available.

- Fine tuning of application programs to achieve performance improvements, when necessary, can be delayed until after the application is in production. This capability enables an application to become operative sooner.

- Startup costs for new applications may be reduced.

  A new application can be developed and tested on the existing system, assuming the required I/O devices are present in the configuration, before the additional real storage the application requires for performance on a production basis is actually installed. When the application is ready for production, the additional real storage required can be added to the system. In some cases it may be possible to operate the application on a production basis on the existing system without adding real storage initially, because during the startup period, transaction volume is very low. As the volume grows, real storage can be added to achieve better performance.

- Growth of existing applications and the maintenance of operational programs is simplified.

  Because of the removal of most real storage restraints, new functions can be more easily and more rapidly added to most existing applications. Program expansion because of added functions or maintenance changes does not require the use of overlay techniques, multiple job steps, etc., when the size of the extended program exceeds the original storage design point size.

  In general, alteration and debugging of nonoverlay programs are also easier than alteration and debugging of programs with planned overlay or dynamic structures.

- Application programs whose real storage requirements, based on transaction volume and complexity, vary widely during their execution may be justified, designed, and installed more easily.

  Design, coding, and testing time can be reduced because dynamic storage management is automatically provided by the operating system. Time and effort need not be spent structuring such programs to use available real storage dynamically to support the functions and/or response times required.

- Design and installation of one-time, low-usage, or low-volume programs of very large storage size are more easily justified. Existing applications in these categories that currently operate in a batch environment can also more easily be altered to operate online, a growth step that might not be justifiable in a nonvirtual storage environment.

- Applications can be installed on a trial basis for the purpose of observing and evaluating their functions and their operation.

  Most IBM-supplied application program products can be temporarily installed on an existing system, assuming the required I/O devices are present. The additional hardware resources that may be required to operate the application on a production basis can be added later, when the application is permanently installed.

- The benefits of the functions provided by many IBM-supplied application program products with larger storage design points can be realized using smaller amounts of available real storage.

  It may be difficult to justify the real storage required to install a relatively large storage design point application on a system to handle a low volume of transactions, even though the functions provided by the application are very desirable. In a virtual storage environment, such an application can execute using that amount of dynamically available real storage required to satisfy the desired performance requirements for the low volume of activity.

Potential for Additional Operational Flexibility

The reduction of real storage restraints makes most applications more independent of the real storage size of a system configuration and permits most applications to be processed on systems with varying amounts of available real storage without program modification. Dynamic real storage management reduces the amount of job stream and operations preplanning that is normally done to use real storage as efficiently as possible in a multiprogramming environment. The following benefits can be the result:

- A system can back up another system even though it has less real storage than the system it backs up.

  A smaller scale system with the appropriate I/O configuration can provide backup for a larger scale system if necessary. (Performance experienced on the backup system may vary from that normally achieved depending on the two system configurations involved.)

- A single design and one operating procedure can be used for an application that is to operate on multiple systems with varying amounts of real storage, as long as the virtual storage required is supported by all the systems.

  When data processing is decentralized among multiple installations with systems that have different amounts of real storage, one location can design, implement, and maintain an application that can be used by other installations. Duplication of this type of effort can be minimized or eliminated.

- Most applications can be tested on systems with less real storage than the one on which they will run in a production environment, as long as the required amount of virtual storage is supported.

- Growth to a larger real storage configuration can be easier.

  Real storage can be added to an existing system to improve system performance (by the reduction of program section transfer activity) without the necessity of modifying existing application programs so that they take advantage of additional real storage. Additional real storage (up to a maximum of their design point size) is automatically used by programs that operate in a virtual storage environment.

- Operators need not perform certain procedures that are solely related to efficiently managing real storage.

  The operator is concerned with the division of virtual storage and therefore need not change partition sizes at various times (in OS/VS1, for example) for the purpose of making storage available for larger than average jobs. (An installation can define virtual storage partitions that are larger than those currently defined in the OS MFT environment, and the partitions can be made big enough to contain the largest existing or currently planned storage design point programs.) Similarly, in an OS/VS2 environment, the operator no longer need start long running jobs at certain points in time to ensure that available real storage is fragmented as little as possible.

- Priority jobs whose need to be processed cannot be predicted can be scheduled when required.

  A nonvirtual storage environment does not provide the capability of effectively handling the scheduling of high-priority jobs on a random basis. Hence, this type of job is not permitted to exist in an installation, or such jobs must be scheduled to operate only at certain times. In a virtual storage environment, a high-priority virtual partition can be defined in an OS/VS1 environment and reserved for the purpose of processing only high-priority jobs. Except for that required for certain tables, real storage is not required for this partition until a job is actually scheduled. In an OS/VS2 environment, an initiator with a special class can be started that will handle only high-priority jobs. This can be done in MVT as well but because of the possibility of real storage fragmentation, there is no assurance that the high-priority job can be started.

Potential for Performance Improvement

The improved real storage utilization made possible by the use of dynamic address translation hardware can have a positive effect on the performance of a system that handles a job mix whose use of real storage varies considerably while it is being processed. The extent of the performance improvement depends on the types of applications involved and the current utilization of system resources. Therefore, the amount of performance gain, if any, that may be achieved is highly variable by installation. Environments with the greatest potential for improved performance are as follows:

- Batch-oriented multiprogramming environments with application programs of widely varying real storage requirements.

  Real storage may not be most efficiently used in such an environment because (1) real storage can become fragmented when regions are dynamically allocated and freed or (2) it is difficult to divide real storage into a set of areas that is optimum for all programs when real storage is partitioned. (Consider the inefficient use of real storage in an 80K partition allocated for assemble, link-edit, and test jobs in which a 80K language translator, a 44K linkage editor, and problem programs no larger than 60K execute.) In addition, real storage is not efficiently used when the real storage requirement of a given program, based on transaction mix or volume, varies widely, and the amount of real storage that is allocated is designed to handle the peak requirement. (This is typically true of graphics applications, for example.) Further, real storage assigned to a program is not productively used during the time the program is waiting for a human response, such as for the operator to locate and/or mount a volume or to make a decision and enter a message on the console, or during the time required to quiesce the system in order to change partition definitions, start high-priority jobs, or start a teleprocessing program in high real storage.

  In a virtual storage environment, in which all concurrently executing job steps share real storage dynamically and use real storage only when it is actually required for program execution, real storage is more efficiently used. Hence, if real storage currently is the restraint, a given real storage size might be capable of supporting a higher level of multiprogramming than can be achieved without the use of dynamic storage management (assuming other required resources such as CPU time, I/O devices, and channels, are available). For example, installation of a large storage design point, terminal-based application to handle only a few terminals might be possible. Alternatively, a higher level of multiprogramming might be supported by the addition of a smaller real storage increment than would otherwise be required.

  System performance may also be improved if more efficient use of available real storage enables additional heavily used functions to be made resident instead of transient or allows the incorporation of performance-oriented options in the control program. This improvement can apply to environments with batch and online operations, as well as to batch-only multiprogramming environments.

- Multiprogramming environments with a mixture of batch-oriented and terminal-based applications.

  While the real storage required for the communication control portion of a teleprocessing application remains constant, terminal-based processing programs are typically subject to wide variations in the amount of real storage they require during their execution because the transaction mix being handled concurrently varies, the activity of each terminal online varies, or the number of terminals operating concurrently changes. In order to provide the functions desired, ensure the capability of handling peak activity periods and maximum transaction type mixes, and guarantee a given response during times of peak activity, a certain amount of real storage is required. This peak requirement is generally significantly more than is needed during periods of medium and low activity. Allocation of the maximum storage requirement results in inefficient use of real storage, since unused real storage dedicated to any terminal program cannot be used by other concurrently operating batched or terminal-oriented jobs in a nonvirtual storage environment. In addition, it is usually difficult, and sometimes impossible, to effectively preplan real storage usage for an online application.

  Dynamic real storage management in a virtual storage environment automatically provides a much more efficient method of allocating real storage in such an environment. Real storage is not divided into that which can be used only by the terminal-based program(s) and that which can be used only by batched jobs. During times of peak terminal activity, the active sections of terminal-oriented processing programs with a higher priority are automatically allocated real storage, making less real storage available to the lower priority batched jobs in execution at that time. During periods when terminal activity is relatively low, real storage not used by any terminal program is available for assignment to the active sections of executing batched jobs. Such an environment is represented conceptually in Figure 30.05.3.

  In existing mixed batch and online-oriented installations, dynamic real storage management allows programming techniques that can improve the performance of the online application. This improvement can be in the form of better response for existing terminals or the ability to support more terminals. A given online application may also be able to support a higher level of multiprogramming, as a result of better real storage utilization, without any additional programming effort (more TSO regions, for example). A virtual storage environment also makes the concurrent operation of multiple terminal-based applications more practical.

Figure 30.05.3 shows sample allocations of real storage to two batched jobs and two terminal-oriented jobs in a multiprogramming environment during low, medium, and peak activity points in time. Job priority from high to low is TP2, TP1, BJ2, BJ1. For simplicity, virtual and real storage are shown to be totally allocated at all times and no particular virtual storage operating system (OS/VS1 or OS/VS2) is assumed, since the concepts illustrated apply to both, regardless of differences in the way virtual storage is allocated by these operating systems. Real storage is shown to be contiguously allocated to each job in high-to-low priority sequence. This is done only to illustrate the relative amount of real storage the control program has dynamically allocated to each program during the instant shown. In reality, the total amount of real storage allocated to an executing program at any given time is usually not contiguous in a virtual storage environment.

In addition, during times of low terminal program activity, it may be possible to support a higher level of batched job multiprogramming, which is not shown in the figure.

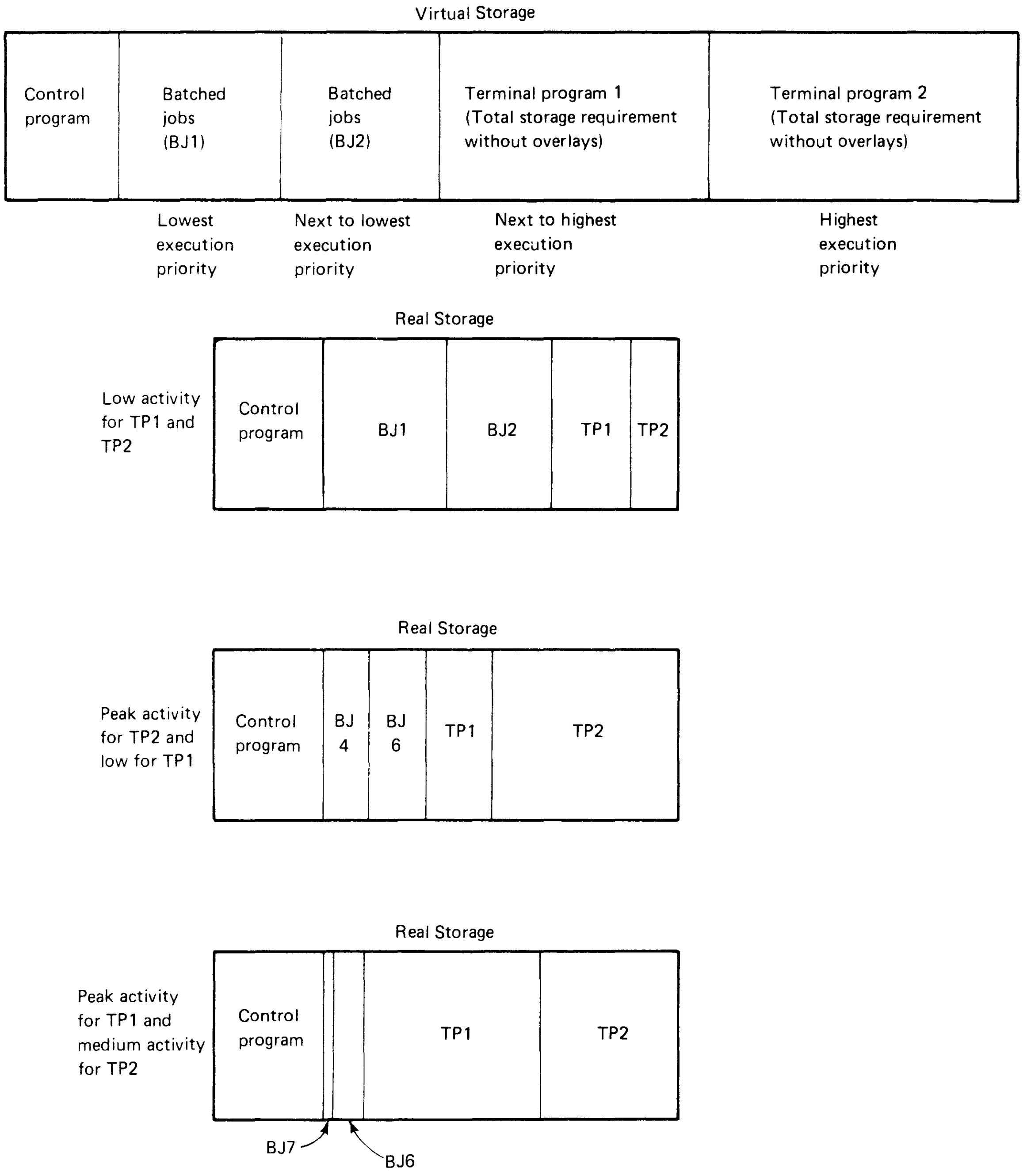

Virtual Storage
Control program
Batched jobs (BJ1)
Batched jobs (BJ2)
Terminal program 1 (Total storage requirement without overlays)
Terminal program 2 (Total storage requirement without overlays)
Lowest execution priority
Next to lowest execution priority
Next to highest execution priority
Highest execution priority
Real Storage
Low activity for TP1 and TP2
Control program
BJ1
BJ2
TP1
TP2
Real Storage
Peak activity for TP2 and low for TP1
Control program
BJ 4
BJ 6
TP1
TP2
Real Storage
Peak activity for TP1 and medium activity for TP2
Control program
TP1
TP2
BJ7
BJ6

Figure 30.05.3. Conceptual illustration of real storage utilization in a mixed batch and online virtual storage environment

Summary

As the preceding discussion indicates, a virtual storage environment is designed primarily to provide new functional capabilities for the installation as a whole, although performance gains are possible for installations with particular environmental characteristics. The general functional aims of IBM-supplied virtual storage operating systems are (1) to use new hardware features and additional control program processing to support certain facilities that are not possible in a nonvirtual storage environment because of real storage restraints, and (2) to handle other functions that must be performed by installation personnel (programmers, operators, and system designers) when virtual storage and dynamic address translation are not used.

It is also important to note that while a virtual storage operating system permits an installation to be independent of real storage restraints to a large degree and enables real storage to be utilized more efficiently, the performance of the system and the specific advantages that can be achieved are still largely dependent on the amount of real storage present in the system and on the computing speed of the CPU, among other things. Hence, virtual storage and dynamic address translation are not a substitute for real storage. Rather, they provide an installation with greater flexibility in the tradeoff between real storage size and function or performance.

The degree to which a particular installation experiences the potential benefits of a virtual storage/dynamic address translation environment is system-configuration dependent and highly application dependent (number, type, complexity of applications installed). In addition, consideration must be given to the system resources that are specifically required to support a virtual storage environment (discussed in Section 30:15). Some of the potential advantages, such as those associated with application maintenance and operational flexibility and those that result from better management of real storage, can be experienced as soon as a virtual storage operating system is installed. Others may be achieved in the future when new applications are installed, and the less restrictive program design techniques available in a virtual storage environment are more fully utilized. In any case, installation of a virtual storage operating system can make System/370 easier to use and can be a major step toward more rapid installation of applications. Such an operating system can be of greatest benefit to installations desiring to move to or to extend online operations and thereby attain the advantages such an environment offers.

VIRTUAL STORAGE AND DYNAMIC ADDRESS TRANSLATION TERMINOLOGY

For the purpose of presenting the concepts of virtual storage and dynamic address translation, in the previous discussion, virtual storage, programs and data, direct access storage, and real storage were described as being divided into areas called sections. In reality, a unique term is used to describe each one of the various sections, namely, virtual storage page, page, slot, and page frame. In addition, virtual storage has two levels of subdivision in System/370. The following defines the new terminology actually used by the System/370 virtual storage operating systems.

Virtual storage in System/370 is divided into contiguous segments, which contain virtual storage pages. A virtual storage segment, as implemented in System/370, is a fixed-length, consecutive set of addresses for either 64K or 1024K bytes which begins on a 64K or 1024K boundary, respectively, in virtual storage. A virtual storage is divided into segments all of one size or the other. In general, in OS/VS1 and OS/VS2 environments, a segment is the unit of virtual storage allocation. Each segment of virtual storage is divided into contiguous, fixed-length, consecutive sets of addresses called virtual storage pages. Each segment in the virtual storage contains the same number of virtual storage pages, each of which is the same size. A virtual storage page, as implemented in System/370, can be either 2K or 4K bytes and is located on a 2K or 4K virtual storage boundary, respectively, within a segment.

The contents of virtual storage--instructions and data--are divided (by the operating system) into fixed-length contiguous areas called *pages*, corresponding in size to the virtual storage page size chosen, either 2K or 4K bytes. The addresses associated with a virtual storage page refer to the contents of a page.

The direct access storage used to contain the portion of the total contents of virtual storage that is not always present in real storage is called *external page storage*. Direct access space within external page storage is divided into physical records called *slots*, which are of page size, either 2K or 4K bytes. A slot, therefore, can contain one page at a time. A virtual storage page that is allocated and that actually has contents usually has a slot in external page storage associated with it to contain these contents (depending on the nature of the contents and how external page storage is managed by the operating system).

Instructions and data are transferred between external page storage and real storage as needed on a page basis. This transfer process is called *paging*, and a direct access device that contains external page storage is called a *paging device*. A slot in external page storage is associated with a particular virtual storage page by means of an algorithm or via tables that are maintained by the control program.

Real storage also is divided into fixed-length, consecutively addressed areas called *page frames*, which are always the same size as the virtual storage page being used, either 2K or 4K bytes. Page frames are located on 2K or 4K real storage boundaries. A page frame is a block of real storage that can contain one page. Hence, a page of data and/or instructions occupies a slot when it is in external page storage and a page frame when it is in real storage. Whether or not a page is present in real storage, a program addresses the contents of the page using virtual storage addresses.

The act of transferring a page from external page storage into real storage is called a *page-in*. This action may also be described as the *loading of a page*. The reverse act, transferral of a page contained in real storage to a slot in external page storage, is called a *page-out*. Figure 30.05.4 illustrates the relationship of virtual storage, external page storage, and real storage that was conceptually shown in Figure 30.05.2. (Note that the terms *swap-in*, *swap-out*, and *working set* have a specific meaning in a time-sharing environment and are defined in *OS/Virtual Storage 2 Features Supplement* under "Time Sharing Option".)

As previously indicated, DAT hardware uses tables to perform address translation. These tables are the *segment table* and *page tables*. One segment table and a set of page tables are required to perform address translation for one virtual storage. The segment table defines the virtual storage size, indicates allocated virtual storage, and points to the real storage location of the page tables. The page tables indicate which pages are currently in real storage and contain the real storage addresses of these pages. As pages are paged in and out, the control program makes changes to the page tables as required.

Basic to the implementation of virtual storage using direct access storage and DAT hardware is the method of determining when pages are to be brought into real storage and, therefore, when real storage is allocated to pages. The method supported by IBM-supplied virtual storage operating systems, that of bringing a page into real storage only when it is needed by an executing program, is called a *demand paging* technique. Since programs execute on a priority basis in OS/VS1 and OS/VS2 environments, as they do in OS (MFT and MVT) environments, real storage is, in effect, still allocated on a priority basis.

A request for a page-in is generated by the occurrence of a page exception or a page translation exception, a condition that is also called a page fault. An interruption occurs during the execution of an instruction when DAT hardware attempts to translate a virtual storage address into a real storage address and the appropriate page table indicates that the page is not currently present in real storage. A page fault condition causes an interruption in order to alert the control program to the fact that a page frame must be allocated. Usually, a page-in is required also to bring in the referenced instruction or data.

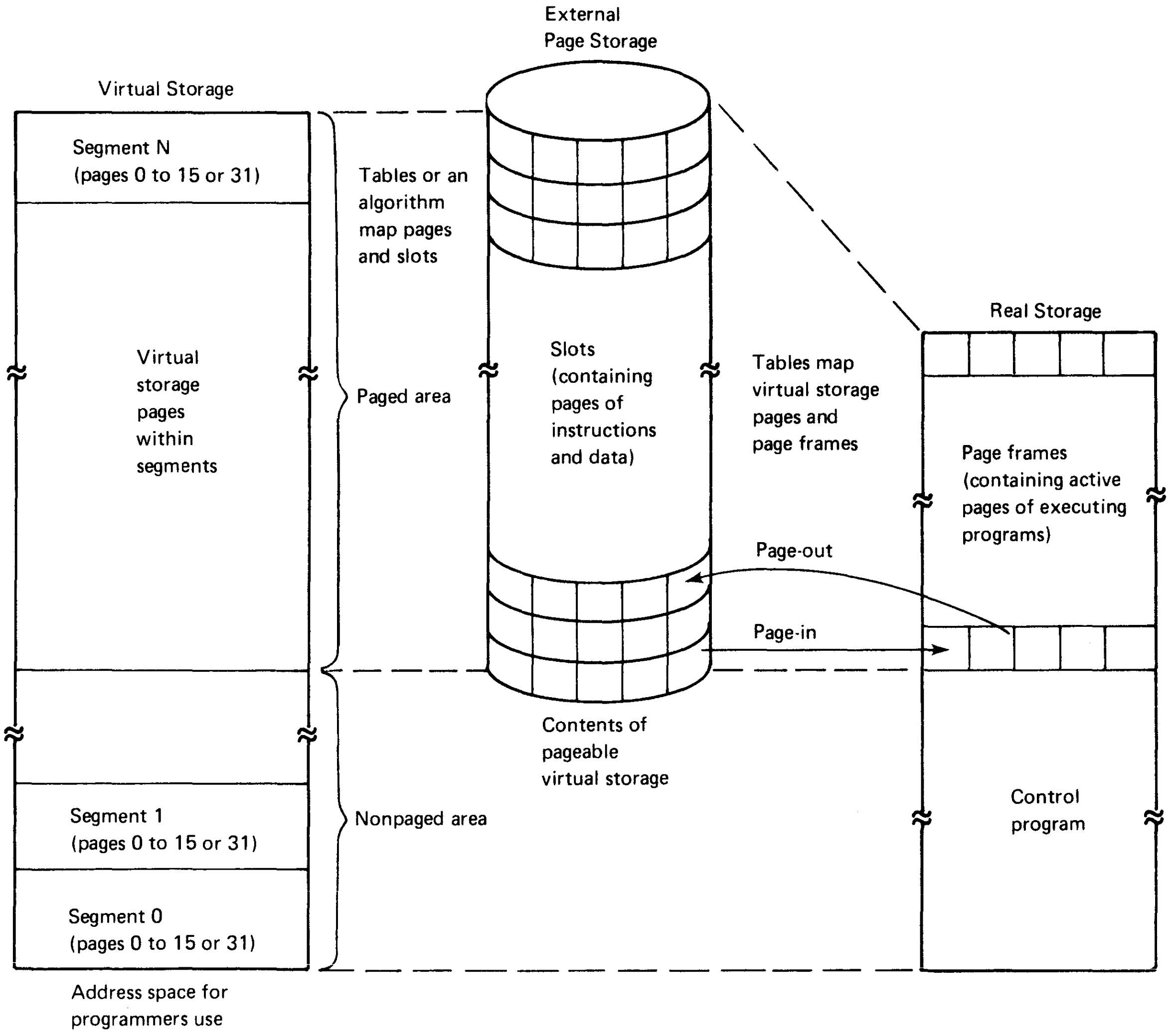

External
Page Storage
Virtual Storage
Segment N
(pages 0 to 15 or 31)
Tables or an algorithm map pages and slots
Real Storage
Virtual storage pages within segments
Paged area
Slots (containing pages of instructions and data)
Tables map virtual storage pages and page frames
Page frames (containing active pages of executing programs)
Page-out
Page-in
Contents of pageable virtual storage
Control program
Segment 1
(pages 0 to 15 or 31)
Nonpaged area
Segment 0
(pages 0 to 15 or 31)
Address space for programmers use

Figure 30.05.4. Layout of virtual storage, external page storage, and real storage

While page-ins are usually initiated as a result of a page fault, OS/VS1 and OS/VS2 provide an Assembler Language macro that can be used to cause one or more pages to be brought into real storage before they are referenced. Such requests are sometimes referred to as page-ahead requests. A page-ahead is required if, for reasons of proper system operation, a routine must operate without incurring any page faults.

Use of this macro is restricted because unlimited use of this facility can defeat the objective of demand paging.

When a page fault occurs and the control program determines that a page frame is not currently available for allocation, a choice must be made as to which allocated page frame will be taken away from the page to which it is currently assigned. The rule governing this choice is called the page replacement algorithm. If the page replacement algorithm is designed to choose from among only those page frames currently allocated to the program that caused the page fault, it is said to operate locally. If a page frame can be chosen from among all those available for allocation to all executing programs, the algorithm is said to operate globally. OS/VS1 and OS/VS2 implement a global page replacement algorithm. VM/370 supports a global page replacement algorithm and supports a local page replacement algorithm as an option. The algorithms used attempt to keep the most active pages of executing programs present in real storage. Hardware is included in System/370 models with dynamic address translation that indicates whether or not a page has been referenced or changed. Hence, when a page frame is required, a page determined by the algorithm to be relatively inactive is chosen for replacement.

Prior to loading a new page into the page frame chosen, the existing contents of the page frame must be saved if they were modified during processing. If modification occurred, a page-out operation is required; otherwise, an exact copy of the page already exists in external page storage. Code that is not modified during its execution, therefore, has an additional advantage in a virtual storage environment in that it need never be paged out once it has been written in external page storage. A program requiring a page-in is placed in the wait state until the page it requires has been loaded, during which time CPU control is given to another ready task, if one is available.

For various reasons, it is necessary to prevent a page-out of certain pages that are in real storage. One reason is for better operation of the system. This reason applies to certain frequently used control program routines, some routines that operate with the CPU in a disabled state (masked for I/O and external interruptions), most system tables, and most system control blocks. Integrity of system operation is another reason. Pages associated with certain types of operations must not be paged out while the operation is in progress, in order for the operation to proceed correctly. For example, pages that contain I/O buffer areas must remain in real storage while the buffers are being referenced during an I/O operation, after which a page-out can take place, if necessary. Another reason is the existence of time dependency. A page should not be written out if the program to which the page belongs must complete a logical operation that requires the page in less time than it takes to perform a page-in. Programs that handle I/O device testing operations, such as online tests (OLT's), can have such a time dependency.

A page that is identified as one that cannot be paged out (or, that is nonpageable) is called a fixed page. IBM-supplied operating systems support both long-term fixing and short-term fixing. Pages that should never be paged out when they are present in real storage are marked long-term fixed. The resident portion of an operating system control program is never paged and, therefore, its pages are marked long-term fixed. Pages that must be fixed for only a portion of the time they are present in real storage are marked short-term fixed. For example, a page containing an I/O buffer is marked short-term fixed prior to the initiation of the I/O operation that references the buffer. After the I/O operation completes, the page is unfixed and it becomes eligible for a page-out. Pages should be marked fixed only when necessary since page fixing reduces the amount of real storage that can be shared by concurrently executing paged programs (that which is available to be allocated to the nonfixed pages) and can, therefore, affect system performance.

As indicated previously, in OS/VS1 and OS/VS2 environments, a portion of the control program is resident in real storage. Its pages are marked fixed. This portion of the control program is not placed in external page storage (because it is not paged) even though it is allocated space in virtual storage. Certain other portions of an OS/VS1 and an OS/VS2 control program are pageable and are made resident in virtual storage, which means they are contained in external page storage during system operation. During system initialization, these pageable control program routines are allocated virtual storage and loaded into real storage from system libraries by the program fetch routine. These routines will be written in external page storage as a result of normal paging activity in OS/VS1 and as a result of specific page-out requests in OS/VS2. Control program routines that are resident in virtual storage are brought into real storage from external page storage, instead of from a system library, when they are required during system operation.

Just as control program routines can be fixed or pageable, problem programs operate in one of two modes in OS/VS1 and OS/VS2 environments: paged mode or nonpaged mode. The latter is also sometimes called virtual equals real (V=R) mode. When a problem program operates in paged mode, it is resident in virtual storage and pageable. A pageable program operates in a contiguous area of virtual storage (partition or region) and is assigned available real storage on a demand paged basis. Hence, virtual storage addresses must be translated into real storage addresses. The real storage dynamically allocated to programs operating in paged mode need not be contiguous and such programs normally can operate with less real storage than their design point (virtual storage) amount dynamically allocated to them. This is the mode of operation described in Section 30:05.

Paged mode is the normal mode of operation of programs in a virtual storage environment. However, certain programs cannot operate correctly in this mode, and must run in nonpaged (V=R) mode. In general, a program must operate in nonpaged mode if it:

- Contains a channel program that is modified while the channel program is active (Section 30:10 discusses the reason)

- Is highly time dependent (involves certain testing operations on I/O devices, for example)

- Must have all of its pages in real storage when it is executing (for performance reasons, for example)

Other characteristics that require a program to be executed in nonpaged mode and that are operating system dependent are listed in the programming systems supplements, which also discuss steps that can be taken to avoid running a program in nonpaged mode.

In OS/VS1 and OS/VS2 environments, a program that operates in nonpaged mode is dynamically allocated a contiguous virtual storage area and a contiguous real storage area of the same size with addresses identical to those of the allocated virtual storage area. (That is, virtual and real storage addresses of the allocated area are equal.) Since programs operating in V=R mode are not paged, they do not occupy external page storage. The entire program (except for dynamically requested modules) is loaded into real storage when it is initiated, and all its pages are fixed. The amount of real storage allocated to a program that runs in nonpaged mode must be a multiple of the page size used.

30:10 DYNAMIC ADDRESS TRANSLATION HARDWARE FOR THE MODEL 168

Dynamic address translation is a standard facility of the Model 168. It is made operative by turning on the translation mode bit in the current PSW. The system must also be operating in EC mode. When DAT is operative, storage addresses in programs referring to instructions and data are translated into real storage addresses after instructions are fetched during program execution. The address in the instruction counter is translated also. When DAT is not in operation, storage addresses in programs are used as real storage addresses. The storage addresses in CCW lists are not translated by channel hardware during channel program operation. The channel indirect data addressing feature, required on all installed channels for a Model 168 when a virtual storage operating system is used, and programmed channel program translation are discussed later in this subsection under "Channel Indirect Data Addressing".

The following instructions are associated with dynamic address translation: LOAD REAL ADDRESS, RESET REFERENCE BIT, and PURGE TLB. These instructions are valid in BC mode as well as in EC mode. They operate identically regardless of which mode is in effect. All are privileged instructions.

VIRTUAL STORAGE ORGANIZATION

The Model 168 (as well as other System/370 models with DAT hardware) supports a virtual storage segment size of either 64K or 1024K bytes, as determined by bits 11 and 12 of control register 0. With either segment size, the page size can be 2K or 4K, as determined by bits 8 and 9 of control register 0. A segment size of 1024K bytes is not supported by DOS/VS, OS/VS1, OS/VS2, or VM/370. Table 30.10.1 summarizes the virtual storage organization provided in System/370.

Table 30.10.1. Number and size of segments and pages for a 16-million-byte virtual storage

CR 0 Bits 11,12	CR 0 Bits 8,9	Segment Size in Bytes	Number of Segments in the Virtual Storage	Page Size in Bytes	Number of Pages in a Segment
10	01	1,048,576	16	2048	512
10	10	1,048,576	16	4096	256
00	01	65,536	256	2048	32
00	10	65,536	256	4096	16

As already described, the addresses supplied in programs directly address a location in the virtual storage that is supported by the virtual storage operating system. In this sense, program-supplied addresses can be viewed as virtual storage addresses that specify a byte within a particular virtual storage page and segment. The logic of the translation process is described in this subsection in these terms. The architectural definition of dynamic address translation found in System/370 Principles of Operation (GA22-7000-2 and later editions) assumes that the addresses in programs consist of three fields, two of which are used to index tables during the translation process. Under these conditions, the addresses supplied by a program are considered to be logical addresses instead of virtual storage addresses.

For the purpose of translation, a virtual storage address is divided into three fields: (1) a segment field, which identifies a segment within the virtual storage, (2) a page field, which identifies a page within the segment addressed, and (3) a byte displacement field, which identifies a byte within the page addressed. The number of bits in each field varies depending on the segment and page sizes used. Virtual storage address fields for a segment size of 64K and a specific example of how the fields are used to address a location in virtual storage are shown in Figure 30.10.1.

OPERATION OF DYNAMIC ADDRESS TRANSLATION HARDWARE

Address Translation Tables

One segment table is required to describe one virtual storage. If more than one virtual storage is supported by a single computing system, there is a segment table for each virtual storage implemented. A segment table contains one four-byte entry for each segment in the virtual storage the table describes, up to a maximum of 256 entries for the maximum size virtual storage of 16 million bytes (using 64K segments). The real storage address of the segment table (or of the currently active segment table if multiple virtual storages are implemented) is contained in control register 1. The current length of the segment table is also indicated in control register 1. The length value is used by the hardware during translation to ensure that the segment entry being referenced falls within the segment table.

The segment table entries point to the real storage locations of the page tables. There is one page table for each segment in the virtual storage defined (or, in OS/VS2, currently allocated), up to a maximum of 256 page tables for a 16-million-byte virtual storage with 64K segments. A segment table entry contains an indication of the length of the page table, the high-order 21 bits of the real storage address of the page table, and an indication of whether or not the entry itself is valid and can be used for translation purposes (invalid bit). If the invalid bit is on in a segment table entry, a segment translation exception occurs during the translation process.

A page table has one entry for each page in the particular segment the page table describes. For a 64K segment, there are 32 or 16 entries in a page table depending on whether a 2K or a 4K page is used, respectively. A page table entry is two bytes in size. It contains the 12 (for a 4K page) or 13 (for a 2K page) high-order bits of the real storage address of the page frame that is currently allocated to the virtual storage page that the page table entry describes. Each page table entry also contains an invalid bit to indicate whether the entry can be used for translation. The invalid bit is on when a virtual storage page does not have real storage currently allocated to it. A page translation exception occurs during the translation procedure if this invalid bit is on.

Segment and page table formats and entries used for address translation are shown in Figure 30.10.2. In effect, the segment and page tables define the relationship between virtual and real storage at any given time. The segment table reflects the current size of virtual storage, which must be a multiple of the segment size (64K for IBM-supplied support), and the location of required page tables. The segment table also indicates, by means of its invalid bits, which segments of virtual storage are currently allocated and have a page table available. The page tables indicate, via their invalid bits, which virtual storage pages currently have a page frame allocated and the location (real storage address) of these page frames.

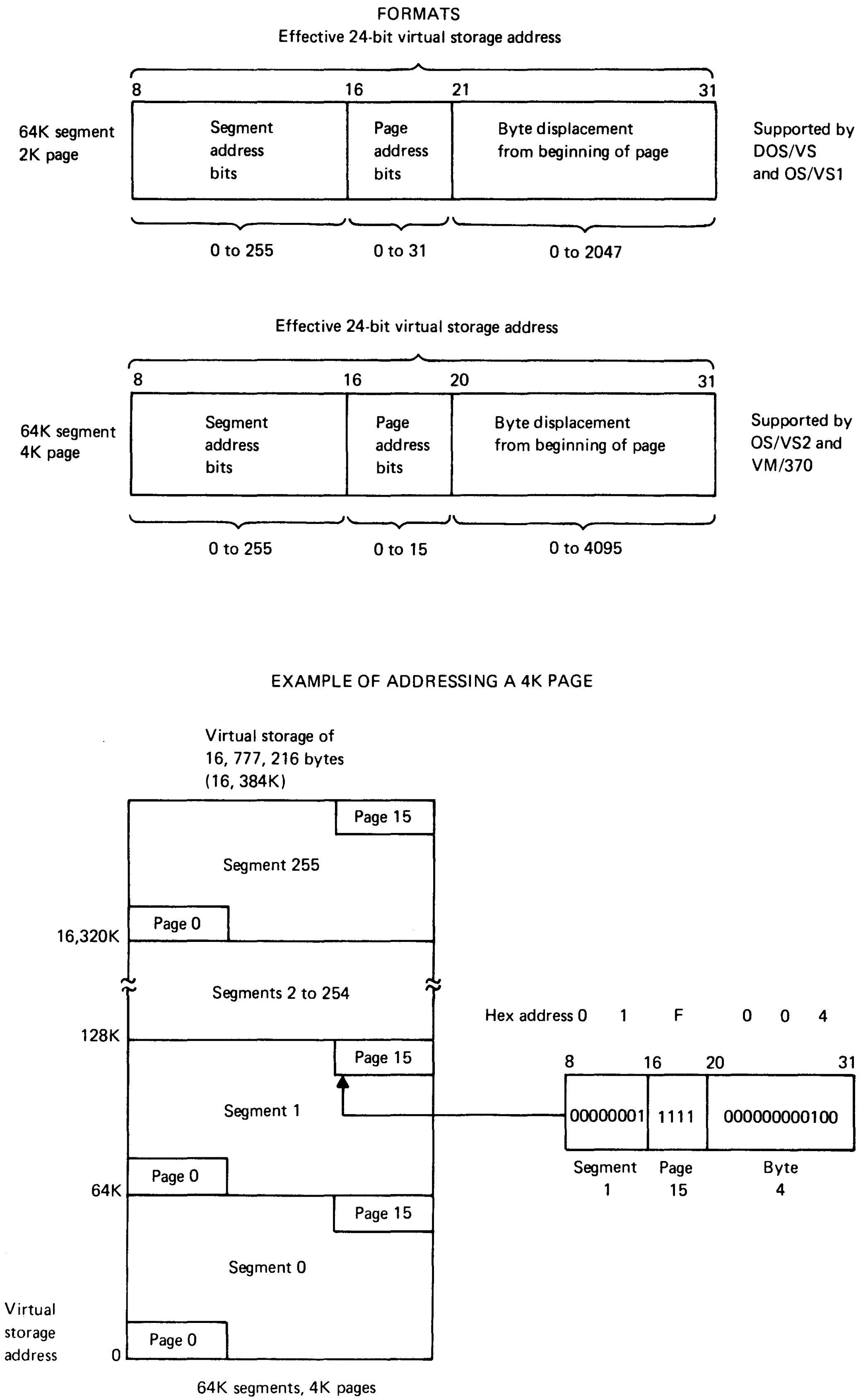


Figure 30.10.1. Virtual storage address fields for a 64K segment

In an OS/VS1 environment, segment and page tables are established at system initialization. Page tables are modified during system operation by control program routines to reflect the current allocation of real storage to virtual storage so that address translation can take place. In an OS/VS2 environment, in which virtual storage as well as real storage is dynamically allocated and deallocated, the segment table constructed during IPL is modified as required during system operation to reflect the allocation of virtual storage, and page tables are created and destroyed as necessary.

Address Translation Process

A translation request is either explicit or implicit. Explicit translation is invoked via execution of the LOAD REAL ADDRESS instruction. Implicit translation is invoked to translate all instruction addresses and data addresses contained in other instructions. Implicit address translation takes place during instruction execution.

The logical flow and the details of the translation process are given in Figure 30.10.3. The procedure consists of a two-level, direct address table lookup operation. Any type of translation exception causes a program interruption and termination of the hardware translation process. The CPU cannot be disabled for translation exception interruptions. Segment and page translation exceptions that occur during an explicit translation request (LOAD REAL ADDRESS instruction) are indicated via the condition code setting instead of via an interruption.

Translation Lookaside Buffer

In the Model 168, a translation lookaside buffer (TLB) is implemented to reduce the amount of time required to perform address translation. The translation lookaside buffer is used to retain up to 128 previously translated addresses. Addresses associated with up to six different virtual storages can be contained in the TLB at any time. Every time a virtual storage address is translated during instruction execution, the virtual storage address, the resulting real storage address and its associated storage protect key, and identification of the virtual storage to which the virtual storage address belongs are placed in one of the 128 TLB locations. A hashing algorithm is applied to the virtual storage address in order to determine which of the 128 TLB locations is to be used.

After the effective virtual storage address has been computed and prior to performing the translation using segment and page tables, the TLB is interrogated to determine whether or not it contains the required translated address. Interrogation of the TLB is done in parallel with reference to the index array for the buffer. Therefore, no translation cycles are required when the translated address is obtained from the TLB. If the TLB does not contain the required translation or if the entry is invalid, as indicated by a zero identification code, the complete table-lookup translation procedure, as previously described, is performed. In the Model 168, the number of CPU (80 nanosecond) cycles required for address translation when the translation is not obtained from the TLB varies from a minimum of 8 to a maximum of 26, assuming no I/O interference, depending on the locations of the segment table and the page table entries required for the translation. In the Model 165 II, from 8 to 46 CPU cycles are required for the translation process when the required translation is not contained in the TLB.

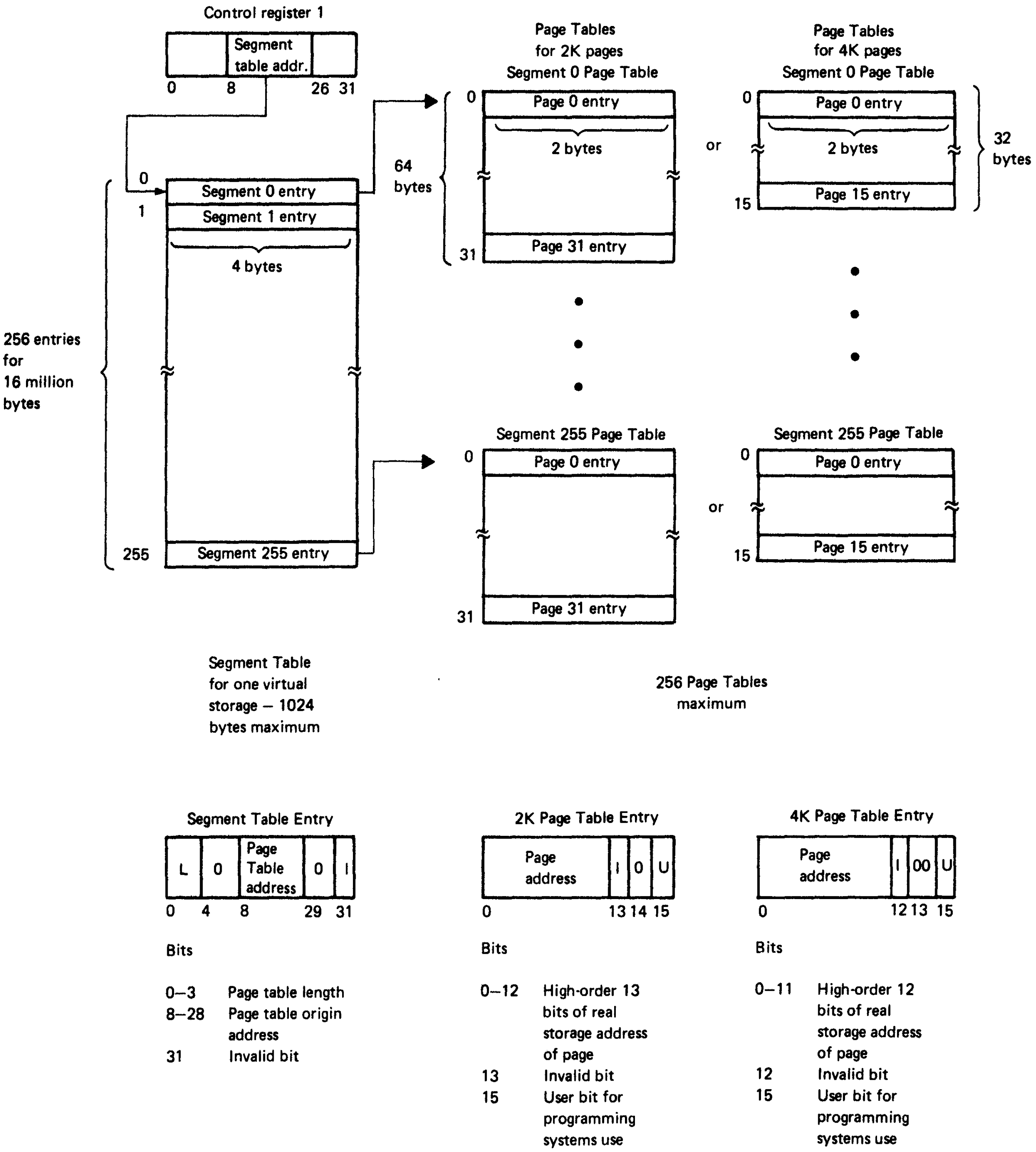
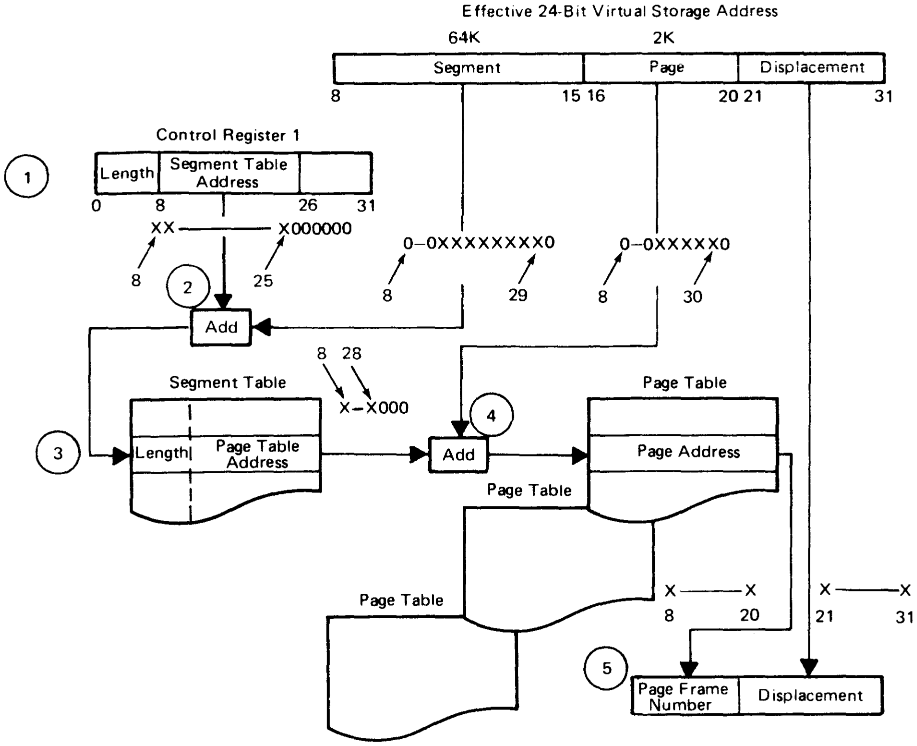


Figure 30.10.2. Segment table and page tables used for dynamic address translation



1. Bits 8, 9, 11, and 12 in control register 0 are checked for validity. A translation specification interruption occurs if an invalid setting is present. Segment address bits from the virtual storage address are checked using length bits in control register 1. If the segment entry address is outside the segment table, a segment translation exception is indicated.

2. Six low-order zeros are appended to the segment table address in control register 1. Two low-order zeros are appended to the segment bits from the virtual storage address. The two values are added to obtain a segment table entry. If the invalid bit is on in this entry, a segment translation exception is indicated.

3. Page address bits from the virtual storage address are checked using page table length bits contained in the segment table entry. A page translation exception is indicated if the entry addressed is outside the page table.

4. Three low-order zeros are appended to the page table address contained in the segment entry. One low-order zero is appended to the page address from the virtual storage address. The two values are added to obtain a page table entry. If the invalid bit is on in this entry, a page translation exception is indicated.

5. The 24-bit real storage address is formed using the 12 or 13 high-order bits from the page table entry and the 12 or 11 low-order bits from the virtual storage address.

Figure 30.10.3. Dynamic address translation procedure

All the entries in the TLB are invalidated (identification codes set to zero) when a reset occurs, the operator enters a storage configuration via the configuration panel, or retry recovery is attempted after a machine check occurs. When a SET STORAGE KEY is issued and valid translated addresses are in the TLB, the TLB is searched and each entry is invalidated that has the same real address as the one for which the key is being set. The PURGE TLB instruction is provided to enable a program to invalidate all 128 TLB entries. In general, this instruction must be issued when an entry in a page table is invalidated, since the real storage address being invalidated could be contained in the TLB. The TLB will be purged by the virtual storage operating systems as required.

A change in segment table origin address, segment size, or page size can also affect the validity of current TLB entries. In order to reduce the number of full TLB purges required by such changes, a segment table origin address register stack (STO-stack) is implemented. The STO-stack can contain the address of six different segment tables at a time. Each segment table could define a different virtual storage. A STO-stack entry also indicates the segment and page size in effect for the virtual storage associated with the segment table address.

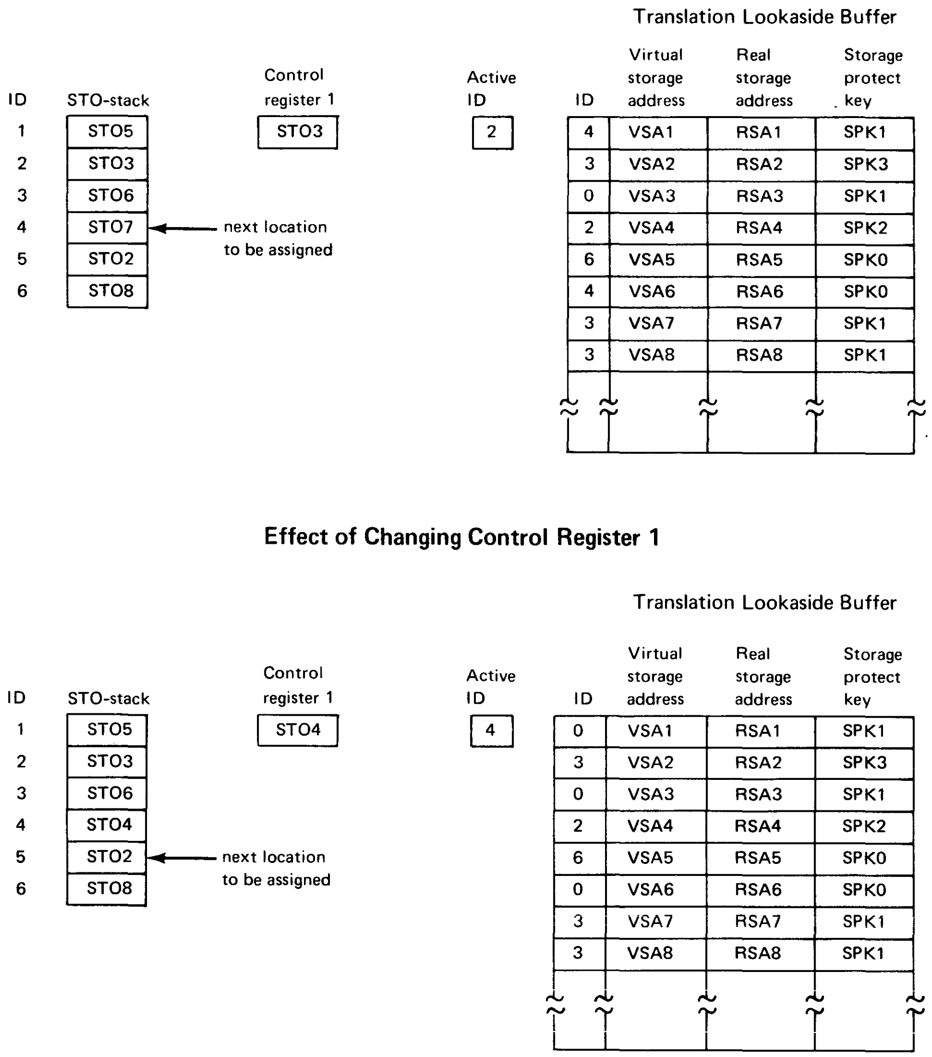
The six entries in the STO-stack have a unique identification number associated with them. One of these numbers is denoted to be the currently active identification number. Whenever a segment table address is placed in control register 1, the segment table address is also placed in the STO-stack, if it is not already there, and the identification number the segment table address is assigned becomes the new active identification number.

A STO-stack identification number is stored with each TLB entry to identify the segment table, and thereby the virtual storage, with which the TLB entry is associated. When the TLB is interrogated to see if it contains the required translation, the STO-stack identification number of the TLB entry is compared to the active identification number. If the identifications are equal, this indicates the TLB location contains a translation from the virtual storage associated with the active identification number. If the identifications are not equal, the TLB location contains a translation for a different virtual storage and, therefore, the TLB entry does not contain the required translation even though it may contain a virtual storage address equal to the one that is to be translated.

When DAT mode is entered or a LOAD CONTROL instruction is issued when DAT mode is operative, the segment table address in control register 1 and page and segment size specifications from control register 0 are compared with each of the STO-stack locations to determine whether a change in these specifications is being made. If a change is indicated, some TLB purging may be required.

If an equal comparison is found between an STO-stack entry and the segment table address, segment size, and page size in control registers 0 and 1, this indicates that the virtual storage associated with the segment table address now in control register 1 is currently one of the six virtual storages whose translations are being maintained in the TLB and that segment and page size have not been changed. The STO-stack identification number of the segment table address now in control register 1 is designated to be the active identification. No TLB purging is required.

If no equal comparison is found between an STO-stack entry and the segment table address, segment size, and page size in control registers 0 and 1, this indicates translations for the segment table now indicated by control register 1 are not currently being maintained in the TLB or that segment or page size is being changed. The new segment table address is placed in the STO-stack, and the STO-stack identification number assigned becomes the active identification. A first-in first-out algorithm is used to determine which STO-stack location to assign. If the new address displaces another segment table address, the TLB entries associated with the displaced segment table (and virtual storage) must be purged. This is done by setting the identification number to zero for each entry in the TLB that has the same STO-stack identification number as the segment table address that was displaced. This identification number is now assigned to the newly stored segment table address. The other TLB entries need not be invalidated. See Figure 30.10.4 for an example of TLB purging when control register 1 is changed.



Translation Lookaside Buffer

ID	STO-stack
1	STO5
2	STO3
3	STO6
4	STO7
5	STO2
6	STO8

ID	Virtual storage address	Real storage address	Storage protect key
4	VSA1	RSA1	SPK1
3	VSA2	RSA2	SPK3
0	VSA3	RSA3	SPK1
2	VSA4	RSA4	SPK2
6	VSA5	RSA5	SPK0
4	VSA6	RSA6	SPK0
3	VSA7	RSA7	SPK1
3	VSA8	RSA8	SPK1

Effect of Changing Control Register 1

Translation Lookaside Buffer

ID	STO-stack
1	STO5
2	STO3
3	STO6
4	STO4
5	STO2
6	STO8

ID	Virtual storage address	Real storage address	Storage protect key
0	VSA1	RSA1	SPK1
3	VSA2	RSA2	SPK3
0	VSA3	RSA3	SPK1
2	VSA4	RSA4	SPK2
6	VSA5	RSA5	SPK0
0	VSA6	RSA6	SPK0
3	VSA7	RSA7	SPK1
3	VSA8	RSA8	SPK1

Figure 30.10.4. TLB purging when control register 1 is changed

Implementation of the STO-stack in the Model 168 enables a control program that supports multiple virtual storages (such as VM/370) to alter control registers 0 and 1 in order to change the virtual storage for which address translation is effective, without automatically causing purging of the entire TLB. The STO-stack facility will also be of benefit in an OS/VS2 environment since OS/VS2 supports two segment tables in order to provide fetch protection for all regions (see *OS/Virtual Storage 2 Features Supplement*).

Addresses Translated

All storage addresses that are explicitly designated by a program and that are used by the CPU to refer to instructions or data in processor storage are virtual storage addresses and are subject to address translation. Thus, when DAT is operative, the starting and ending storage addresses used with the program event recording feature are virtual, as are the storage addresses stored in PSW's during interruptions. Address translation is not applied to addresses that explicitly designate protect key storage locations or to quantities that are formed as storage addresses from the values designated in the base and displacement fields of an instruction but that are not used to address processor storage (shift instructions, for example). In addition, address translation is not applied to the storage addresses in CCW lists used for I/O operations.

Some of the storage addresses supplied to a program by the CPU are virtual and some are real. Table 30.10.2 lists, for the Model 168, those storage addresses designated by a program, either explicitly or implicitly, that are virtual (and, therefore, are subject to translation) and those addresses that are real or not used to reference processor storage and, thus, are not translated. The table also indicates which storage addresses supplied to a program are virtual and which are real.

FEATURES TO SUPPORT DEMAND PAGING

Reference and Change Recording Facility for Real Storage Blocks

A hardware recording facility is standard in the Model 168. This facility provides continuous recording of the activity of all 2K real storage blocks via reference and change bits. The settings of these recording bits can be used by control program routines to support a demand paging environment. This hardware facility is always active; it does not depend on EC or translation mode being operative.

The seven-bit key associated with each 2K real storage block in the Model 168 has four storage-protect bits, one fetch-protect bit, one reference bit, and one change bit. During system operation, the activity of each 2K real storage block is monitored by hardware. Whenever a fetch is made either by a CPU or a channel to a real storage address, the reference bit in the key associated with the 2K storage block that contains that real storage address is turned on by the hardware. A store into any real storage address causes the hardware to turn on both the change bit and the reference bit for the affected 2K block.

Store/display operations initiated from the 3066 console also cause appropriate changing of the reference and change bits. The RESET REFERENCE BIT instruction is provided to allow the reference bit of any 2K real storage block to be reset by programming without altering the contents of the other six bits in the protect key. A CPU fetch that is satisfied with data contained in the buffer does not cause reference recording in the Model 168. There are situations, however, in which instruction or operand prefetching may cause the reference bit for a page frame to be turned on even though the contents of that page are never used.

Table 30.10.2. Virtual and real storage addresses used by and supplied to programs in the Model 168

<u>Virtual Storage Addresses Explicitly Designated by the Program</u> (translated)

- Instruction address in the PSW
- Branch addresses in instructions
- Addresses of operands in instructions
- Operand address in the LOAD REAL ADDRESS instruction
- PER starting address in control register 10 and PER ending address in control register 11

<u>Real Storage Addresses Explicitly Designated by the Program</u> (not translated)

- Operand addresses in SET STORAGE KEY, INSERT STORAGE KEY, and RESET REFERENCE BIT instructions
- Machine check extended log pointer in control register 15
- I/O extended log pointer in location 172
- Segment-table-origin address in control register 1
- Page-table-origin address in a segment table entry
- Page frame address in a page table entry
- CCW address in the channel address word (CAW)
- Address in a CCW specifying a data area or the location of another CCW
- Data address in channel indirect data address lists

<u>Addresses Not Used to Address Storage</u> (not translated)

- Operand addresses specifying the amount of shift in fixed-point, logical, or decimal shift instructions
- Operand address in LOAD ADDRESS and MONITOR CALL instructions
- I/O addresses in I/O instructions and in the Input/Output Communication Area (IOCA)

<u>Real Storage Addresses Used Implicitly</u> (not translated)

- Addresses of PSW's used during an interruption and in executing the programmed or manually initiated restart function
- Address used by the CPU to update the timer at location 80
- Address of the CAW, the CSW, and the I/O address within the IOCA used during an I/O interruption or during execution of an I/O instruction, including execution of STORE CHANNEL ID
- Addresses used for the store status function

<u>Virtual Storage Addresses Provided to the Program</u>

- Address stored in the instruction address field of the old PSW during an interruption
- Address stored by a BRANCH AND LINK instruction
- Address stored in register 1 by TRANSLATE AND TEST and EDIT AND MARK instructions
- Address stored in location 144 on a program interruption for a page translation or segment translation exception
- Address stored in location 152 on a PER interruption

<u>Real Storage Addresses Provided to the Program</u>

- The translated address generated by the LOAD REAL ADDRESS instruction
- Address of the segment table entry or page table entry provided by the LOAD REAL ADDRESS instruction
- Failing storage address in location 248
- CCW address in the CSW

The hardware reference and change recording facility is used by the page replacement algorithm of a virtual storage operating system. When a page is loaded into a page frame, the reference and change bits for that page frame are set to zero. (When a 4K page size is used, the reference and change bits for both of the 2K storage blocks involved are reset.) Thereafter, the reference bit is used to determine the activity of a page. The change bit is inspected to determine whether a page must be paged out when its page frame is reassigned. The SET STORAGE KEY instruction must be used to reset the change bit.

Instruction Nullification

When a page fault occurs in a demand paging environment, execution of the instruction that caused the page fault stops and the control program gains control to initiate a page-in operation. When the contents of the missing page have been loaded (and the appropriate page table entry has been updated), the instruction that caused the page fault is reissued. In order for the instruction to operate correctly the second time, execution of the instruction must have been stopped such that reexecution gives the same results as would occur if the instruction had been executed only once. Therefore, the contents of real storage, the general and floating-point registers, and the PSW must not be altered.

The execution of an instruction is said to be nullified when it is stopped such that no operation was performed, no fields were changed, and the PSW indicates the address of the instruction that was stopped. Interruptible instructions, such as MOVE LONG, are divided into execution units. One or more execution units may have completed before a page fault is detected. In this case, only the current execution unit is nullified.

Various methods are used, depending on the type of instruction, to determine the need for nullification. In some cases, execution is attempted where hardware detection of page faults permits nullification. In other cases, pretesting is required to determine whether the virtual storage pages to be referenced have page frames allocated. Nullification testing is required only for instructions whose translated addresses reference storage. In the Model 168, testing is performed by instruction unit hardware and/or additional microcode routines that are executed prior to normal instruction execution. However, for some instructions, prefetching of the data accomplishes pretesting so that no additional pretesting cycles are required. A LOAD instruction that addresses a word on a fullword boundary is an example of such an instruction.

Similarly, if a store fullword instruction addresses a four-byte field that is not on a fullword boundary, a pretest is required to determine whether all four bytes are contained in real storage. The pretest microcode for this instruction issues a fetch to the highest addressed byte in the four-byte data field (virtual storage address in the instruction plus three). The absence of a page translation exception during translation of the virtual storage address indicates that (1) if the data field spans two pages, at least the second of the two pages is present in real storage or (2) the data field is totally contained in one page, which is present in real storage. Hence the instruction is allowed to proceed without nullification. If the data field actually does span two pages and the first page is not present in real storage, this fact will be indicated by a page fault during translation of the address of the high-order byte of the field. Instruction nullification will occur and the page fault will cause a page-in of the first page to be initiated by the control program as usual.

If the pretest fetch operation does cause a translation exception, the store fullword instruction is nullified and the control program gains CPU control to load the missing page. Once again, the page-in caused by the pretest may have brought in the second of two pages spanned by the data field or the only page containing the data field. After the page-in, the instruction is reexecuted.

CHANNEL INDIRECT DATA ADDRESSING

Since address translation is not performed by the channels for programs that operate in paged mode, address translation must be performed on CCW lists by programming prior to the initiation of START I/O instructions. Such address translation need not be performed on the CCW lists in programs that operate in nonpaged mode.

In addition, a contiguously addressed I/O area in virtual storage can span a set of noncontiguous page frames. Hence, a method of handling a noncontiguously addressed I/O area in real storage during the operation of a CCW list is required. The channel indirect data addressing feature is used to provide this capability. As is shown in Figure 30.10.5, the use of channel indirect data addressing allows the channel program logic used in the CCW list with virtual storage addresses to be maintained in the new CCW list that contains real storage addresses.

When channel indirect data addressing is present, bit 37 of a CCW is designated as the indirect data address (IDA) flag. The IDA flag applies to read, read backward, write, control, and sense commands and is valid in both BC and EC modes. When the IDA flag in a CCW is zero, bits 8 to 31 of the CCW specify the real storage address of the beginning of the I/O area as usual. When the I/O area referenced by a CCW is completely contained in one page, an indirect data address list (IDAL) is not required and the IDA flag is set to zero. When the IDA flag is one, CCW bits 8 to 31 specify the real storage address of an IDAL instead of an I/O area. When the I/O area referenced by a CCW spans two or more pages, an IDAL is required and the IDA flag is set to one.

An IDAL consists of two or more contiguous indirect data address words (IDAW's) of four bytes each. There is one IDAW in an IDAL for each 2K storage block spanned by the I/O area. An IDAW, which must be aligned on a fullword boundary, contains a real storage I/O area address in bits 8 to 31. Bits 0 to 7 must be zero. The first IDAW in the list points to the beginning of the I/O area to be used by the CCW and is obtained by translating the virtual storage address contained in the original CCW. Any valid real storage address can be specified in the first IDAW of a list. All IDAW's after the first must address the beginning (or end for a read backward operation) of a 2048-byte block located on a 2048-byte boundary, or a program check occurs. That is, bits 21-31 of the address in the IDAW must be zeros (or ones for a read backward).

Figure 30.10.5 shows an example of the IDAL's required for a command-chained CCW list when 2K pages are used. The IBM-supplied virtual storage operating systems construct a new CCW list with translated addresses that is used to control the I/O operation. The new CCW list points to any required IDAL's.

When a START I/O instruction is executed, the channel fetches the first CCW in the list, pointed to by the channel address word (CAW), and inspects bit 37. If it is zero, the operation is started in the I/O area specified by the real storage address in the CCW. If bit 37 is a one, the first IDAW is fetched from the real storage address in the CCW. The I/O operation is begun using the real storage address in the first IDAW and, assuming that the I/O operation is not a read backward, ascending real storage addresses in the I/O area are used by the channel until a 2048-byte boundary is reached.

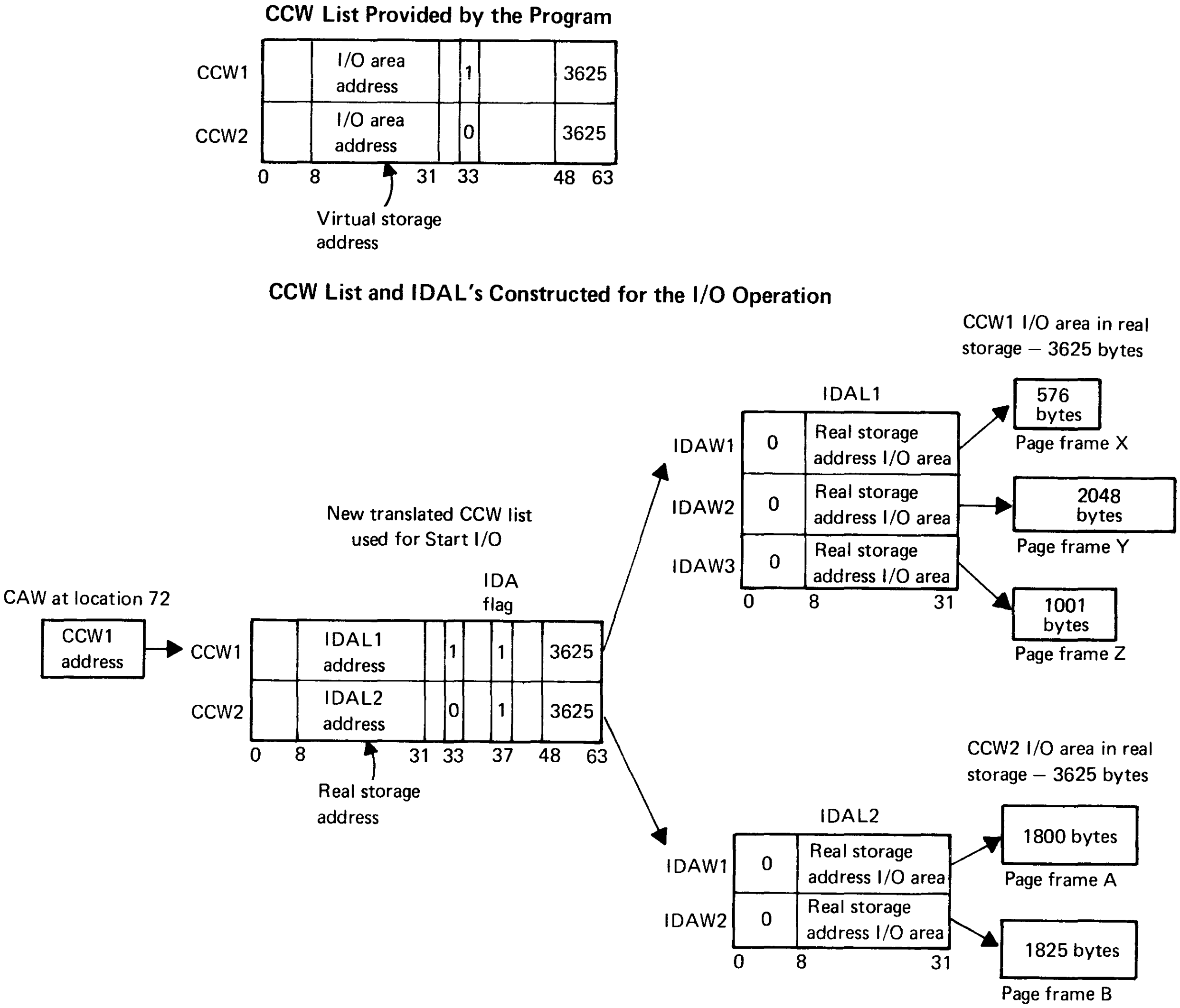

CCW List Provided by the Program
CCW1
I/O area address
1
3625
CCW2
I/O area address
0
3625
0
8
31
33
48
63
Virtual storage address
CCW List and IDAL's Constructed for the I/O Operation
CAW at location 72
CCW1 address
New translated CCW list used for Start I/O
IDA flag
CCW1
IDAL1 address
1
1
3625
CCW2
IDAL2 address
0
1
3625
0
8
31
33
37
48
63
Real storage address
IDAL1
IDAW1
0
Real storage address I/O area
IDAW2
0
Real storage address I/O area
IDAW3
0
Real storage address I/O area
0
8
31
CCW1 I/O area in real storage – 3625 bytes
576 bytes
Page frame X
2048 bytes
Page frame Y
1001 bytes
Page frame Z
CCW2 I/O area in real storage – 3625 bytes
IDAL2
IDAW1
0
Real storage address I/O area
IDAW2
0
Real storage address I/O area
0
8
31
1800 bytes
Page frame A
1825 bytes
Page frame B

Figure 30.10.5. Example of IDAL's required for a CCW list

The channel detects a 2K boundary by monitoring I/O area address bits 21-31. When these bits change from all ones to all zeros, the first byte of the next 2K real storage block is indicated. At this point, the channel accesses the second IDAW in the list to obtain the next real storage I/O area address to be used, and the data transfer operation continues. The channel continues using the IDAL until the operation indicated by the CCW completes (CCW count reaches zero, interrecord gap on tape reached, etc.). The next CCW is accessed if command or data chaining is indicated. Bit 37 is inspected and the I/O operation continues as described until the CCW list is exhausted.

When a program operates in paged mode, the CCW list for an I/O operation must be inspected and the appropriate IDAL's must be constructed prior to issuing a START I/O instruction. At the completion of the I/O operation, some retranslation is also required. In general, the following steps must be taken for each CCW in a given list:

1. Determine whether the I/O area referred to in the CCW spans pages or is contained in only one. If a single page is involved, translate the virtual storage address to real and store it in the

CCW. Ensure that a page frame is allocated to the page containing the buffer and that the page frame is marked fixed.

2. If two or more pages are involved, set up the required number of IDAW's, place a pointer to the IDAL in the CCW, and turn on CCW bit 37.

3. While setting up IDAW's, determine whether all pages in the I/O area have real storage allocated. If not, ensure that page frames are allocated and fixed.

At the completion of the I/O operation, the real storage address in the channel status word must be translated to a virtual storage address, and the pages that were short-term fixed prior to initiation of the I/O operation must be unfixed. Channel program translation and page fixing are performed by the I/O control portion of the control program in IBM-supplied operating system support. A program that contains a CCW list that is dynamically modified during its execution cannot operate correctly in paged mode, since the modification is made to the CCW list with virtual storage addresses rather than to the translated CCW list that is actually controlling the I/O operation on the channel.

30:15 SYSTEM PERFORMANCE IN A VIRTUAL STORAGE ENVIRONMENT

A virtual storage environment is designed to provide new data processing capabilities. As is true about any other capability offered by an operating system, support of a new function requires control program use of a certain amount of the hardware resources of the system. In this respect, virtual storage is no different from multiprogramming and the many other new capabilities that have continuously been added to OS since its initial release.

The characteristic that makes virtual storage different from most other features is that virtual storage is not primarily designed to improve system performance, as are many other control program facilities. Virtual storage is first a functional tool and, in certain cases, can also be a performance tool. The objectives of OS virtual storage operating systems are to (1) provide new functions, (2) maintain upward compatibility with OS nonvirtual storage environments, and (3) provide performance equal to or better than that achieved with a nonvirtual storage operating system using the same system configuration. Attainment of the last objective will not be possible for all existing System/370 configurations.

In addition, some of the new functions a virtual storage environment provides cannot be achieved in a nonvirtual storage environment or are not practical, and in these cases, performance is not the primary consideration when using the facility virtual storage offers. As the cost of hardware resources continues to decline on a unit cost basis (cost per processor storage bit, cost per direct access bit, etc.), it becomes increasingly more economical to use system resources to perform functions that otherwise are handled by installation personnel.

The other new characteristic of virtual storage is that it enables a given system configuration to provide a wider range of performance, as well as function, as a result of the new factors that affect operation of a system with virtual storage support. Thus, a slightly different approach must be taken in planning for and in evaluating system performance in a virtual storage environment.

Many of the same factors that affect system performance in an OS/VS1 or OS/VS2 environment are the same as those that apply to OS MFT or OS MVT, respectively. First, the system configuration must include the hardware resources (CPU speed, channels, I/O devices, real storage) required for the control program and job mix. This subsection identifies the system resources specifically required to support a virtual storage environment. Second, the system should be designed to balance resource usage to achieve optimum throughput, and to use applicable performance and control program design options the particular operating system offers, taking into account the characteristics of the installation job stream.

The performance of a system in a virtual storage environment is also affected by certain new factors that do not apply to systems without virtual storage support. This subsection identifies these new factors, explains how they generally affect system performance, and indicates steps that can be taken to increase and maximize system performance when a virtual storage operating system is used.

This discussion applies to OS/VS1 and OS/VS2, and is restricted to performance factors that are common to the virtual storage environments they support. The virtual storage operating systems also offer new performance-oriented enhancements that are not related to the implementation of virtual storage. These unique performance features are discussed in the optional programming systems supplements.

The performance information in this subsection is designed to present concepts and considerations for a virtual storage environment. Figures and graphs are used for illustrative purposes. They do not represent any particular installation or measured results. Their purpose is to illustrate the interrelated factors of multiprogramming performance in a virtual storage environment. The performance information presented is conceptual. It is based on the experience and judgment of IBM individuals with performance knowledge and on performance measurements made during development of OS/VS1 and OS/VS2. Therefore, it may not apply to all installations.

SYSTEM RESOURCES REQUIRED TO SUPPORT A VIRTUAL STORAGE ENVIRONMENT

In order to support a demand paged virtual storage environment using System/370, in which programs are operating in paged mode, additional system resources are used by the IBM-supplied virtual storage operating systems, as follows:

- Dynamic address translation hardware requires CPU time to perform virtual storage to real storage address translation. The amount of time required is determined by the System/370 model and the number of times the full table-lookup translation procedure must be performed. The Model 168, for example, has a translation lookaside buffer that is designed to reduce use of the full table-lookup translation procedure. The CPU time required is also affected by program structure (which is discussed later). A small amount of additional CPU time is also required to pretest certain instructions that reference storage, as discussed under "Instruction Nullification" in Section 30:10. Studies have shown that a relatively small percentage of the total CPU time specifically required to support a virtual storage environment is devoted to address translation by DAT hardware.

- CPU time is required to translate the virtual storage addresses in channel programs (CCW lists) into real storage addresses, build indirect data address lists (where necessary), and short-term fix pages that will be referenced during I/O initiation, execution, and interruption handling. Channel program translation and page fixing are performed prior to the initiation of each I/O operation with a channel program that contains virtual storage addresses. Channel status word retranslation and page unfixing is performed at the completion of these I/O operations. The amount of CPU time this function requires per data set is affected by the number of I/O requests (EXCP macros) issued, the number of CCW's in the channel programs started, the number of pages that must be fixed, and whether or not indirect data address lists have to be constructed. Studies have shown that a large portion of the total CPU time specifically required to support a virtual storage environment is used to perform channel program translation and page fixing.

- CPU time is required to process page faults and for the execution of other control program code that is specifically required to support a virtual storage environment. CPU time is required for such things as servicing additional program interruptions, managing and allocating real and external page storage, maintaining tables used by DAT hardware, and testing for paged or nonpaged mode of program operation.

- I/O time is required for paging operations. The amount of paging I/O time required is related to the number of page faults that occur and the speed of the paging I/O device(s) used. In OS/VS2 environments, the total I/O time required for paging includes some I/O time that is also required in OS MVT environments to load transient control program routines.

- Direct access storage is required for external page storage. The amount required depends on the amount of virtual storage that is to be supported and the way in which the particular operating system organizes and manages external page storage. (See the optional programming systems supplements for external page requirements by device type.)

- The amount of real storage required by the resident (fixed) control program is increased by the amount of real storage needed for additional routines and code that are included specifically to support a demand paged virtual storage environment.

The effect this additional use of hardware resources has on the performance of a given system configuration depends on the resource requirements of the job stream and the current utilization of system resources. To the degree that the additional required CPU and I/O time can be overlapped with existing CPU and I/O time that is currently unoverlapped, system throughput is not affected. System throughput will be affected by the increase in CPU and I/O time that cannot be overlapped.

When a virtual storage operating system is used with an existing system configuration, for example, and the same job stream is processed, performance is affected by the use of any new performance enhancements these operating systems provide as well as by an increase in resource utilization that is required to support a virtual storage environment. When a Model 168 replaces a Model 165, performance is also affected by the fact that the Model 168 has a faster internal performance than the Model 165.

Figure 30.15.1 conceptually illustrates possible system performance when a virtual storage operating system is installed on a Model 168 with the same amount of real storage and the same I/O device configuration as the replaced Model 165. A sample throughput for a Model 165 is shown in panel 1. (It is not meant to represent any specific Model 165 throughput.) Panels 2 and 3 illustrate the conditions under which existing performance can be maintained and the last two illustrate the conditions under which existing performance can be improved.

Panel 1

Sample existing CPU and I/O utilization and overlap for a Model 165.

EXISTING SYSTEM THROUGHPUT MAINTAINED

Panel 2

Some of the additional CPU and I/O time required is overlapped with previously unoverlapped I/O and CPU time (points A). Additional CPU and I/O time that cannot be overlapped (point B) is offset by a reduction in the amount of CPU and I/O time required to process the same job stream. Results are achieved in the same elasped time.

Panel 3

Additional CPU and I/O time required (dotted lines) is overlapped and offset by operating the system at a higher level of multiprogramming to achieve greater overlap. Results are achieved in the same elapsed time.

EXISTING SYSTEM THROUGHPUT IMPROVED

Panel 4

Unoverlapped CPU and I/O time required is exceeded by reductions in previously used CPU and I/O time. Better overlap of previously used CPU and I/O time is also achieved. Same results are achieved in less elapsed time.

Panel 5

A higher level of multiprogramming is used to perform more work and achieve better overlap of CPU and I/O time. More results are achieved in the same elasped time.

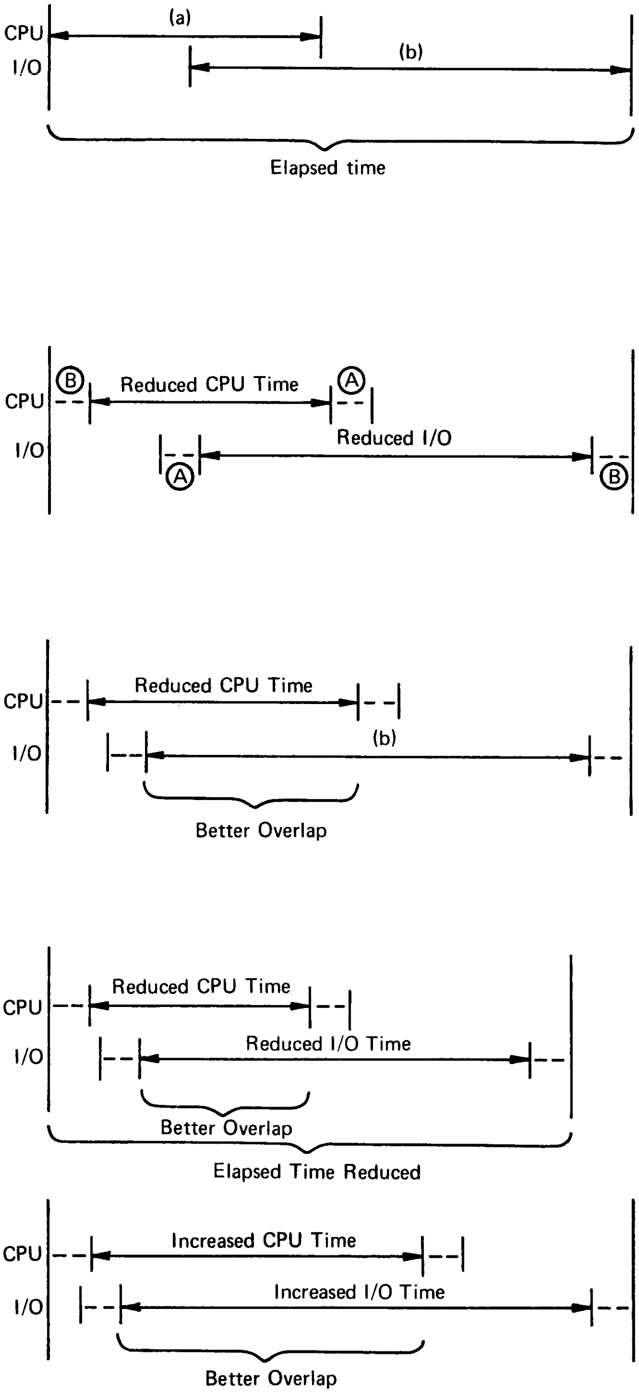


Figure 30.15.1. Possible system performance when a virtual storage operating system is used with a Model 168 with the same I/O configuration and real storage size as the replaced Model 165

Existing throughput is maintained if both of the following occur:

1. A portion of the additional CPU and I/O time required to support a virtual storage environment is overlapped with CPU and I/O time that previously was not overlapped, as shown by points A in panel 2.

2. The amount of additional CPU and I/O time that cannot be overlapped (shown by points B in panel 2) is offset by reductions in previously used CPU and I/O time that occur as a result of the faster internal performance of the Model 168 and use of new performance features of the virtual storage operating system, as shown in panel 2. The unoverlapped CPU and I/O time may also be offset by a combination of the faster internal performance of the Model 168 and the achievement of better overlap as a result of operating the system at a higher level of multiprogramming to process the same work (as shown in panel 3).

Existing system throughput can improve if (1) unoverlapped CPU and I/O time required to support a virtual storage environment is exceeded by reductions in previously used CPU and I/O time and/or if previously used CPU and I/O time are better overlapped (as shown in panel 4) or (2) a higher level of multiprogramming is used to perform more work and provide better CPU and I/O overlap in the same elapsed time (as shown in panel 5).

NEW FACTORS THAT AFFECT SYSTEM PERFORMANCE

In addition to the factors that affect system performance in a nonvirtual storage environment, the performance of a system in a virtual storage environment is affected by the relationship of the following factors: the speed and number of paging devices, the speed of the CPU, the size of real storage, the structure of the programs in the job stream, and the way in which real storage is organized and allocated by the virtual storage operating system. The interrelationship of each of these factors and their individual effect on performance, except for the last factor listed, are as follows (page replacement algorithms are not discussed):

Speed and Number of Paging Devices. A certain amount of I/O time is required to read in (or write out) a page using a given direct access device type. This time is a function of device type characteristics--seek time, rotation time, and data transfer rate. Assuming one page-in is performed at a time, no page-outs, and no contention for the paging device or its channel, a maximum paging rate, in terms of the number of page faults that can be serviced per time interval, could be calculated for a given device type. This rate could be improved by certain programming techniques, such as use of rotational position sensing when it is present, and initiation of multiple page-in and page-out requests with a single channel program. (Various techniques are implemented in OS/VS1 and OS/VS2.) The maximum paging capability of a given system can be increased by various means, such as using more than one paging device or using a faster paging device.

The paging characteristic of a virtual storage environment is the feature that permits an operating system to support virtual storage that is larger than real storage. The paging activity of a system begins to adversely affect system performance, however, once the CPU is in the position of frequently having to wait for paging I/O operations to complete. When requests for paging operations are permitted to occur faster than the paging rate the system can sustain, such that the system can do little or no processing except that related to paging, the system is in a paging-I/O bound situation and is said to be thrashing. When a thrashing condition exists, little or no productive work can be accomplished unless paging activity is reduced.

In order to prevent thrashing, the System/370 virtual storage operating systems monitor the activity of the system to determine when paging activity becomes excessive. At this point, the OS control program performs task deactivation. This involves placing a task (partition or region) in deactivated status and releasing the page frames currently allocated to the partition or region. These page frames are then available for allocation to other tasks to reduce paging activity. Later, when paging activity becomes sufficiently low, the deactivated partition or region is reactivated.

CPU Speed. An improperly balanced relationship between CPU speed and paging device speed can also cause the system to become I/O-bound as a result of paging. A Model 168 can execute a certain number of instructions during the time required to service a page-in request using a given direct access device type. A Model 168 can execute many more instructions during a page-in from a 2305 Model 2, for example, than can a Model 158. As long as there is useful work for the CPU to perform while paging operations occur, the system is not kept waiting for paging I/O. However, if the concurrently operating programs are constantly executing instructions faster than the pages they require can be brought into real storage, an excessively high paging rate can develop and task deactivation will be the result. In general, therefore, the larger scale System/370 models require faster paging devices to handle a particular page fault rate than do the smaller scale models.

Real Storage Size. The amount of real storage present in a system affects the number of page faults that occur when a given job stream is processed. If the amount of real storage present in the system is equal to the total amount of virtual storage being used by the concurrently executing tasks, no page faults occur for programs that have been fetched and initiated. When the amount of real storage present is less than the amount of virtual storage being used, page faults occur. The total number of page faults that occur for a given job stream is affected by the ratio of virtual storage used to real storage available.

Assuming the amount of virtual storage used in a given system remains the same, the virtual-to-real storage ratio can vary. This occurs while a given system experiences variations in the amount of real storage actually available for paging as the amount of fixed real storage changes during job stream processing. The real storage available for paging at any point in time is the difference between the amount of real storage in the system and the total amount of long- and short-term fixed real storage. For IBM-supplied virtual storage operating systems, the total amount of fixed real storage at any given time is the sum of the:

- Resident (fixed) control program size, which does not vary after IPL

- Amount of long-term fixed real storage required for control blocks, which can change as the level of multiprogramming changes in OS/VS1 and OS/VS2 environments

- Amount of short-term fixed real storage required for outstanding I/O operations that have virtual channel programs, which flucuates with the I/O activity of the system

- Amount of long-term fixed real storage required by the job steps executing in nonpaged mode, if any

- Amount of long-term fixed real storage required by programs that operate in paged mode but that have a portion of their partition or region always fixed (TCAM in OS/VS1 and OS/VS2, for example)

As the virtual-to-real storage ratio of a job stream increases, so usually does the page fault rate. In general, the page fault rate increases slowly for a while. At some point, the increase in page faults begins rising rapidly as the virtual-to-real storage ratio continues to increase. Figure 30.15.2, shown later, illustrates the general relationship between the number of page faults and the virtual-to-real storage ratio.

The amount of real storage available to process a given job stream also varies when a given job stream is processed on systems with various amounts of real storage, such as when a smaller scale system is used to back up a larger scale system.

The degree to which reducing the real storage available for paging affects the page fault rate depends on the paging activity pattern of the programs in a job stream. Therefore, the virtual-to-real storage ratio at the point at which a given number of page faults occurs will usually vary by job stream. The point can also be different for systems with similar paging activity patterns and the same amount of real storage installed, but with different amounts of long-term fixed real storage.

As the virtual-to-real storage ratio increases, because of a reduction in the real storage available (or an increase in the amount of virtual storage used), and the page fault rate increases, more demand is placed on the paging devices. If too small an amount of real storage is present in a system, this situation can cause the page fault rate to exceed the permissible rate and task deactivation will occur. In general, therefore, in order to obtain a certain level of performance, a configuration that supports a given job stream and virtual storage size may require more real storage when a relatively slower paging device is used than if a faster paging device is used.

Program Structure. The total amount of virtual storage a program uses is not nearly as significant a factor in system performance as the way in which virtual storage is used. That is, the pattern and frequency of reference to pages in a program has more effect on the number of page faults that occur than the total size of the program. For example, assume a case in which a program has a 100K virtual storage design point. If the program can be structured to execute as a series of logical phases of four or five pages each, and the pages of each logical phase reference only each other, no more than four or five page frames (8K to 10K or 16K to 20K of real storage, depending on page size) need be dynamically available to the program at one time and paging activity occurs only as the program progresses from one logical phase to the next. However, assume the program is structured such that during its execution each page of instructions constantly references a large number of different pages of instructions and data for very short durations on a highly random basis. An excessively high paging rate could occur if only four or five page frames were dynamically available to such a program at any time.

As indicated previously, most types of programs naturally have a locality of reference characteristic so that they can be structured to operate as a series of logical phases. In the simplest case, for example, a program can logically consist of an initialization phase, a main phase, one or more exception handling phases, and a termination phase. The total amount of virtual storage referenced in each logical phase usually varies but, generally, the amount is less than the total size of the program. In addition, the pages that are part of (referenced in) a given logical phase can usually be described as active or passive.

For the purpose of the discussion in this subsection, an active page is defined as one with a high probability of being referenced multiple times during execution of the logical phase, while a passive page has a low probability of being referenced more than once during execution of the phase. A logical phase experiences the least amount of paging activity as it executes when its active pages remain in real storage during its execution and its passive pages are paged in when required. A program uses real storage most efficiently when the active instructions and data in each logical phase are contained within the fewest number of pages possible.

The locality of reference characteristic does not apply to certain types of programs. For example, it does not apply to any program that is designed to optimize its performance at execution time by using the total amount of storage it has been allocated. This characteristic is usually true of sort/merge programs that initialize themselves to use all the storage made available to them in their partition or region during the sorting passes. The reference pattern for such a sort/merge is random and encompasses all the storage (and, therefore, all the pages) the program is assigned.

RELATIONSHIP BETWEEN VIRTUAL STORAGE SIZE AND SYSTEM PERFORMANCE

Assuming other required system resources are available, a given configuration can support a given virtual storage size and provide satisfactory performance when paging activity is kept at an acceptable level. Minimal paging activity occurs when enough real storage is present in the system to contain most or all of those pages of concurrently executing programs that are active at any given time. Paging activity then is required primarily for passive pages. Active pages are paged in (and later paged out as required) as the set of active pages for each program changes from one logical phase to another. The paging device(s) present must be capable of handling the demand for pages that results from the range of paging activity of the system.

As the amount of virtual storage used in a given system increases, the number of active and passive pages that the system must handle increases also. The ratio of active to passive pages will vary for a given increase in virtual storage, depending on how the additional virtual storage is used. As long as enough real storage is present to contain all or most of the increased number of active pages, the increase in paging activity required to support the additional virtual storage will be needed primarily for passive pages and should be relatively small. As soon as the use of more virtual storage causes the number of concurrently active pages to constantly exceed the capacity of real storage, the paging activity increase required to support the additional virtual storage becomes relatively large. As more and more active pages must be handled, paging activity could exceed the maximum paging capability of the system if task deactivation did not occur.

Figure 30.15.2 illustrates the increase in page faults that generally occurs as more virtual storage is used in a given system configuration. The curve begins at the point at which the amount of virtual storage used is equal to the amount of real storage present (virtual-to-real-storage ratio is 1). Paging activity begins as soon as the amount of virtual storage used exceeds the real storage present. As the virtual-to-real-storage ratio increases, so does paging activity. The system moves from passive paging activity (primarily paging of passive pages) into active paging (paging active pages in and out more of the time) and approaches the maximum paging capability of the system. As indicated previously, Figure 30.15.2 also illustrates the increase in page faults that generally occurs as less real storage is made available to support a given virtual storage size. The increase in page faults also causes the virtual-to-real storage ratio to increase.

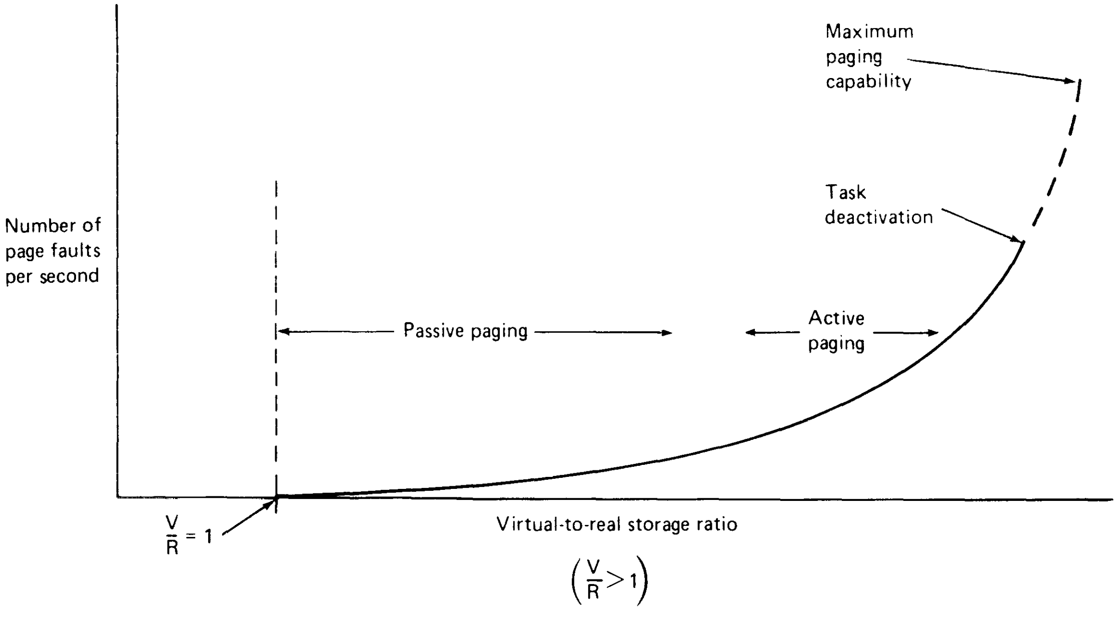


Figure 30.15.2. General effect on page faults of increasing the ratio of the virtual storage used to real storage present in the system

Figure 30.15.3 illustrates how the paging factor only generally affects system performance. Figure 30.15.5, shown later, illustrates system performance taking into account all factors. The curve shows the performance of the system when passive and active paging are occurring, relative to the virtual-to-real storage ratio. The use of virtual storage can be increased with little or no adverse effect on performance as long as paging remains in the passive area. This is true because in the passive paging area there is a relatively small amount of paging and a high probability that all or most paging processing (CPU and I/O time) can be overlapped with other processing. As paging activity increases, there is a higher probability that CPU processing will be held up waiting for a paging operation to complete. As the CPU enters the wait state more frequently to wait for paging I/O and less paging I/O is overlapped, the paging factor causes performance to degrade more rapidly.

The actual virtual-to-real storage ratio at the time active paging begins in Figures 30.15.2 and 30.15.3 is a variable and depends on the way in which virtual storage is used, that is, active-to-passive page ratio of concurrently executing tasks.

Figure 30.15.4 illustrates the way in which the paging factor only can affect system performance in a given configuration, based on the active-to-passive page ratio. If the ratio of active to passive pages for executing tasks is relatively high most of the time, as shown in curve 1, the virtual-to-real storage ratio at the point at which active paging begins will be relatively low. Performance drops very rapidly in this case as more virtual storage is used. This happens because the increased paging processing (I/O and CPU time) cannot be overlapped with other processing. This situation may apply to an installation initially when a switch from a nonvirtual storage to a virtual storage environment is made and more virtual storage is used, since existing programs were structured for optimum performance in a given partition or a region size rather than for optimum performance in a virtual storage environment.

If the active-to-passive page ratio for the system is low, as shown in curve 3, the virtual-to-real storage ratio can be relatively high when active paging begins. The performance curve stays flatter longer as virtual storage is increased when the active-to-passive page ratio is low. This situation can apply to an installation in which all executing programs are structured such that real storage requirements and page faults are minimized. An installation that continues executing all or most existing programs as they are presently designed and that structures new applications for optimum performance (low active-to-passive ratio) may be more common. Such installations may experience a virtual-to-real storage ratio somewhere between the low and the high extremes possible for a given job stream, as shown in curve 2.

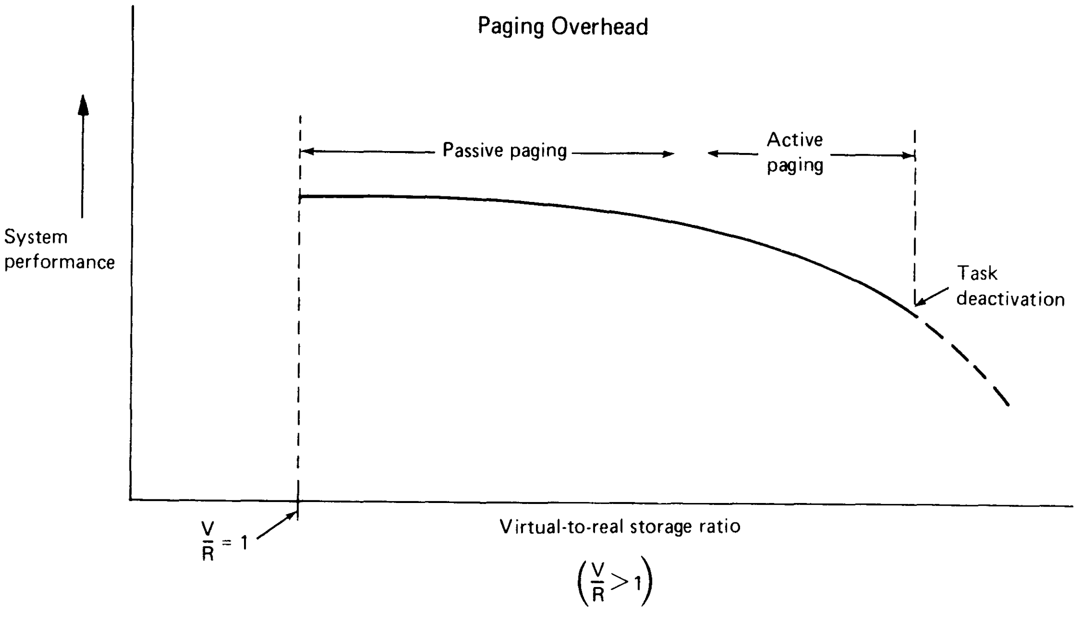


Figure 30.15.3. General effect on system performance of the paging factor only

The amount of virtual storage used in a system can be increased in several ways. First, the size of existing application programs can be increased by the addition of new functions. Second, the level of multiprogramming or multitasking can be increased, assuming other required resources, such as CPU time and I/O devices, are available. Third, the size of existing application programs can be expanded by (1) restructuring programs with a planned overlay or a dynamic structure to take them out of these structures and (2) combining two or more job steps within a job into one logical job step. The active-to-passive ratio of the additional pages the system must handle will usually be higher when the level of multiprogramming is increased than when existing jobs are restructured.

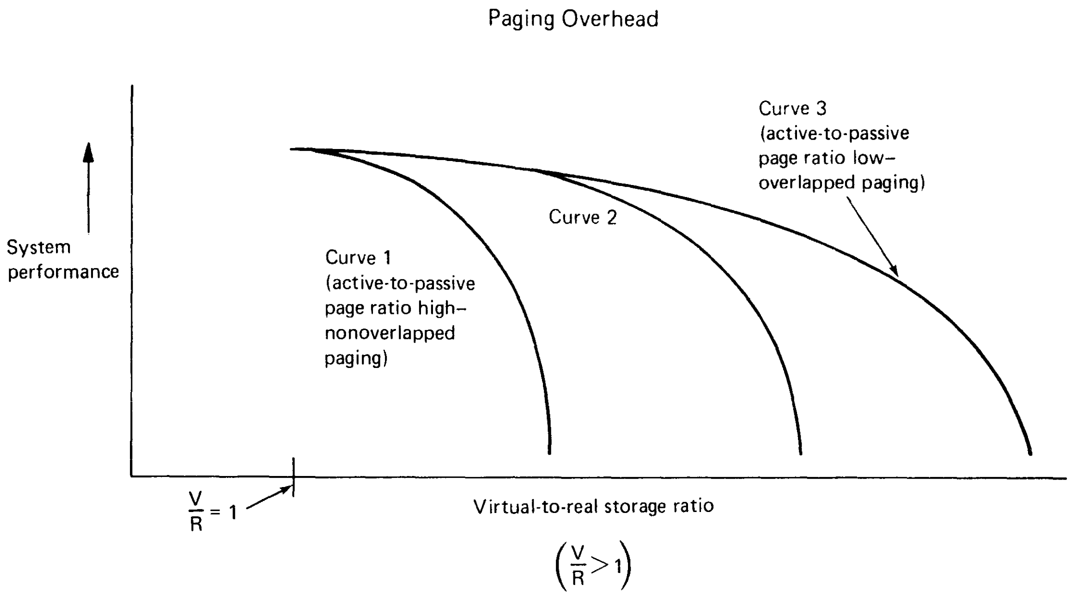


Figure 30.15.4. General effect of the paging factor on system performance with various active-to-passive page ratios.

The way in which an installation should view the amount of virtual storage used and system performance for a given configuration, taking all performance factors into account, is illustrated in Figure 30.15.5. The increased use of virtual storage is beneficial to system performance up to a point. Thereafter, additional virtual storage can be used to handle additional functions at a variable cost in system performance. In reality, the virtual-to-real storage ratio and the page-fault rate vary during system processing as the amount of virtual storage used (out of the total amount supported) and the amount of real storage available for paging vary. Best overall system performance is achieved when paging activity falls most of the time in the area identified on the curve as the operating range. More significant performance reduction begins when active paging is experienced.

Occasional active paging on an exception basis should be acceptable. More frequent active paging can be performed to achieve a desired function that does not justify changing the system configuration. However, when paging activity in a system is constantly at the point at which task deactivation occurs, system configuration changes should be made to improve system performance. Such changes might be the addition of more real storage, the addition of more or faster paging devices, or installation of a faster CPU. A history of the paging activity of the system can be maintained by recording the paging statistics provided by OS/VS1 and OS/VS2.

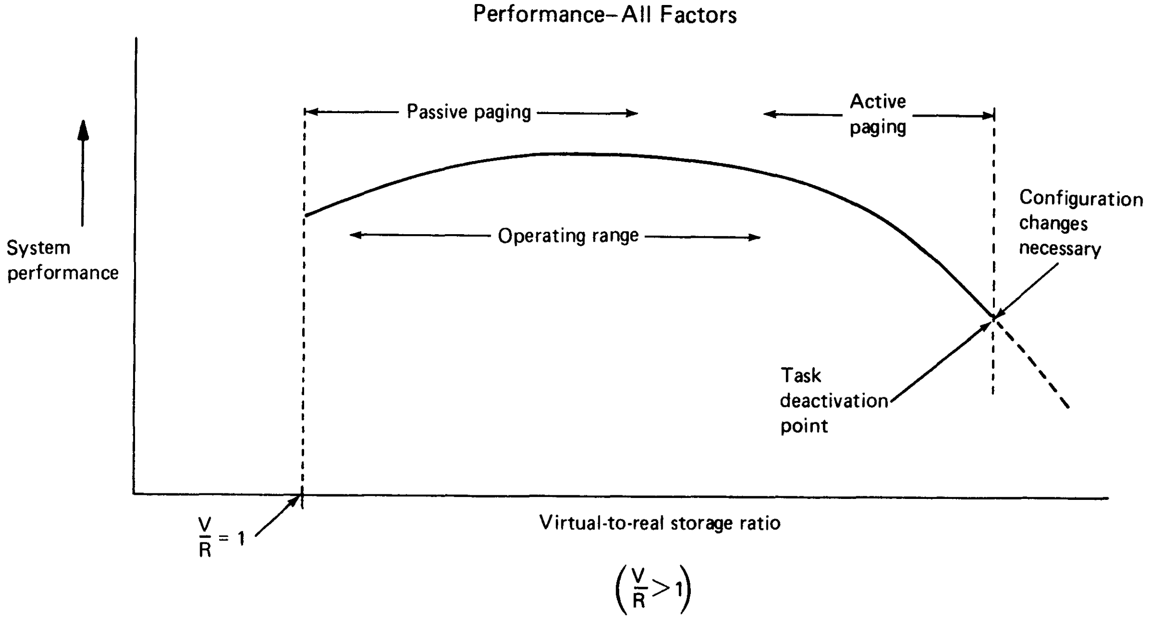


Figure 30.15.5. General system performance curve for a virtual storage environment

INCREASING SYSTEM PERFORMANCE IN A VIRTUAL STORAGE ENVIRONMENT

The IBM-supplied virtual storage operating systems are designed to provide an acceptable level of performance when existing problem programs are run without modification. However, given the additional resource requirements of virtual storage support and the new factors that affect system performance in a virtual storage environment, once a virtual storage operating system is installed (either on an existing configuration or a larger configuration) certain steps can be taken to improve performance or to achieve optimum performance. The benefit to be achieved by taking any one of the steps outlined must be evaluated on an installation basis, taking the specific configuration and operating environment into account. Some steps, for example, are more practical for large configurations than for small configurations. The following can be done:

- Use larger I/O buffers. This step is practical primarily for sequential data sets and can be used most effectively when previous real storage limitations prevented the use of larger buffer sizes, in general, and optimum buffer sizes for disk data sets. In addition to reducing the total I/O time required to process a data set, as would occur in a nonvirtual storage environment, increasing buffer size reduces the number of I/O requests required to process the data set and, thereby, reduces the CPU time required for channel program translation and page fixing. This technique should be used taking into account the amount of real storage present in the system. If the buffer size of several data sets that are being processed concurrently is increased considerably or made large initially, the amount of real storage that must be short-term fixed increases considerably also and potentially increases the number of active pages. This may adversely affect system performance if the system has a relatively limited amount of real storage available for paging.

- Increase the page fault handling capability of the system when paging activity constantly causes task deactivation. This can be accomplished by (1) using a direct access device for paging that is faster than the currently used paging device, (2) allocating more direct access devices for paging to enable more overlap of paging activity, or (3) reducing or eliminating contention for the existing paging device(s). Contention for the paging device can be relieved by using dedicated paging devices, or reducing the amount of other I/O activity on the channel to which the paging device is attached, or dedicating a channel to paging. Alternatively, the same paging device configuration can be maintained while page fault occurrence is decreased by the addition of real storage.

- Use code that does not modify itself. Use of this type of code can reduce the amount of page-out activity required. Such code can be produced using OS PL/I and the OS Assembler Language.

- Execute programs in nonpaged mode only when actually required. Use of nonpaged mode should be limited because the amount of real storage available for paging operations during the operation of a nonpaged program is reduced by the size of the program and can affect system performance. If a nonpageable program is to be present in a system for an extended period of time or at all times, it should be considered part of the fixed real storage requirement so that the amount of real storage actually available for paging can be more accurately determined.

- Structure new application programs to operate efficiently in a paging environment. This is done by structuring programs to achieve a reasonable balance between page faults and real storage requirements. The extent to which this is done can vary widely by installation. The benefits that can be obtained should be evaluated in light of the additional programmer effort required. In this respect, deciding on the degree to which programs should be structured for efficient operation in a paging environment is similar to deciding how a high-level language should be used. The emphasis can be on most efficient program execution, which can require more programmer effort, or on most efficient use of programmer time, which can result in less efficient programs. Alternatively, there can be a tradeoff between programmer time and efficient programs (only the most frequently used programs are optimized, for example).

  Many of the general program structure techniques discussed do not require a large amount of additional effort or knowledge on the part of programmers--only that they adopt a particular programming style. All of the suggested techniques can be used by Assembler Language programmers. Some can be used with certain high-level languages and not with others. More of the suggested techniques can be used in PL/I programs than in other high-level language programs.

  Two major steps can be taken to structure programs to use real storage most efficiently and to incur the smallest possible number of page faults. The first is to adopt a certain programming style, one aspect of which is similar to the style that has been encouraged with System/360 and System/370, namely, that of modular programming. The second is to package program code and data within page boundaries. The objective of improving programming style is to construct a program that consists of a series of logical processing phases each of which contains a relatively small number of active pages. The objective of packaging code within pages is to group active code together to avoid crossing page boundaries such that more real storage than is really necessary is required to contain the active pages of a logical phase.

In order to cause references to active instructions and data to be localized, the following general rules should be applied to programs:

1. A program should consist of a series of sequentially executed logical phases or--in System/370 programming terminology--a series of subroutines or subprograms. The mainline of the program should contain the most frequently used subroutines in the sequence of most probable use, so that processing proceeds sequentially, with calls being made to the infrequently used subroutines, such as exception and error routines. This structure contrasts with one in which the mainline consists of a series of calls to subroutines. Frequently used subroutines should be located near each other. Infrequently used subroutines that tend to be used at the same time whenever they are executed should be located near each other also.

2. The data most frequently used by a subroutine should be defined together so that it is placed within the same page, or group of pages, instead of scattered among several pages. If possible, the data should be placed next to the subroutine so that part or all of the data is contained within a page that contains active subroutine instructions (unless the routine is to be written such that it is not modified during its execution). This eliminates references to more pages than are actually required to contain the data and tends to keep the pages with frequently referenced data in real storage.

3. Data that is to be used by several subroutines of a program (either in series or in parallel by concurrently executing subtasks) should be defined together in an area that can be referenced by each subroutine.

4. A data field should be initialized as close as possible to the time it will be used to avoid a page-out and a page-in between initialization and first use of the data field.

5. Structures of data, such as arrays, should be defined in virtual storage in the sequence they will be referenced, or referenced by the program in the sequence in which a high-level language stores them (by row or by column for arrays, for example).

6. Subroutines should be packaged within pages when possible. For example, avoid starting a 1500-byte subroutine in the middle of a 2K page so that it crosses a page boundary and requires two page frames instead of one when it is active. Subroutines that are smaller than page size should be packaged together to require the fewest number of pages, with frequently used subroutines placed in the same page when possible. This applies to large groups of data as well. The linkage editor supplied with OS/VS1 and OS/VS2 has new control statements that can be used to cause CSECTS and COMMON areas to be aligned on page boundaries, and to indicate the order in which CSECTS are placed in the load module. This linkage editor facility can be used with certain high-level language programs that contain multiple CSECTS (such as PL/I and ANS COBOL) as well as with Assembler Language programs.

- Use the OS PL/I Optimizing Compiler instead of OS PL/I F. The code produced by this language translator has characteristics that makes it more suited to a virtual storage environment than the code produced by PL/I F. First, generated code is grouped into functionally related segments, by PROCEDURE and DO group, for example, which can help reduce paging. When PL/I allocates buffers and I/O control blocks, they are packed together and can potentially require fewer pages than if no attempt was made to define them together. Reentrant code can be produced by the OS PL/I Optimizing Compiler, and its library routines are reentrant. This reduces page-out requirements. User-written reentrant PL/I routines that are required by concurrently executing problem programs can be made resident in virtual storage and shared to reduce real storage and paging requirements for active pages of these routines.

- Use the shared library feature of the OS PL/I Optimizing Compiler and the COBOL Library Management Facility of the OS ANS COBOL language translator to make library modules resident in virtual storage so they can be shared by concurrently executing problem programs. Pages containing active libraray modules will tend to remain in real storage and thereby reduce paging and real storage requirements for these modules.

- Restructure existing application programs to incur as few page faults as possible, use the least amount of real storage, and take advantage of the program structure facilities that a virtual storage environment offers. This can be accomplished by (1) using the techniques described above, (2) taking planned overlay and dynamic structure programs out of these structures, and (3) combining into one logical step two or more steps of a job that would have been one job step if the required real storage were available. The last technique can eliminate redundant I/O time that is currently used to read the same sequential input file into two or more job steps, and to write intermediate results from one job step in one or more sequential data sets for input to the next job step.

- Increase the level of multiprogramming in the system. This can be accomplished by (1) performing more peripheral I/O operations concurrently (more readers and writers in OS), (2) operating more regions or partitions concurrently, or (3) increasing the use of multitasking (structuring a TCAM message processing program to use multitasking to enable several different types of transactions to be processed concurrently, for example).

  System throughput can be improved in a virtual storage environemnt if a higher level of multiprogramming causes more CPU and I/O time to be overlapped, which results in more effective utilization of available system resources. When a larger number of tasks are in the system under these conditions, the less chance there is for the CPU to enter the wait state because no task is ready to execute. Better utilization of real storage in a virtual storage environment can enable more tasks to be present in the system.

  In order to achieve performance gains by increasing the level of multiprogramming, the potential for more overlap of CPU and I/O time must exist in a system and/or the potential must exist for reduction of I/O time via increased overlapping of channel activity and reductions in unoverlapped seek time (that can result from new system performance enhancements). The required hardware resources, such as CPU time, real storage, I/O devices, and direct access storage, must be available as well. The most critical resource in this situation is available CPU time. As the percentage of CPU utilization gets higher, there is less potential for increasing throughput via increasing the level of multiprogramming.

The information presented in this subsection is designed to enable the reader to more fully understand the way a system operates in a virtual storage environment and the facts that influence system performance. Understanding the environment and knowing the actions that can be taken to increase system performance will enable each installation to quantify the amount of effort that is to be devoted to optimizing the performance of a virtual storage operating system.

SECTION 50: I/O DEVICES

50:05 I/O DEVICE SUPPORT

All I/O devices, consoles, and telecommunications terminals that can be attached to the Model 165 can be attached to the Model 168. However, all I/O devices supported by OS MFT and MVT are not also supported by OS/VS1 and OS/VS2. (See the optional programming systems supplements for I/O device support.) Model 65 devices that are not part of the standard Model 165 I/O configuration are not part of the standard Model 168 I/O configuration. The only difference between Models 165 and 168, as far as I/O device attachability is concerned, is inclusion of the capability of connecting 3330-series direct access storage to the Model 168 via the Integrated Storage Control feature.

50:10 INTEGRATED STORAGE CONTROL FEATURE FOR 3330-SERIES DISK STORAGE

Optionally, one Integrated Storage Control (ISC) feature can be installed on a Model 168 to attach 3330-series disk storage to one or two 2880 Block Multiplexer Channels. Attachment of 3330-series disk storage via 3830 Storage Control, Models 1 and 2, is possible as well.

The Integrated Storage Control feature includes dual direct access storage controls, each of which is functionally like 3830 Storage Control Model 2 except for the following:

- The Integrated Storage Control feature is contained in the main frame of the Model 168 and is powered by the Model 168 CPU.

- The Two-Channel Switch, Additional feature (that provides four-channel switching) cannot be attached to the storage controls in the ISC feature.

Both logical storage controls in the ISC feature can be attached to the same 2880 channel or they can be attached to two different 2880 channels connected to the Model 168. Each logical storage control can have attached a maximum of two 3330-series strings of up to eight drives each. (The first module in each string must be a 3333 Disk Storage and Control unit.) Therefore, up to 32 drives (four strings) can be attached to the Model 168 via the ISC feature. The 3330-series drives operate just as if they were attached via 3830 Storage Control Model 2. That is, when multiple requesting is used, each logical storage control within the ISC can handle up to 16 channel programs concurrently, one on each of its drives, and only one of the 16 drives can be transferring data at a time. When a malfunction occurs, diagnostics can be run on one logical storage control and its drives, while normal operations take place on the other logical storage control in the ISC.

The Two-Channel Switch optional feature is also available for the ISC feature. When installed, this feature provides a two-channel switching capability for both of the logical storage controls. The Two-Channel Switch feature permits each integrated storage control unit to be attached to two channels in the same Model 168 or to one channel in the Model 168 and one channel in another System/370.

The ISC feature provides lower cost attachment of 3330-series disk storage than 3830 Storage Control Model 2 when two storage control units are required, and floor space is saved since the ISC is in the Model 168 CPU.

SECTION 70: COMPARISON TABLES

These tables have been included for quick reference. The first compares hardware features of the System/360 Model 65 and System/370 Models 165, 165 II, and 168. The second compares OS MFT, MVT, VS1, and VS2 support of the Model 168.

70:05: COMPARISON TABLE OF HARDWARE FEATURES OF THE SYSTEM/360 MODEL 65 AND SYSTEM/370 MODELS 165, 165 II, AND 168

Hardware Feature	System/360 Model 65	System/370 Model 165	System/370 Model 165 II	System/370 Model 168
I. CPU				
A. BC mode of system operation	Comparable to BC mode	Standard	Standard	Standard
B. EC mode of system operation	Not implemented	Not implemented	Standard	Standard
C. Instruction set				
1. Standard set (binary arithmetic)	Standard	Standard	Standard	Standard
2. Decimal arithmetic	Standard	Standard	Standard	Standard
3. Floating-point arithmetic	Standard	Standard	Standard	Standard
4. Extended precision floating-point	Not available	Standard	Standard	Standard
5. New instructions a. COMPARE LOGICAL CHARACTERS UNDER MASK COMPARE LOGICAL LONG HALT DEVICE INSERT CHARACTERS UNDER MASK LOAD CONTROL MONITOR CALL MOVE LONG SET CLOCK SHIFT AND ROUND DECIMAL START I/O FAST RELEASE STORE CHANNEL ID STORE CHARACTERS UNDER MASK STORE CLOCK STORE CONTROL STORE CPU ID	Not available	Standard (except for MONITOR CALL)	Standard	Standard
b. LOAD REAL ADDRESS PURGE TLB RESET REFERENCE BIT SET CLOCK COMPARATOR SET CPU TIMER STORE CLOCK COMPARATOR STORE CPU TIMER STORE THEN AND SYSTEM MASK STORE THEN OR SYSTEM MASK	Not available	Not available	Standard	Standard

Hardware Feature	System/360 Model 65	System/370 Model 165	System/370 Model 165 II	System/370 Model 168
D. Overlap of instruction fetching and preparation with instruction execution	Instruction unit normally prepares one instruction at a time. Imprecise interruptions occur only for storage violations.	Instruction unit can process several instructions while execution unit executes one instruction. Imprecise interruptions can occur.	Same as Model 168	Same as Model 165 except that instruction and execution unit implementation is enhanced and imprecise interruptions cannot occur.
E. High-speed multiply	Not available	Optional	Optional	Optional
F. CPU cycle time	200 nanoseconds, 8-byte data path	80 nanoseconds, 8-byte data path	Same as Model 165	Same as Model 165
G. Dynamic address translation	Not available	Not available	Standard	Standard
H. Interval timer	Standard (16.6 ms resolution)	Standard (3.33 ms resolution)	Standard (3.33 ms resolution)	Standard (3.33 ms resolution)
I. Time of day clock	Not available	Standard	Standard	Standard
J. CPU timer and clock comparator	Not available	Not available	Standard	Standard
K. Monitoring feature	Not available	Not available	Standard	Standard
L. Program event recording	Not available	Not available	Standard	Standard
M. Direct control	Optional	Standard	Standard	Standard
N. Interruption for SSM instruction	Not implemented	Not implemented	Standard	Standard
O. Compatibility features (all are optional and mutually exclusive)	1. 7070/7074 2. 7080 (for both 705 and 7080) 3. 709/7040/7044/ 7090/7094/7094II	1. 7070/7074 2. 7080 (for both 705 and 7080) 3. 709/7090/7094/ 7094II (does not include 704, 7040, 7044)	Same as Model 165	Same as Model 165
P. Control logic	Microprogram in ROS	Microprogram in capacitor ROS and monolithic WCS.	Same as Model 168	Microprogram in monolithic ROS and monolithic WCS
Q. Instruction retry by hardware	No	Yes	Yes	Yes
R. Machine check interruption	One level of machine check provided for all machine errors and one machine check mask	Occurs after corrected and uncorrected errors. There are four types of machine check and many are individually maskable.	Same as Model 168	Same as Model 165, except five types of machine check are implemented and more data is logged by a Model 168.

Hardware Feature	System/360 Model 65	System/370 Model 165	System/370 Model 165 II	System/370 Model 168
S. Fixed storage area size in lower storage (including logout area for machine and channel errors)	328 bytes including CPU and channel logouts	1504 bytes reducible to 512 if the extended logout area of 992 bytes is moved	Same as Model 168	1928 bytes reducible to 512 if the extended logout area of 1416 bytes is moved
T. Multiprocessor systems	1. Multisystem optional feature permits interconnection of two Model 65s. Main storage is shared (512K or more). Direct control is required. 2. The support or main processor in an ASP configuration can be a Model 65. Two or three systems are connected via a Channel-to-Channel Adapter.	1. A multisystem feature is not available 2. A Model 165 can be a support or a main processor in an ASP configuration	Same as Model 165	Same as Model 165
II. STORAGE				
A. Processor (main storage sizes	256K	-	-	-
	512K	512K	-	-
	768K	-	-	-
	1024K	1024K	1024K	1024K
		1536K	-	-
		2048K	2048K	2048K
		3072K	3072K	3072K
				4096K
B. Type of processor storage	Ferrite cores	Ferrite cores	Ferrite cores	Monolithic technology
C. Processor storage cycle	750 nanoseconds (for 8 bytes). Two-way interleaving of sequential accesses other than by the channels is provided.	2 microseconds. Storage is 4-way doubleword interleaved for CPU and channel requests. 32 bytes can be obtained every two microseconds.	Same as Model 165	Storage is 4-way doubleword interleaved. Read/write cycle for 8 bytes on a doubleword boundary is 480 nanoseconds. Partial write requires 800 nanosecond cycle.
D. High-speed buffer storage	No	8K is standard, 8K more can be added. 80 nanosecond cycle.	Same as Model 168	Same as Model 165. Buffer row deletion is implemented also.

Hardware Feature	System/360 Model 65	System/370 Model 165	System/370 Model 165 II	System/370 Model 168
E. Processor storage validity checking	Parity checking by byte. No hardware error correction is provided.	ECC checking on a doubleword. Single-bit errors are corrected by hardware.	Same as Model 165	Same as Model 165
F. Byte-oriented operands	No	Standard	Standard	Standard
G. Store and fetch protection	Standard	Standard	Standard	Standard
H. Shared processor storage	Optional (Model 65 system shares 2361 Core Storage with a Model 50, 65, or 75)	Not available	Not available	Not available
I. 2361 Core Storage	Optional. Up to 8 million bytes can be attached.	Cannot be attached	Cannot be attached	Cannot be attached
III. CHANNELS				
A. Total number per system	Up to 7	1. Up to 7 standard 2. Up to 12 with Extended Channels optional feature	Same as Model 165	Same as Model 165
B. 2870 Multiplexer Channel	One or two can be attached	Same as Model 65	Same as Model 65	Same as Model 65
C. 2860 Selector Channel (1.3 MB)	A maximum of 6 can be attached	Same as Model 65	Same as Model 65	Same as Model 65
D. 2880 Block Multiplexer Channel (1.5 MB). Two-Byte Interface feature permits a 3.0 MB data rate	Cannot be attached	A maximum of 6 can be attached without the Extended Channels feature, a maximum of 11 with this feature	Same as Model 165	Same as Model 165
E. Channel dual I/O bus	Not available	Not implemented	Not implemented	Standard
F. Maximum aggregate data rate for channels	In excess of 4 MB for one 2870 and six 2860's	In excess of 9 MB with twelve channels	Same as Model 165	16 MB
G. Channel retry data provided after channel error	Yes	Yes	Yes	Yes
H. Channel-to-channel adapter	Optional on 2860	Optional on 2860	Optional on 2860	Optional on 2860
I. Channel indirect data addressing	Not available	Not available	Optional (required by the virtual storage operating systems)	Optional (required by the virtual storage operating systems)

Hardware Feature	System/360 Model 65	System/370 Model 165	System/370 Model 165 II	System/370 Model 168
IV. OPERATOR CONSOLE DEVICES	1. 1052 Printer-Keyboard (optional) 2. Second 1052 Printer-Keyboard is optional 3. A 2250 Display Unit and a remote 2150 Console are optional 4. Other devices can be used as primary and secondary consoles	1. Stand-alone 3066 Model 1 System Console is required. It includes: a. A CRT-Keyboard combination for operator/system communication b. An indicator viewer c. A microfiche document viewer d. A processor storage configuration plugboard e. A system activity meter f. A device for loading WCS and microdiagnostics The store status function is not provided. 2. Optionally, other devices can be used as secondary consoles as listed for the Model 65	1. Stand-alone 3066 Model 1 System Console is required. The store status function is supported.	1. Stand-alone 3066 Model 2 System Console provides same features as 3066 Model 1 and store status function. Other consoles can be attached as for Model 165.
V. I/O DEVICES				
A. 3505 Card Reader and 3525 Card Punch	No	Yes	Yes	Yes
B. 3211 Printer	Yes	Yes	Yes	Yes
C. 3803/3420 Magnetic Tape Subsystem	Yes	Yes	Yes	Yes
D. Direct access devices (2311,2314,2303, 2301, and 2321)	All attach	Same as Model 65	Same as Model 65	Same as Model 65
E. 3330-series with RPS and multiple requesting	No	Yes	Yes	Yes
1. 3830 Storage Control Model 1	-	Yes	Yes	Yes
2. 3830 Storage Control Model 2	-	Yes	Yes	Yes
3. Integrated Storage Control feature	-	No	No	Yes
F. 2305 facility Models 1 and 2 with RPS and multiple requesting	No	Yes on 2880	Yes on 2880	Yes on 2880

70:10: OS SUPPORT OF THE MODEL 168

Hardware Feature	OS MFT and MVT	OS/VS1	OS/VS2
I. CPU			
A. Mode of system operation	BC mode only. Up to 15 problem program partitions or regions.	EC and DAT modes only. One virtual storage of up to 16 million bytes is supported. Up to 15 problem program partitions. Up to 37 system task partitions.	EC and DAT modes only. One virtual storage of 16 million bytes is supported. Up to 63 problem program regions of which up to 42 can be TSO foreground regions.
B. Instruction set			
1. Standard set (binary arithmetic)	All languages	All languages	All languages
2. Decimal arithmetic	All languages except FORTRAN	All languages except FORTRAN	All languages except FORTRAN
3. Floating-point arithmetic	All languages except RPG	All languages except RPG	All languages except RPG
4. Extended precision floating-point	Assemblers F and H, PL/I Optimizing Compiler, PL/I Checkout Compiler, FORTRAN H, FORTRAN H-Extended	Same as MFT and MVT	Same as MFT and MVT
5. New instructions a. COMPARE LOGICAL CHARACTERS UNDER MASK COMPARE LOGICAL LONG INSERT CHARACTERS UNDER MASK LOAD CONTROL MONITOR CALL MOVE LONG SET CLOCK SHIFT AND ROUND DECIMAL START I/O FAST RELEASE STORE CHANNEL ID STORE CHARACTERS UNDER MASK STORE CLOCK STORE CONTROL STORE CPU ID	Mnemonics in Assemblers F and H. Option to generate certain instructions in ANS Full COBOL Version 3 (CLCL, MVCL, ICM, SRP)	Same as OS MFT and MVT	Same as OS MFT and MVT

Hardware Feature	OS MFT and MVT	OS/VS1	OS/VS2
b. LOAD REAL ADDRESS PURGE TLB RESET REFERENCE BIT SET CLOCK COMPARATOR SET CPU TIMER STORE CLOCK COMPARATOR STORE CPU TIMER STORE THEN AND SYSTEM MASK STORE THEN OR SYSTEM MASK	Not supported	All are supported by the System Assembler	Same as OS/VS1
C. Interval timer	Supported for timing facilities, except for time of day	Same as MFT and MVT	Not supported
D. Time of day clock	Supported for time of day	Same as MFT and MVT	Supported for time of day
E. Clock comparator and CPU timer	Not supported	Not supported	Supported for timing facilities except for time of day
F. Expanded machine check interruptions	Supported by MCH	Same as MFT and MVT	Same as MFT and MVT
G. Monitoring feature	Supported by GTF and an Assembler mnemonic	Same as OS MFT and MVT	Same as OS MFT and MVT
H. Program event recording	Not supported	Supported by Dynamic Support System	Supported by Dynamic Support System
I. Interruption for SSM instruction	Not supported	Supported	Supported
J. Compatibility features	All are supported	All are supported	All are supported
II. STORAGE			
A. Real storage sizes (1024K to 4096K)	All are supported	All are supported	All are supported
B. Byte-oriented operands	Programmers can use the byte alignment hardware facility in Assembler programs	Same as MFT and MVT	Same as MFT and MVT
C. Store and fetch protection	Store protect only is supported	Store protect only is supported	Store and fetch protection are supported for all regions

Hardware Feature	OS MFT and MVT	OS/VS1	OS/VS2
III. CHANNELS			
A. Byte multiplexer channels	One or two are supported	One or two are supported	One or two are supported
B. Block multiplexer and selector channels	Supported	Supported	Supported
C. Channel retry performed	Yes	Yes	Yes
D. Channel indirect data addressing	Not supported	Supported	Supported
IV. CONSOLES			
A. 3066 Console	Supported. MCS and DIDOCS required	Same as MFT and MVT	Same as MFT and MVT
B. Alternate and additional consoles supported	Yes	Yes	Yes
V. I/O DEVICES			
A. 3505 Card Reader and 3525 Card Punch	Supported	Supported	Supported
B. 3211 Printer	Supported	Supported	Supported
C. 3803/3420 Magnetic Tape Subsystem	Supported	Supported	Supported
D. 2314/2319 facilities	Supported for system residence, data sets, SYSIN devices, and SYSIN and SYSOUT data sets	Supported for system residence, data sets, paging devices, JES spooling devices, and SYSIN devices	Supported for system residence, data sets, paging devices, SYSIN and SYSOUT data sets, and SYSIN devices
E. 3330-series with RPS and multiple requesting attached via 3830 Storage Control Model 1, 3830 Storage Control Model 2, or Integrated Storage Control	Supported as V.D. above. RPS, multiple requesting, and sixteen-drive addressing are supported.	Same as V.D. above. RPS, multiple requesting and sixteen-drive addressing are supported.	Same as V.D. above. RPS, multiple requesting, and sixteen-drive addressing are supported.
F. 2305 Facility Models 1 and 2 with RPS and multiple requesting	Supported for system residence, data sets, and SYSIN/SYSOUT data sets. RPS and multiple requesting are supported.	Same as V.D. above except for SYSIN devices. RPS and multiple requesting are supported.	Same as V.D. above except for SYSIN devices. RPS and multiple requesting are supported.

INDEX (Sections 01 to 70)

SECTION 90: OS/VIRTUAL STORAGE 1 FEATURES

If required, the OS/Virtual Storage 1 Features Supplement, GC20-1752, should be inserted here.

This page intentionally left blank

SECTION 100: OS/VIRTUAL STORAGE 2 FEATURES

If required, the OS/Virtual Storage 2 Features Supplement, GC20-1753, should be inserted here.

This page intentionally left blank

SECTION 110: VIRTUAL MACHINE FACILITY/370 FEATURES

If required, the *Virtual Machine Facility/370 Features Supplement* should be inserted here. Availability of this supplement is to be announced.

This page intentionally left blank

READER'S COMMENT FORM

A Guide to the IBM System/370 GC20-1755-0

Model 168

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold fold

fold fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.

GC20-1755-0

Your comments, please . . .

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 812
1133 Westchester Avenue
White Plains, New York 10604

Fold Fold

**International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)**

**IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)**

A Guide to the IBM System/370 Model 168 Printed in U.S.A. GC20-1755-0

GC20-1755-0

A Guide to the IBM System/370 Model 168 Printed in U.S.A. GC20-1755-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)