Section 3

The IBM System/360— a series of planned machines which span a wide performance range

In this introduction, besides making some general comments on the IBM System/360, we will attempt an analysis of the performance and costs of the series. Performance is notoriously difficult to measure, as we noted in Chap. 3, and costs are even more so. With respect to the latter, what is publicly available are price data, not manufacturing-cost data.

These prices reflect not only marketing policies but also accounting policies within the organization for the attribution of costs to product lines. For example, we have had to determine Pc and Mp prices on the basis of incremental Mp prices within a C. Nevertheless, the 360 series provides two things which make a comparative analysis worthwhile. First, the common ISP makes simple performance measures more comparable; second, the common manufacturer makes relative prices more a reflection of relative costs than would otherwise be the case. Neither of these aspects is perfect, as we will note at several points in the discussion. Nevertheless, the 360 series provides as good an opportunity to attempt cost/performance analysis as we know. Indeed, this opportunity has already been grasped in a paper by Solomon [1966], which we have found very valuable and use to provide a basis of Pc power.

Analyses of the type we attempt here produce only rather crude pictures and are subject to question if all the input data are not very carefully checked. We have not done the latter, depending instead on published sources. For the purpose of this book, illustration of the style of analysis seems sufficient. In addition, using a performance measure based only on Pc power measurements, as we do here, leaves many questions unanswered because it does not address the soft areas of analysis relating to throughput, task environment, and the operating system software.

Unlike the other introductions in this book, the reader may find it worthwhile to scan this one, read the chapters in the section, and then return to this introduction when the system has become somewhat familiar.

The IBM System/360 is the name given to a third-generation series of computers which constitute the current primary IBM product line. They all have a common ISP but differ in interpreter speeds and PMS structure. Many PMS elements are used in common, particularly K's, Ms's, and T's.

The System/360 series is presented both because IBM's market dominance makes it the most prevalent current computer and because its implementations span the largest performance and price range of any series. The C('360) models should be compared with one another (Table 1) to be aware of their capabilities. Their introduction dates and their relationship are shown in Fig. 1. Chapters 43, 44, and 32 discuss the logical structure of the system, the implementations,[1] and the microprogrammed Model 30.

A succinct description of the design goals and innovations is given in the abstract of the paper Architecture of the IBM System 360 [Amdahl et al., 1964a]:

[1]Chapters 43 and 44 are from *IBM Systems Journal*, vol. 3, no. 2, 1964, which was devoted exclusively to the System/360. The other articles (listed in the bibliography) are recommended for additional details.

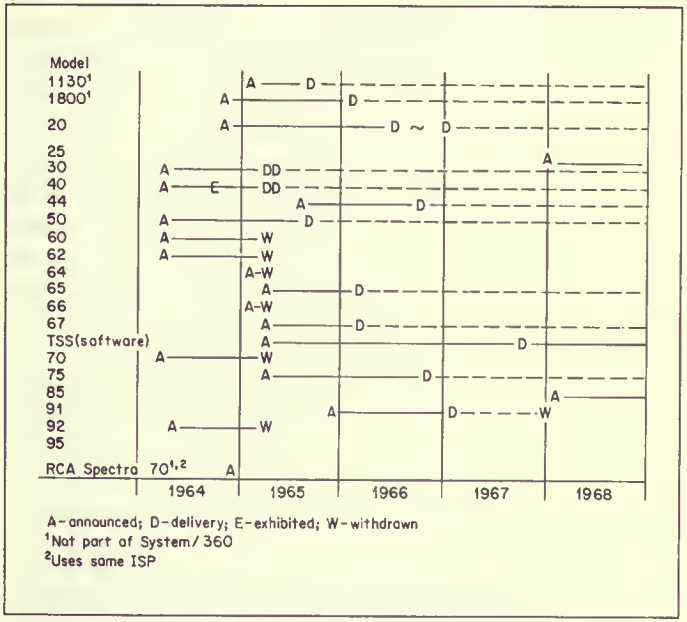


Fig. 1. IBM System/360 models introduction dates.

The architecture* of the newly announced IBM System/360 features four innovations:

1 An approach to storage which permits and exploits very large capacities, hierarchies of speeds, read-only storage for microprogram control, flexible storage protection, and simple program relocation.

2 An input/output system offering new degrees of concurrent operation, compatible channel operation, data rates approaching 5,000,000 characters/second, integrated design of hardware and software, a new low-cost, multiple-channel package sharing mainframe hardware, new provisions for device status information, and a standard channel interface between central processing unit and input/output devices.

3 A truly general-purpose machine organization offering new supervisory facilities, powerful logical processing operations, and a wide variety of data formats.

4 Strict upward and downward machine-language compatibility over a line of six models having a performance range factor of 50.

The above four featured innovations are all stated as IBM Corporation design results. It seems better to analyze them in terms of design constraints and implementation results. It appears that the design constraints, from marketing and management directions, were compatibility (item 4 above) and the use of common peripheral equipment (item 2 above). Thus we can measure the 360 design in terms of how well it meets these constraints. With some minor exceptions, all the peripheral components existed at the time of the design and had been used with other IBM computers; thus a goal was already realized. A measure of the design can also be based on a comparison with alternative designs. In the following sections we suggest that several forms of multiprocessing would yield higher performance at lower cost. A difficult and important constraint, though not mentioned above, is the necessity of program compatibility with almost all earlier IBM computers.

It should be noted that, at the outset of the IBM System/360 announcement, another company, RCA, adopted the 360 ISP as a design constraint for its own future computer development. Although some price-performance characteristics appear to be better in the RCA series, the implementation scheme is similar. The lower RCA prices do not reflect entirely implementation and technology but include RCA marketing and profit strategy. In addition, of course, there should have been lower development costs.

An interesting aspect of the design is the method used to implement the individual computer models (of the range) and their associated costs. From the standpoint of innovation, the 360 was the first computer series to cover a wide range. The more basic P's (Models 20 ~ 65) were implemented via a microprogrammed processor. This is based on a computer program within an M(read only), i.e., a Read Only Storage/ROS, to interpret the common ISP. A payoff from this implementation strategy is a solution to the "compatibility design constraint," which is the ability to provide compatibility with the customer's previous (IBM) machine, which of course was not a member of the 360 series. This is undoubtedly the most difficult constraint to meet in the P designs, and probably the most significant real innovation. From the marketing viewpoint, it provided the user with a crutch to go from a former IBM computer to the System/360. This is accomplished through "emulation," which (as defined by IBM) means the ability of one C to interpret another's programs *at a reasonable performance level.* These emulations are realized by various microprogrammed P's being designed to interpret both the 360 ISP and one or more of IBM 704, 709, 1401, 1410, 1440, 1460, 1620, 7010, 7040, 7044, 7070, 7074, 7090, 7094.

Most of the above ISP's have a different structure from the 360 ISP. For example, the 1401 (Chap. 18) series instructions and data are variable-length character strings; the 1620 has variable-length data strings; the 704 series process fixed- and floating-point data with single-address instructions; and the 7070 is a fixed-word decimal computer. Thus the 360 C's represent the first machines to be two logical processors in the same physical implementation.

The emulated speeds are often better than that of the original hardwired computer. This is not surprising, considering the change in technology; it is a very attractive feature. The 360 Mp performance is often a factor of 5 to 10 times the "emulated" computers; and the M(ROS) data rates are a factor of 25 times the Mp's. For example, the Model 65 emulating a 7090 runs faster than a hardwired 7090 (Table 1). The use of an M(ROS) for defining an ISP is questionable if we ignore the emulation constraint. Note, by way of evidence, that the hardwired models 91 and 44 have the lowest cost-to-performance ratios in the series.

There are minor deviations in the particular models, but all

* The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.

Table 1† IBM System/360 Models, IBM 1130, and IBM 1800 computer characteristics

Parameter	*1130*[a]	*1800*[a]	*20*[b]	*25*	*30*	*40*	*44*	*50*	*65|67*	*75*	*85*	*91*	
Pc (technology: (hybrid/h|μ.ro|μ.rw)); Pio (technology)	h;h	h;h	μ.ro	μ.rw	μ.ro	μ.ro	h;h	μ.ro	μ.ro;h	h;h	h,μ.ro,μ.rw,h	h;h	
M (ro|rw; t.cycle: μs/w;	...	...	?	(Mp)	1.0	0.625	...	0.5	0.2	...	0.08		
size:w;	...	...		...	4096	4096	...	2816	4000	...	2000,500		
b/w;	...	...		...	60	60	...	90	100	...	108		
technology: (ind|cap|core);	...	...		core	cap	ind	...	cap	cap	...	ro,rw		
ISP's implemented in P.microprogram	...	...	...	1401[c]	1401|1620[d]	1401|1410[e]	...	1410|7070[f]	7070|7090[g]	...	7090		
S (concurrency: (Mp;Pc))	1;1	1;1	1;1	1;1	1;1	1;1	1;1	1;1	1;2|8;5	1,2,4;1	(4,1)|(1,1)	16;1	
Mp (i.width: (by); (8, 1 parity) b/by;	2	2	1	2	1	2	4	4	8	8	16	8	
t.cycle: μs/w;	3.6	2|4	7.2	0.9	1.5	2.5	1.0	2.0	0.75	0.75	(0.96,1.04)|0.08[h]	0.75	
size: $\log_2$(by);	13~14;	13~16;	12~14;	14~15;	13~16;	14~18;	15~18	16~19;	17~(20|24);	18~20;	19~22;	20~22;	
i.avg: $\log_2$(by);	13.5	14.5	13	14.75	14.5	16	6.5	17.5	18.5	19	20.5	21	
i.rate: b/μs;	4.45	8|4	1.1	17.8	5.3	6.4	32	16	(85~170)|425	85,170,340	~512|1600	1370	
t.1-bit: μs;	0.22	0.125|0.25	0.9	0.056	0.19	0.16	0.031	0.063	0.017|0.0025	0.017	2×10^{-3}|6.3×10^{-4}	7.3×10^{-4}	
t.64-bit: μs)	14.	8|16	58.	3.6	12.1	10	2	4.0	0.75,0.375|0.147	0.75,0.375,0.18	0.125|0.04	0.047	
C(t.matrix.q: μs;	...	...	...	...	71	23.4	...	7.35	1.8	1.02	...		
t.sqrt.q: μs;	...	...	...	...	118	26.8	...	6.8	1.97	1.24	...		
t.field-scan.q: μs;	...	...	...	...	35	12.8	...	5.8	1.8	1.64	...		
t.scientific-mix.q.μs;	...	...	...	...	47	15	...	8.0	2.4	1.45	...		
t.all.q: μs;	...	...	148	110	88	25.4	5.7	8.5	2.3	1.55	*	*	
t.avg.q: μs;	42[o]	24|48[o]	120	80	60	20	4	8	1.9|2.2	1.3	0.5[k]	0.4[k]	
power/p_1(1/t.avg.q);	2.9	5.0|2.5	1.0	1.5	2.0	6.0	30	15	63|54	92,100[j]	252[j]	314[j]	range: 1~314
power/p_2(1/t.64-bit);	4.2	7.2|3.6	1.0	16.	4.8	5.8	29	14	77,155|394	77,155,310	465|1450	1230	range: 1~1450
power/p_3[Stevens, 1964][i];	...	...	...	...	2.0	7.0	...	20	42-60	100	...	...	range: 2~100
power/P_4[Conti, 1968][j]	...	...	...	...	...	...	...	...	1	1.58	3.9~4.3 [m]	5	range: 1~5
Mp utilization efficiency/(t.64-bit/t.avg.q))	...	...	0.49	0.045[n]	0.2	0.5	0.5	0.5	0.37~0.18	0.54~0.27	0.25/0.08	0.2	
Pc(cost:$/s)	0.00064	0.0019|0.0016	0.00049	0.00050	0.0013	0.0030	0.0041	0.012	0.022|0.029	0.037	0.087	0.091[l]	range: 1~186
Mp(cost.avg:$/s)	0.00049	0.0014|0.0012	0.00065	0.0027	0.0023	0.0049	0.0050	0.0084	0.023|0.032	0.031	0.080	0.069	range: 1~123
C(cost.min: $/s)	0.00096	...	0.0019	...	0.0043	0.008	0.008	0.022	0.054|	0.075	...	0.20	range: 1~105
C(cost.avg: $/s)	0.0018	0.0077	0.0045	0.0085	0.0130	0.027	0.024	0.051	0.08|	0.128	0.18	0.30	range: 1~65
Pc(cost) + Mp(cost.avg)	0.00113	0.0033|0.0028	0.00114	0.0032	0.0036	0.0079	0.0091	0.020	0.045|0.061	0.068	0.167	0.160	range: 1~160
C(cost.min: $/s)/$p_1$	0.00069	...	0.0038	...	0.0043	0.003	0.00053	0.0029	0.0017	0.0016	...	0.0013	
C(cost.avg: $/s)/$p_1$	0.0013	0.0031	0.009	0.011	0.0130	0.009	0.0016	0.0068	0.0025|	0.0028	0.00143	0.0019	
Pc(cost)/.5p_1	0.00046	0.0008|0.0013	0.00098	0.00067	0.0013	0.0010	0.00028	0.0016	0.0007|0.0011	0.0008	0.00069	0.00058	
C(cost.min)/C(cost.avg.)	0.5	...	0.42	...	0.32	0.3	0.33	0.43	0.68|	0.59	...	0.66	avg: .47
Pc(cost)/Mp(cost.avg.)	1.3	1.4|1.3	0.75	1.85	0.57	0.61	0.82	1.4	0.96|0.91	1.2	1.1	1.3	avg: 1.1
Pc(cost)/C(cost.avg.)	0.35	0.25	0.11	0.06	0.10	0.11	0.17	0.24	0.28|	0.29	0.47	0.3	avg: 0.23

† This table is presented as PMS expressions.

[a] Not IBM System/360 compatible, but made with hybrid technology.

[b] Similar, but not identical to System/360 ISP.

[c] C('IBM 1401, 1440, 1460).

[d] C('IBM 1620).

[e] C('IBM 1410, 7010).

[f] C('IBM 7070, 7074).

[g] C('IBM 709, 7040, 7044, 7090, 7094).

[h] Two M's; an M(content addressable) working with Mp.

[i] Estimated, see Chap. 44.

[j] See Conti [1968], based on running many programs.

[k] Models 85, and 91 are too difficult to predict because of instruction buffering based on Conti [1968].

[l] Cost derived from purchase cost/45.

[m] Varies depending on buffering and multiply options.

[n] Meaningless per se; Mp is used by microprogram defining System/360 ISP.

[o] 1130 and 1800 are not program-compatible. The very high penalty factor of 3 is used to compare them to System/360 ISP.

implementations belong to a common ISP subset. The Model 20 and the Model 91, the extremes of the series, deviate most from the standard 360 ISP. The range of models (Table 1) shows the comparative effects of implementation on the actual processing times. For example, the designers of the various C's were constrained by memory bandwidths. Since the core memories have about the same cycle time (0.75 ~ 2.0 microseconds), variation in bandwidth is obtained by increasing the data path width from 8 to 64 bits and by increasing the number of independent Mp's. By looking at just Mp bandwidth, for models 30 ~ 65, we obtain a range of 5.3 to 85 megabits/s, corresponding to a performance range of about 1 to 16. By doubling the number of independent memories, this factor can be increased to 32. These models correspond to a Pc performance range of 1 to 32. Although we might expect a narrower range (based on Mp speed), the range can be increased by performance suppression (at the low end). Power range can be increased by lowering the absolute performance of Model 30. This is accomplished by making performance tradeoffs to lower cost.

Logic technology

The logic of the 360 series is realized in a hybrid technology, composed partly of integrated-circuit techniques and partly of the solid-state techniques standard in second-generation machines. It is a "thick-film" technology that deposits the circuitry on a ceramic substrate. This is called Solid Logic Technology (SLT) and is used solely by IBM. This production technique allows only for the fabrication of passive circuit elements on the substrate. The semiconductor elements (diodes and transistors) are produced independently, using standard semiconductor production techniques on a wafer. The semiconductors are then cut and bonded to the substrate, and the complete SLT logic unit is encapsulated. The substrates correspond roughly to logic elements (gates, inverters, flip-flops, etc.). The SLT units are placed on larger printed-circuit boards.

Although SLT differs fundamentally from integrated-circuit technology, the overall size of the final printed-circuit boards is about the same. At the time the decision was made to develop the technology, it was unclear that integrated-circuit technology would reach mass-production state. Thus the SLT program was an intermediate design prior to integrated-circuit technology. The two approaches are about the same from the standpoint of reliability, especially when one considers the soldered printed-circuit mounting. The number of connections to the printed-circuit board are about the same. The production technology of the 360 series is outstanding, perhaps surpassed only by the 360 marketing plan.

The Instruction-set processor

The following discussion covers only the Pc. The instruction set consists of two classes, Scientific ISP and Data Processing ISP, which operate on the different data-types. These data-types correspond roughly to the IBM 7090 (Chap. 41) and IBM 1401 (Chap. 18). For the scientific ISP they are half- and single-word integers, address integers, single, double, and quadruple (Model 85) floating point, and logical words (boolean vectors); for the data-processing ISP they are address or single-word integers, multiple byte strings, and multiple digit decimal strings. These many data-types give the 360 strength in the minds of its various types of users. The many data types may be of questionable utility and constrain the ISP design by having to perform few operations, rather than having a more complete operation set for a few basic data types. The viewpoint taken here is a biased one; we feel that, unless a particular data-type adds significant processing and storage capability, it should not be fundamental to the ISP. The decimal-string integers appear to cost in storage and processing time. Their redeeming virtues are that little or no conversion is required at input or output time, and their internal representation is easily recognized by people.

Advantages of general-registers organization

The ISP uses a general-register organization. The ISP power can be compared with several similar general-register ISP structures such as those of the UNIVAC 1107, 1108; the DEC PDP-6, PDP-10; the SDS Sigma 5, Sigma 7; and the early general-registers-organized machine Pegasus (Chap. 9). Of the above machines the 360 Scientific ISP appears to be the weakest in terms of instructions and the completeness of the instruction set.

For example, in Pegasus, PDP-6, and the UNIVAC 1107 symmetry is provided in the instruction set. For any binary operation b the following are possible:

GR ← GR b Mp
GR ← GR b GR
Mp ← GR b Mp
Mp ← Mp b Mp

The 360 ISP provides only the first two. Additional instructions (or modes) would increase the instruction length.

In the System/360 the only advantage taken of general registers is to make them suitable for use as index registers, base registers, and arithmetic accumulators (operand storage). Of course, the commitment to extend the general-purposeness of these general registers would require more operations. Chapter 3 (page 61) suggests advantages for general register organizations.

The 360 has a separate set of general registers for floating-point data. This provides more processor state and temporary storage but again detracts from the general-purpose ability of the existing registers. Special commands are required to manipulate the floating-point registers independent of the other general registers. Unfortunately the floating-point instruction set is not quite complete (e.g., fixed- to floating-point conversion), and several instructions are needed to move data between the fixed and floating registers.

When multiple data-types are available, it is desirable to have the ability to convert among them unless the operations are complete in themselves. The System/360 might use more data conversion instructions, for example, between the following:

1 Fixed precision integers and floating-point data
2 Address-size integers and any other data
3 Half-word integer and other data
4 Decimal and byte string and other data (decimal string to and from byte string conversion is provided)

Some of the facilities are redundant and might be handled by better but fewer instructions. For example, decimal strings are not completely variable-length (they are variable up to 31 digits, stored in 16 bytes), and so essentially the same arithmetic results could be obtained by using fixed multiple length binary integers. This would remove the special decimal arithmetic and still give the same result. If a large amount of fixed field decimal or byte data were processed, then the binary-decimal conversion instructions would be useful.

The communication instructions between Pc and Pio are minimal. The Pc must set up Pio program data, but there are inadequate facilities in Pc for quickly forming Pio instructions (which are actually yet another data-type). There are, in effect, a large number of Pio's as each device is independent of all others. However, signaling of all Pio's is via a single interrupt channel to Pc.

The Pc state consists of 26 words of 32 bits each:

1 Program state word, including the instruction counter (2 words)
2 Sixteen general registers (16 words)
3 Four 2-word floating-point general registers (8 words)

Many instructions must be executed (taking appreciable time) to preserve the Pc state and establish a new one. A single instruction would be preferable; even better would be an instruction to exchange processor states, as in the CDC 6600 (Chap. 39).

Addressing and multiprogramming

The methods used to address data in Mp have some disadvantages. It is impossible to fetch an arbitrary word in Mp in a single instruction. The address space is limited to a direct address of only 2^{12} bytes. Any Mp access outside the range requires an offset or base address to be placed in a general register. Accesses to several large arrays may take significant time if a base address has to be loaded each time. The reason for using a small direct address is to save space in the instruction. We know of no published attempt to analyze the tradeoffs, even of instruction efficiency alone, although undoubtedly such comparisons were made within IBM.

Another difficulty of the 360 addressing is the inhomogeneity of the address space. Addressing is to the nearest byte, but the system remains organized by words; thus, many addresses are forced to be on word (and even double-word) boundaries. For example, a double-precision data-type which requires two words of storage must be stored with the first word beginning at a multiple of an 8-byte address. (However, the Model 85, which is a late entry in the series, allows arbitrary alignment of data-types with word boundaries.) When a general register is used as a base or index register, the value in the index register must correspond to the length of the data-type accessed. That is, for the ith value of a half integer, single integer, single floating, double floating (long), and quadruple floating (extended), i must be multiplied by 2, 4, 4, 8, and 16, respectively, to access the proper element.

A single instruction to load or store any string of bits in Mp (as provided in the IBM Stretch) would provide a great deal of generality. Provided the length were up to 64 bits, such an instruction might eliminate the need for the more specialized data-types.

A basic scheme for dynamic multiprogramming is nonexistent (i.e., although static multiprogramming is done, relocation

hardware is not present). Only a simple method of Mp protection is provided, using protection keys (see Chap. 43, page 597). This scheme associates a 4-bit number (key) and a 1-bit write protect with each 2 kby block, and each Pc access must have the correct number. Both protection of Mp and assignment of Mp to a particular task (greater than 2^4 tasks) are necessary in a dynamic multiprogramming environment. Although the architects of System/360 advocate its use for multiprogramming, the operating system does not enforce conventions to enable a program to be moved, once its execution is started. Indeed, the nature of the 360 addressing is based on absolute binary addresses within a program. The later experimental Model 67 does, however, have a very nice scheme for protection, relocation, and name assignment to program segments [Arden et al., 1966].

PMS structures and implementations of the computer

The PMS structures of the various models in System/360 are basically similar, except for the upper end of the series and for the Model 44 (complete compatibility can be purchased as an option). We take up the main group first and then discuss the others individually.

Models 30, 40, 50, and 65

The PMS of Models 30, 40, and 50 is the tree-structured Mp-Pc shown in Fig. 2.[1] They all use a P.microprogram, although with different ISP's. Some gross characteristics are given in Table 1. The Pc of Model 65 is also microprogrammed, but it has hardwired Pio's. A PMS diagram of Model 65 (and Model 75) is given in Fig. 3.

The C structures with M(ROS) use a single physical P.microprogram to realize the Pc, the Pio('Multiplexor Channel), and the Pio('Selector Channel). This technique of using a single shared physical P for multiple logical P's with fast changing of P.state is the same one that Pio('Multiplexor) uses. The

[1] The structure of the Mp's does not include the local M's used for access control, i.e., the storage protect key mechanism, which it is hoped the student will forget about (forever).

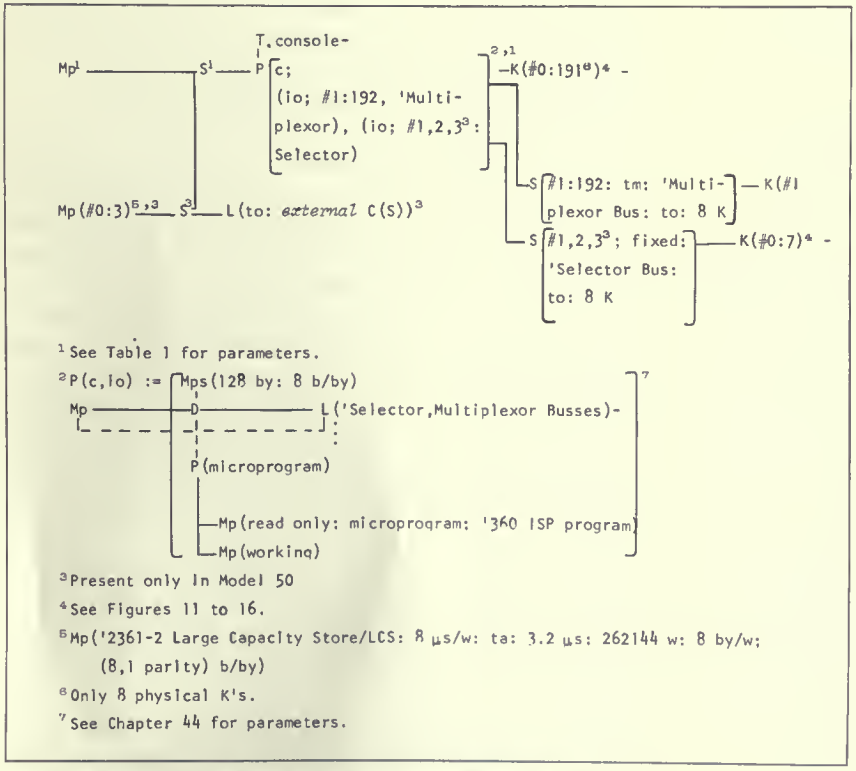


Fig. 2. IBM System/360 Models 30, 40, and 50 PMS diagram.

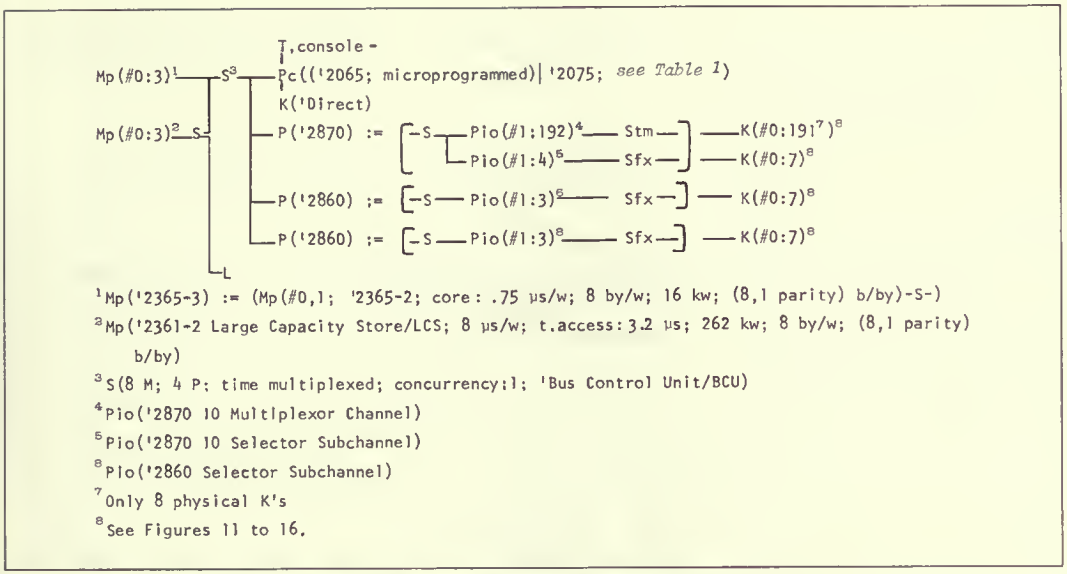


Fig. 3. PMS structure for IBM System/360 Models 65 and 75 PMS diagram.

Pio('Multiplexor) is equivalent to multiple Pio's. Within the physical P both interrupts and polling are used to switch among the P's. Polling is used to service the several P's since the main program loop of the ISP interpreter returns to a common point each time the next instruction is fetched. That is, the interpretation cycle for the 360 ISP starts by fetching the instruction, proceeds to fetch the operands, executes the instruction, and then returns results to Mp. The instruction-interpretation process takes only a few Mp references for most instructions.

A few instructions require a long (or indefinite) interpretation time, e.g., character translate, edit, etc., since the operations are on character strings. Here, the iterative program loop which operates on each character of the string must test the attached K's to detect when the Pio interpreter is to be run for data transfers. The long instructions can take several hundred microseconds and cannot be interrupted; thus the response time for an interrupt can be very poor. Figure 4 gives a simplified picture of the registers organization of a Model 50, but it is also typical of Models 30, 40, and 65.

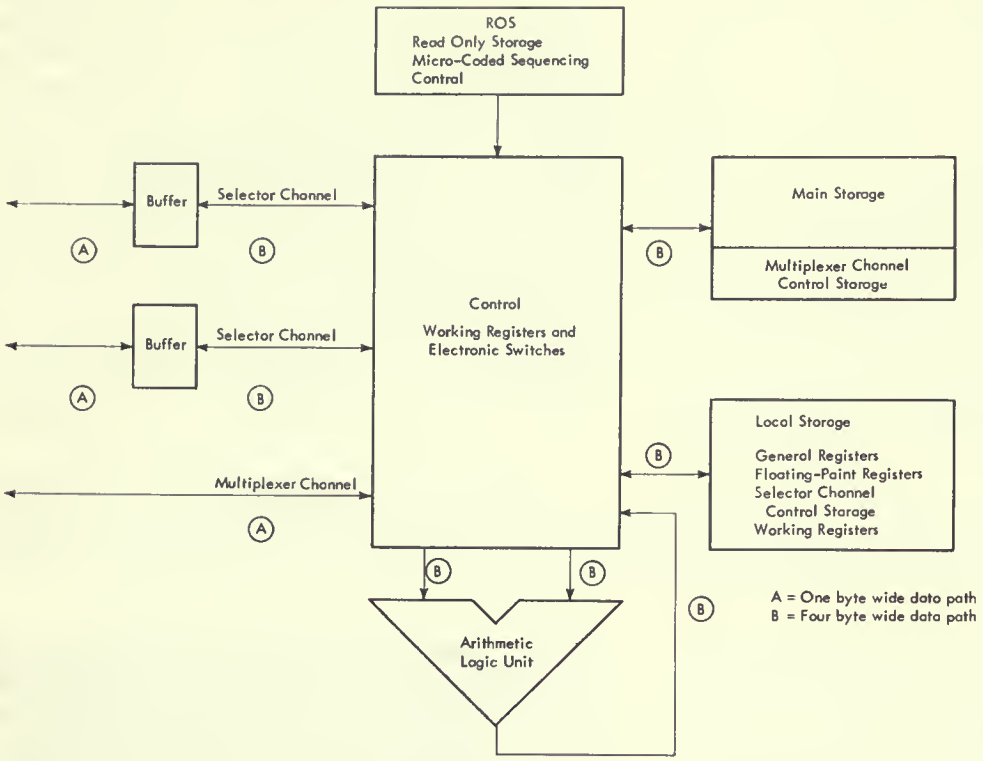
The actual System/360 ISP interpretation program in each of the models is different. In addition, each model has microprograms for interpreting other ISP's through emulation. Tucker [1967] discusses how the models were changed as the emulation constraint was added. Table 1 gives the computers which each of the models can emulate. A register structure of the C('30) and the operation for the P.microprogram ISP are given in Chap. 32, page 386. Tables 2 and 3 in Chap. 44 give the additional parameters which influence the instruction interpretation rate of the P.microprogram. The significant parameters for a P.microprogram are the M(ROS) hardware characteristics (speed, size, and information width); the number of fields in the M(ROS) instructions, which gives an indication of the number of control functions performed in parallel; the M(general register) rates and their location in the structure; the Mp data rate; and the characteristics of M(temporary) within P. The activity of transferring data from a K, via the Pio('Selector), is done concurrently with normal instruction interpretation in Models 30, 40, and 50. A program in M(ROS) sets up the data transmission with Mp, and transmission is controlled by an independent hardware control.

Model 20

This model is a subset of the System/360. It has eight 16-bit general registers. It is possible to write programs which will run on both the Model 20 and other models. Model 20 does not have Pio's, and Pc issues instructions to control the attached K's.

Model 25

The Model 25 is an interesting C. Perhaps some of the interest of the authors is caused by the mystery (to the authors) as to what its ISP is. Its ISP is no doubt described in maintenance



	Capacity/Number	Data Width	Access/Speed/Rate
General registers	16	4 bytes	0.5 microsecond R/W cycle/4 bytes
Floating-point registers	4	8 bytes	0.5 microsecond R/W cycle/4 bytes
Adder		4 bytes	0.5 microsecond
Local storage			0.5 microsecond R/W cycle/4 bytes
Read only storage			0.5 microsecond Rd cycle
Basic machine cycle			0.5 microsecond
Multiplexer channel			
Burst mode		1 byte	
Multiplex mode		1 byte	
Selector channel		4 bytes	
Data transfers			
Processor to storage		4 bytes	
Storage to storage		4 bytes	
Selector channel to processor		4 bytes	
Multiplexer channel to processor		1 byte	
Control unit to channel		1 byte	

Fig. 4. IBM System/360 Model 50 data-flow diagram and system characteristics. (*Courtesy of International Business Machines Corporation.*)

manuals. We can make the following observations based on its characteristics taken from its manual of Functional Characteristics. These appear in Table 1. The observations are:

1 It has a very high-performance Mp, namely, Mp(core; .9 μs/w; 16|24|32|48 kby; 2 by/w); the Mp power is almost that of a Model 50.

2 There is a relatively straightforward Pc which is microprogrammed. The Pc uses Mp for its memory. The System/360 ISP is defined in conventional M(read,write). Of the Mp(48 kby) 16 kby is reserved for a microprogram.

3 Its performance is between that of Models 20 and 30, performing a 360 ISP instruction in about 80 μs.

4 The penalty paid (slowdown factor) to interpret the 360 ISP is therefore $80/1.8 \simeq 45$.

5 A small 180-nanosecond local store is used for operands.

6 The Pc cost appears to be about the lowest in the series.

We should ask ourselves:

1 Why do we want an intermediate-level P.microprogram with its own M.read-only, as in the other processors? These P's just seem to waste power.

2 Why should we bother to implement an intermediate-level 360 ISP? We know the final user will write programs in a much higher level language. Thus two levels of interpretation are required instead of one. It is assumed that to program a given task will take, say, x μs if using the 360 ISP. We assume the same task programmed directly in the Pc could take as short a time as x/45 μs if the Pc were used directly.

We assume that if the P.microprogram, which is used to define the System/360 ISP, were used to interpret a FORTRAN ISP, the speed for a Model 25 FORTRAN ISP might easily approach that of the Model 50.

Model 44

Model 44 does not use M(ROS), but its Pc and Pio are hardwired (Models 75 and 91 are also hardwired). The PMS structure of the Model 44 is given in Fig. 5. Model 44 (and 91) stand out as having better performance per unit of cost than their nearest neighbors, which are implemented with M(ROS), as can be seen from Table 1. It must be noted that Models 44 and 91 are not strictly compatible with the 360 ISP since they do not process variable-string and variable-decimal-data formats, although Model 44 options can make it completely compatible. (Subroutines will probably perform satisfactorily for most applications.)

The PMS structure of the Model 44 (Fig. 5) is a tree. The C('44) structure indicates 2-Pio('High Speed Multiplexor Channels/HSMPX) which are between a P('Selector) and P('Multiplexor) in power, since a single physical P('HSMPX) with four subchannels can behave as four independent Pio's. The organization of the Model 44 Pc registers is given in Fig. 6, which reveals a straightforward implementation. The heavy lines in Fig. 6 indicated an ORing of register outputs to form a single data bus (usually 16 or 32 bits wide). The 16-bit crossover function box allows the right and left halves (16 bits) of the input to be exchanged when output. Almost all the units are registers (except the adders, parity generators, and ORers). The A, Ax, B, and Bx registers are used as the M.working for performing instructions, where the x indicates an extension register used in the 64-bit floating-point operations. The C register

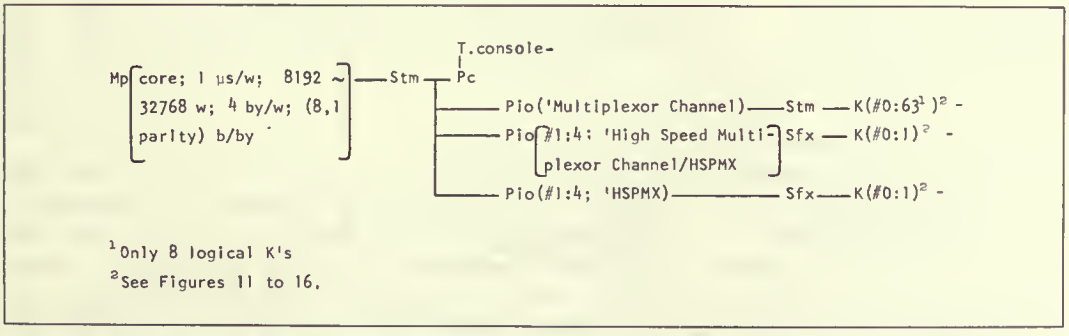


Fig. 5. IBM System/360 Model 44 PMS diagram.

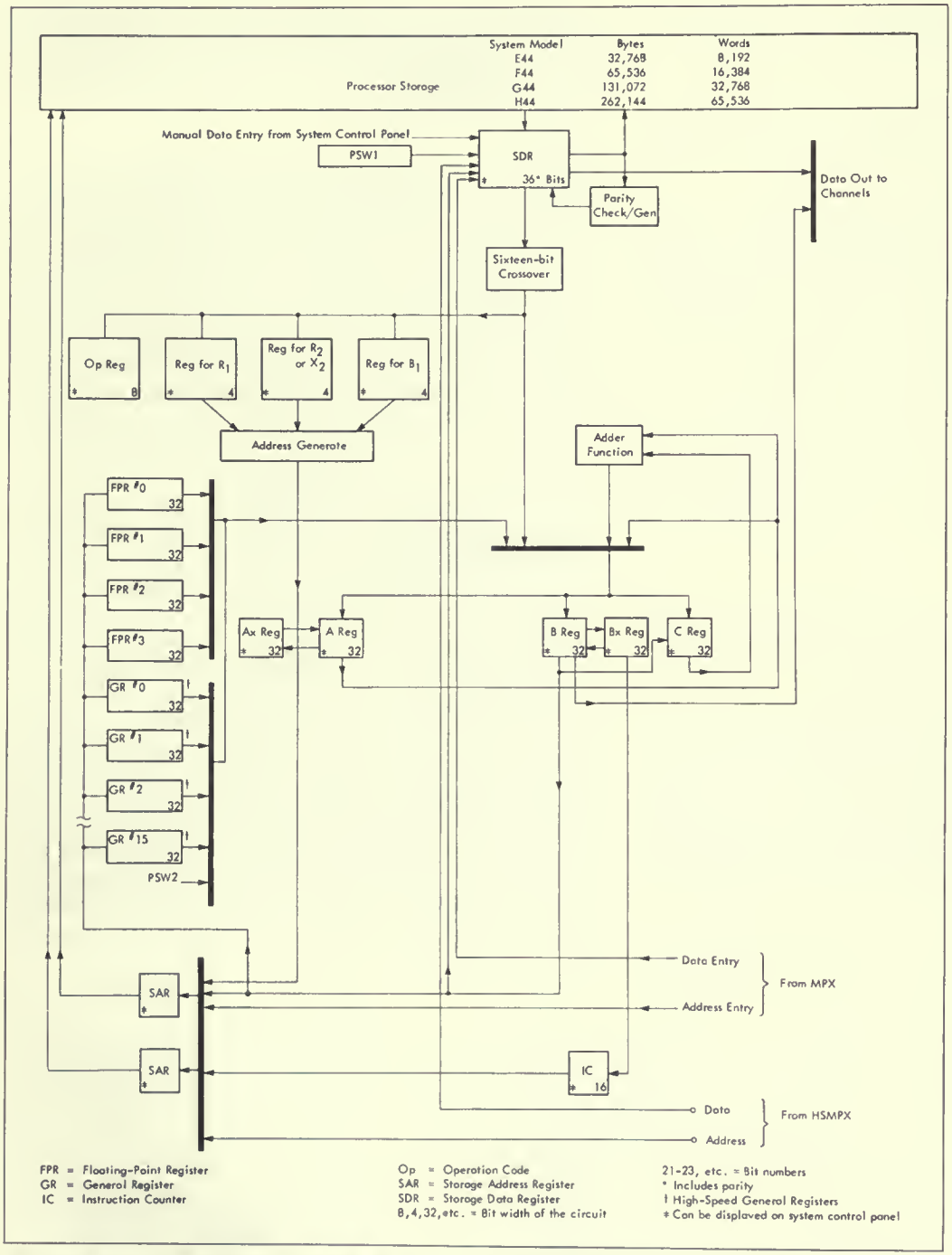


Fig. 6. IBM System/360 data flow in Model 44 CPU. (*Courtesy of International Business Machines Corporation.*)

is a second operand register used for arithmetic and logical operations.

Model 75

The PMS structure of Model 75 is given in Fig. 3. Models 65, 67, 75, and 91 all use the same basic Mp('2365; core). The S(n Mp; mP), which switches between the n Mp modules and the m Pc and Pio's, varies with model, however. C('65) and C('75) use a simple time-multiplexed S in Pc, called the S('Bus Control Unit/BCU). This S makes decisions about which P is to use which Mp, rather than having each Mp arbitrate the P requesting service locally. When the memories are all about the same speed, such an S is all right; however, it has severe limitations when slow speed (8 microseconds for the large core store) and high-speed memories (0.75 microsecond) are intermixed. The principal difference between Models 65 and 75 is that C('75) is hardwired and, depending on the size of the configuration, may have lower cost/performance.

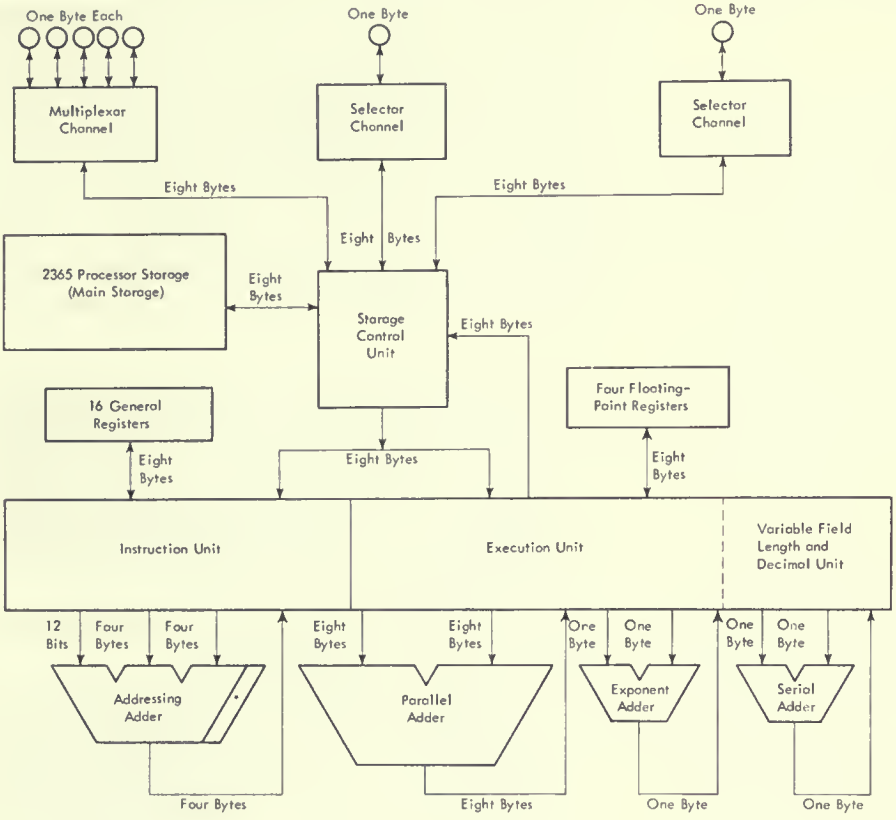
The simplified functional unit diagram of C('75) (Fig. 7) is more abstract than the register interconnection diagram of a C('44) (Fig. 6). From this description (Fig. 7) of the logic design, one is able to conjecture what is necessarily within the instruction, execution, variable field length, and decimal functional units. The diagram is presented at a nonuniform level at both the PMS and register-transfer levels. There is somewhat more detail than in the PMS structure (Fig. 3). The Model 75 is possibly the first System/360 to require an intermediate-level diagram between a PMS structure and a register-transfer diagram. The instruction unit contains the instruction location counter (part of the ISP) and is responsible for obtaining the next instruction and the operands. Since there can be overlap in the instruction fetching process, this unit is responsible for holding a number of instructions and stores up to 128 bits (2 double words) of instructions at a time. The execution unit and the variable field and decimal units carry out operations on data. The execution unit processes floating-point and fixed-point data.

Model 67

The Model 67 was introduced in April, 1965, for the purpose of time sharing. The entry was prompted by M.I.T.'s project MULTICS. M.I.T. had ordered a GE 645 for experimental research in time sharing. IBM formed a group for the development of a time-shared computer and responded with the Model 67. The Model 67 is essentially a Pc('65) with adequate S's for multiprocessing and a K between Mp and Pc for multiprogramming and memory mapping. Because of software uncertainties, the Model 67 ran as a Model 65 in most installations (in 1968). The University of Michigan and M.I.T.'s Lincoln Laboratory, the first two customers having considered the MULTICS proposal, were instrumental in outlining the specifications [Arden, et al 1966]. Several 67's have been delivered, and the software continues to evolve and be scheduled for completion (see Fig. 1). Questions of costs per console must wait until the system is stable enough to test and evaluate, although in April, 1969 IBM considered the system attractive (operational) enough to market. The most significant outcome of the experiment to date is:

1. The hardware seems capable of supporting a straightforward time-sharing system [Corbato et al., 1962]. Had IBM first developed a simple system based on proved concepts, they would be capable of undertaking research into more complex systems like the version to which they originally committed themselves. (Vendors should have some basis of actual operating experience before committing a product to market.)
2. The problems of building really large-scale software systems are not fully understood yet.
3. The idea of a virtual memory with a large address space (2^{32}w) is excellent. Many storage allocation problems are simplified by this concept. Unfortunately, the system software builders seem well on their way to filling such a memory. Thus the new freedom allows relaxation in this level of programming.
4. There is a problem of getting users into Mp.core so that Pc can be kept busy. Thus a swapping system is often found waiting for Ms.drum or Ms.disk information. Work at Carnegie-Mellon University using a Mp('LCS; core; .5 ~ 1 mw; 8 by/w; 8 μs/w) seems to indicate that a large number of users can have adequate response from the Model 67 if the users reside in core and are not subjected to swapping [Lauer, 1967; Fikes et al., 1968].

The above items relate to the software. The hardware (Fig. 8) is interesting from several aspects. First, there are adequate facilities for memory mapping and program segmentation. This general scheme is outlined in Fig. 9. In the Model 67 a user's segment and page maps are in Mp, and these maps point to physical Mp blocks of the program. Each time a reference is made, the map is checked for the actual reference. In order to avoid the accesses to Mp for each Mp reference, a K, with an M(content address), is located between Pc and Mp to trans-



	Data Width	Access/Speed/Rate	Comment
2365 Processor Storage	8 bytes	.75 microsecond storage cycle	All models
2361 Core Storage	8 bytes	8 microsecond storage cycle	All models
General registers	1 word	200 nanoseconds	16 General registers
Floating-point registers	2 words	200 nanoseconds word	4 Floating-point registers
Addressing adder	3 bytes	200 nanoseconds	
Parallel adder	8 bytes	200 nanoseconds	
Exponent adder	1 byte	200 nanoseconds	
Serial adder	1 byte	200 nanoseconds	
Basic machine cycle		200 nanoseconds	
2860 selector channel	1 byte	1.3 million bytes per second	8 bytes to storage
2870 Multiplexor channel	1 byte	110 kb to 450 kb	8 bytes to storage
Burst mode	1 byte	50–110 kb	
Multiplex mode	1 byte	50–110 kb	
Selector subchannel	1 byte	100 kb, each	

Fig. 7. IBM System/360 Model 75 data-flow diagram and system statistics. (*Courtesy of International Business Machines Corporation.*)

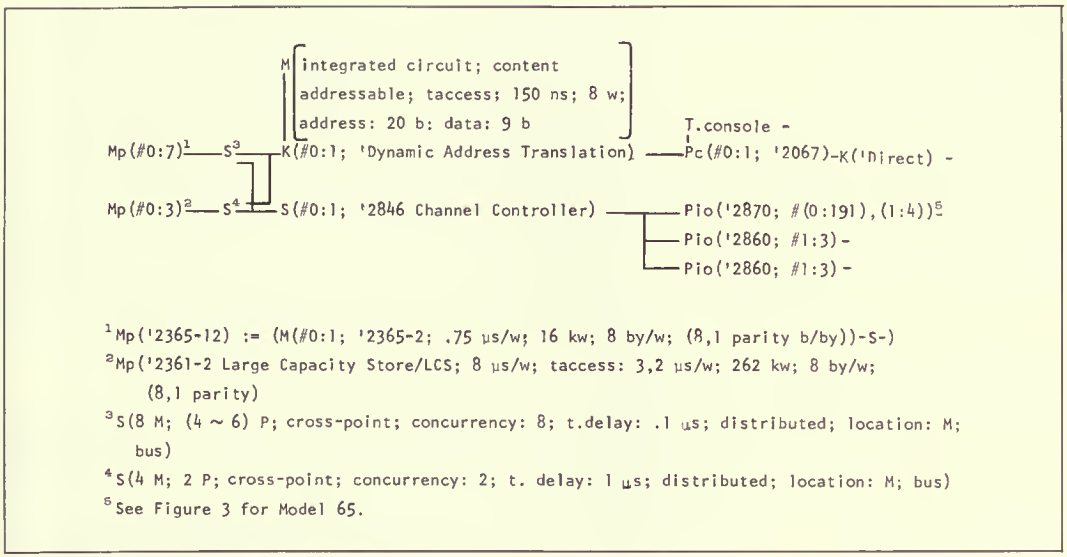


Fig. 8. IBM System/360 Model 67 PMS diagram.

form a 24- or 32-bit virtual address in Pc into an actual 19- to 22-bit physical address in Mp. This K is not shown in Fig. 9 because it is not logically necessary. The scheme suggested in Fig. 9 uses control bits in the map to determine legal Mp accesses. In the Model 67 the storage key mechanism holds whether a given page can be accessed by a given numbered user (instead of associating the control with the mapping as shown in Fig. 9).

Second, the Model 67 is the first acknowledgment by IBM of multiprocessor computers, since it provides adequate switching to allow multiple Pc's. The C('65) multiprocessing configuration has been introduced based on Model 67 structure. Multiprocessors are necessary for reliability, not solely for performance reasons.

The PMS structure of C('67) in Fig. 8 does not have to use the S('Bus Control Unit/BCU),[1] as in the C('65). The C('67) can have an S in each Mp, so that four P's can communicate with an Mp, as shown in Fig. 8. Each Mp makes the decision about the P request to be honored next. Thus the problem of having an "all knowing" S('BCU) is solved by allowing each Mp to do local scheduling, rather than having a dialogue with another component (with time delays). The S('BCU) in a duplex C('67) is still present, but with less power, in the form of the S('2846 Channel Controller). It is used to arbitrate the Pio accesses to Mp.

Without multiprocessing, the Pc seems very badly mismatched with respect to Mp. Consider, for instance, the data rates on the C('67). From Fig. 8 its maximum possible Mp data rates are:

For 1 Mp('2365-12):

$$\frac{2 \times 64 \text{ bits}}{0.75\ \mu s} = 171 \text{ megabits/sec}$$

and for 1 Mp('2361 Large Core Store):

$$\frac{64 \text{ bits}}{8\ \mu s} = 8 \text{ megabits/sec}$$

Thus the total data rate is

$$171 \times 8 + 8 \times 4 = 1{,}368 + 32 \text{ megabits/sec} = \sim 1{,}400 \text{ megabits/sec}$$

The processing rate is approximately

$$\frac{64 \text{ bits}}{2.2\ \mu s} = 29 \text{ megabits/sec}$$

An Ms.drum rate is approximately

$$\frac{8b \times 1.2}{\mu s} = 10 \text{ megabits/sec}$$

[1]A system with only one port at Mp, controlled by BCU, is called a simplex. A system with multiport Mp is called a duplex.

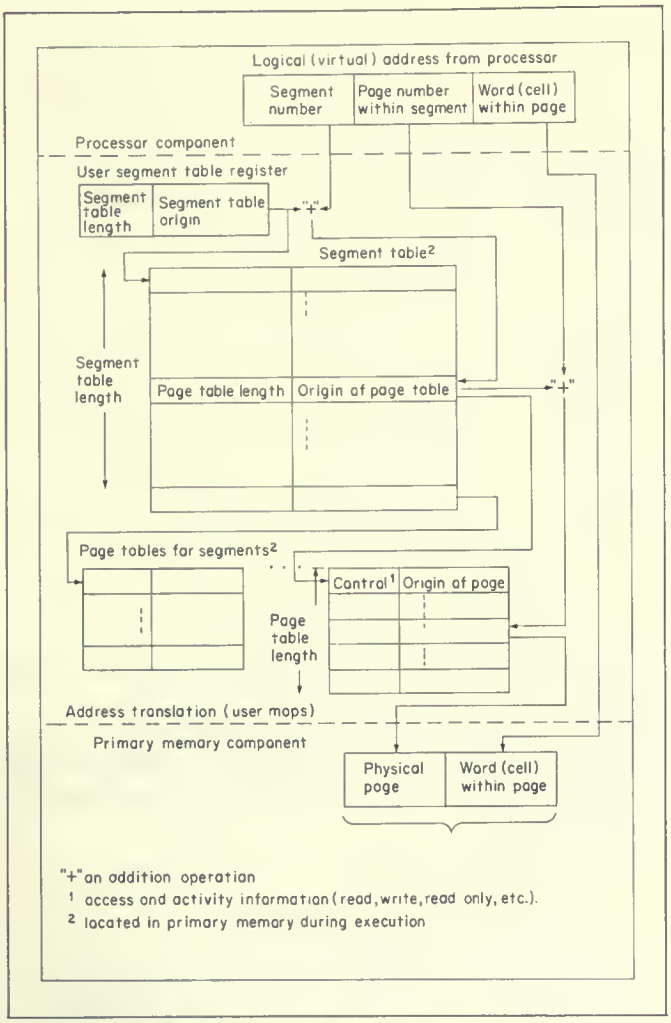


Fig. 9. Memory allocation using pages and segments.

Thus, for the several P's, an effective Mp request rate of 100 megabits/sec might be needed. The data-flow mismatch (between Mp and the P's) occurs because of the P's, the S (the L's connecting P and Mp), the lack of P's, and the fact that t.access = $\sim \frac{1}{2}$ t.cycle.

The Pio('2870), used in Model 65 and above, is described at two structural levels in Fig. 3. The Pio includes a large M.working to store the state of each of the logical Pio's. This Pio state includes the instruction location counter, the control state bits (active, running, interpreting an instruction, processing data, etc.), and buffering (one 8-byte word). By having an M.buffer, the demands on Mp from the Pio's are reduced by a factor of 8. Although the expected data rate from many K's does not require the extra M, there are possible times when the uncertainty of the access times for Mp might cause data loss. Since the M.working is necessary to store the Pio state, the additional space for buffering is not expensive. An alternative design might use Mp for this buffering.

The four Pio('2860 Selector Channel)'s are implemented as independent Pio's, using conventional hardwired logic and buffering. However, they are packaged as one unit.

Model 85

The Model 85 was announced in February, 1968, with the goal of being the highest-performance Model 360 in production. The performance is $\sim(3 \sim 5)$ times the Model 65 and in some cases outperforms a Model 91 [Conti et al., 1968].

The PMS diagram of the Model 85 is shown in Fig. 10. The Pio, T, Ms structure is identical to that of Models 65 and 75 (Fig. 3). The two interesting aspects of the structure in Fig. 10 are the M(content addressable; 'Buffer Storage; 16|32 page; 1024 by/page) and the Pc. The pages are filled in groups of 64 bytes, as references to a particular physical block in Mp.core are made. Conti [1968] gives running times for various programs as a function of buffer memory size. Multiprogramming may degrade the performance more than any other case. This process, which has been referred to as "look aside," or a "slave memory," was suggested by Wilkes [1965]. It is completely analogous to the Model 67 M(content_addressable; 8 w) which is used to hold the segment-page map for a multiprogrammed time-sharing system. It is also analogous to a one-level storage system (Atlas; see Chap. 23) which is formed from two physical M's whose performance differs significantly. Here, the effect is to try to approximate a computer with a large Mp(80 ns/w) by using a large Mp(1 μs/w) and a small Mp(80 ns/w). The CDC 7600 (page 475) has a similar structure, but the Mp-Ms migration is under programmed control.

The P.microprogram used for controlling the Pc(K('Execution Unit)) allows for great flexibility in the definition of ISP's. An Mp(500 w) is available for the user; this may be loaded by a program, and it specifies an ISP. One standard option is to emulate the 704-7094 series.

The Model 85 removes the restriction of aligning words at particular boundaries. Thus any logical word, independent of its length, can be located at any physical location addressed in bytes.

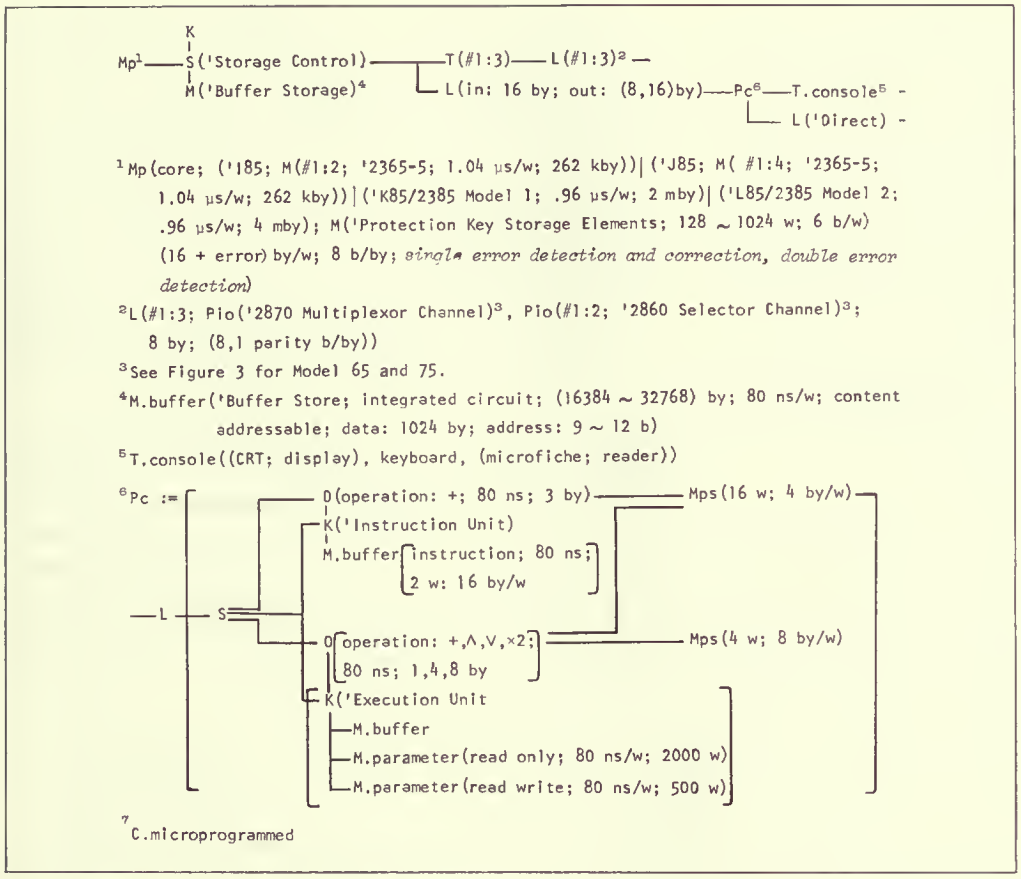


Fig. 10. IBM System/360 Model 85 PMS diagram.

The Pc's data operation performance is impressive. A fixed-point multiply is done in 0.4 μs, and a floating-point multiply takes 0.56 μs (not including accesses).

The data-type, extended floating-point number, is used in Model 85. Thus a 24-, 56-, or 112-bit fraction part can be used.

Model 91

This model has a very low cost/performance ratio (see Table 1). Only about 20 Model 91's were produced before it was withdrawn from the market. It has the highest performance of the series. The Mp is 0.75 μs, but 16 are overlapped to provide a theoretically maximum bandwidth of 16 × 64/0.75 = 1,370 megabits/s. About 2.5 mega-instructions/s are executed; thus, a total of 160 megabits/s of Mp are absorbed by Pc.

There are other interesting models in the '90 series; the Model 92 was a paper machine,[1] and the Model 95 was unannounced but produced, a version of the Model 91 with an Mp(integrated circuit; 60 ns/w; 8 by/w). The Model 91 is not covered in any detail here because of space limitations. It is similar to other very large computers in that many techniques are employed to obtain parallelism. The January, 1967, *IBM Journal of Research*[1] is devoted to design issues of the Model 91.

Models 1130 and 1800

These computers are presented as reference points and have nothing to do with the C('360). They are implemented outside the System/360 framework but use its technology, and so cost comparisons are still somewhat meaningful. These computers

[1]See bibliography at the end of this chapter.

are straightforward, and for a given task which does not use floating-point arithmetic, they should perform as well as any System/360 model. The arguments we use for the intermediate Pc for the Model 25 apply equally well here, too. Namely, why have such a complex ISP when simple ones will do just as well?

The programmed floating-point arithmetic times for a 4-μs 1800 and the "hardwired" (microprogrammed) System/360 Model 30 are compared in Table 2. We would expect the 2-μs 1800 to be better by a factor of 2. Note that the times are about the same for Model 30 and the slower 1800. The cost/performance is especially low with the 1130 (Table 1). In Chap. 33 we discuss the 1800. It is interesting to speculate why the 1130 and 1800 cannot be implemented within the System/360 framework. Are they "loss leaders"? Are they in response to more sophisticated, performance-oriented users?

The PMS structure of the controls, terminals, secondary memories, and special processors

There are many common components which attach to the C's (Figs. 11 to 17). Most of the components which attach to a Pio are not especially interesting, but they give an idea of the behavior and parameters. For example, the expression T('1403 Model 3; line; printer; 1100 line/min; 132 char/line; 8 bits/character; 64 ~ 240 character set) pretty well describes a typical line printer. From the above description one can deduce the data rate of a T(line printer). It is 132 char/line × 1100 line/min × $\frac{1}{60}$ min/s × 8 b/char = 19.4 kb/s.

The channel-to-channel adapter control. The most interesting group of components (outside the C structures) are the special components shown in Fig. 11. The K('Channel to Channel Adapter) allows two P's, either on the same or a different C, to communicate with one another. This K is used in the construction of a dual C system or the N('Attached Support Processor/ASP) in Chap. 40, page 506. A C('40|'50) is attached to a C('65|'75). The C('40|50) is used as a Cio with file processing capabilities. The K has M.buffer. Data can flow in only one direction at a time.

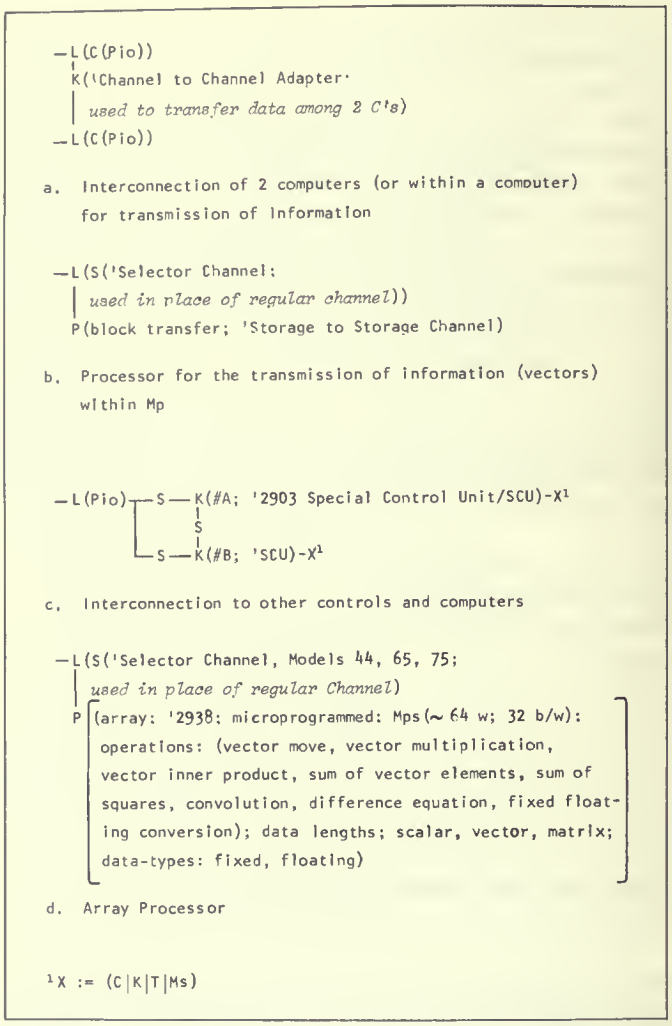


Fig. 11. IBM System/360 special P's and K's PMS diagrams.

Table 2 IBM 1800 (4 μs) and IBM System/360 Model 30 floating-point arithmetic timing

	Operation times (μs)	
Operation	*1800 (4 μs)*	*System/360 Model 30*
+{sf}; +{df}	460; 440	75; 115
×{sf}; {df}	560; 790	320; 1060
÷{sf}	766	600
$\sqrt{}${f}	4500	2965
sin {f}	3000	3876
exponential {f}	2000	4173

The special control unit. The K('2903 Special Control Unit/SCU) consists of two independent K's which are physically packaged together and allow users to interface with the Pio's. Although it has not been discussed, the actual interconnection with a Pio, via the S(Pio; K), is via a physical I/O bus which is arranged

in a bus (or chained) fashion. Such a single interface to handle a wide range of needs (high and low response and data rates) via a single set of electrical conductors requires a great deal of control information to be passed along the link. Therefore a K must have a great deal of knowledge of the dialogue in order to communicate. The hardware to attach to the I/O bus at a K is costly and must be designed carefully. The K('SCU) provides a rather simplified interface to the Pio. All I/O bus synchronization control, communication protocol control, buffering, and electrical isolation are within K('SCU). The K('SCU) is fairly flexible, in that devices connected to it can communicate with one another without Pio (see Fig. 11).

Storage-to-storage-channel processor. The P('Storage to Storage Channel) is a special processor which performs the sole function of transferring data blocks (a word vector) between one location in Mp to another in Mp. It qualifies as a P, since it takes an instruction from Mp containing the location and length, and once the instruction is executed, another is fetched and executed (if it exists). Thus the component has a well-defined interpretation cycle and set of operations. This P is useful in a multiprogrammed environment requiring programs to be moved.

The 2938 array processor. The P.array('2938) is an extremely interesting special P (Fig. 11). It can be connected to Models 44, 65, or 75. It has a limited instruction repertoire, but the instructions it interprets are more complex than those in the ISP of the Pc. The instructions are algorithms for operating on an array (a vector or a matrix). These instructions include:

1 Vector move, similar to the P('Storage to Storage) described above, with conversion either way between fixed and floating point
2 An element-by-element vector sum
3 An element-by-element vector multiplication
4 A row-by-column vector inner product
5 A convolution multiply
6 The solution to a step in a difference equation

The P.array is microprogrammed, using an M(ROS), which

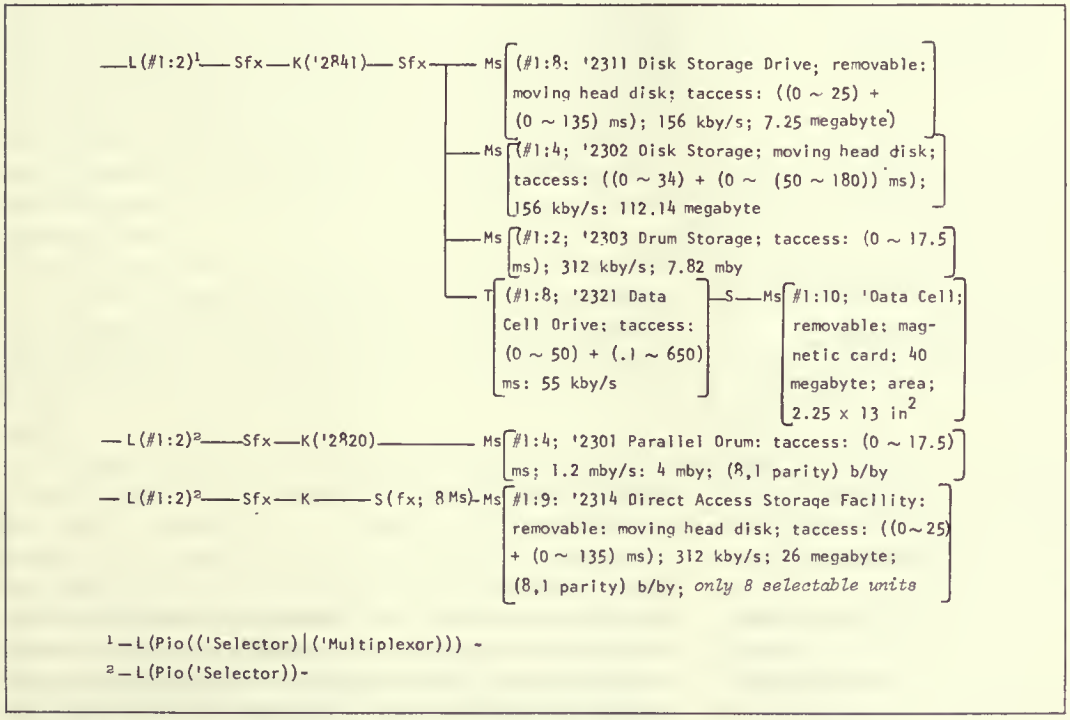


Fig. 12. IBM System/360 Ms(drum, disk, data cell) PMS diagrams.

Fig. 13. IBM System/360 Ms(magnetic tape) PMS diagrams.

makes it possible to construct complex algorithms in a flexible manner. The hardware logic is capable of doing a combined floating-point multiplication and addition in 200 nanoseconds. The impressive results this P achieves in the interpretation of the algorithms are principally because the time to access the algorithm has gone to zero. A measure we might apply to a P is the ratio of the time it spends fetching the algorithm's data to the total time it spends executing the algorithm. In a conventional computer Pc we suggest that a ratio of nearly ½ is very good. Two fetches are usually required—one for data, one

for the instruction. This P has a ratio near one, as it is always accessing data (and rarely instructions).

Secondary-memory structure. Figures 12 and 13 present the Ms PMS structures. All the K's have an optional S, which can be placed between the K and the S(P;K) to allow two Pio's to access a common K (from either of two C's or two Pio's of the same C). The K('2841 Storage Control) is interesting only in being able to control a series of quite disparate devices, on a one-at-a-time basis.

Figure 13 presents all the M(s; magnetic tape)'s. The switch is interesting as it can be used for up to four K's to access simultaneously any of 16 M.tapes. (The vast array of very similar devices is due undoubtedly to marketing rather than production or engineering reasons.) It should be noted that there are two distinct M.tapes: conventional magnetic tape and Hypertape. Hypertape is explicitly addressed and has built-in error-correction coding.

Terminal structure. Figure 14 shows the T(cathode ray tube; display) and T(audio; output). There are terminals for writing and reading from photographic film (35 mm). The two approaches used for audio (vocal) output are noteworthy. One uses an M.drum to record a fixed vocabulary of words; the other uses an encoding mechanism to allow digital information stored in Mp to be transferred via the K('7772 Audio Response) to transforming a coded voice back to an audio output form. The S at the output of the T(audio) provides for audio signals to be switched on a word-by-word basis to any of several output telephone lines.

The structure of the vast array of printing devices that can attach to the C('360) is shown in Fig. 15. Some of the devices are interesting, such as the one that reads pencil-marked or typewritten paper. The main parameters of significance to PMS are the rate the device reads paper together with the kind of paper.

The T and K's which connect to external processes are given in Fig. 16. The K('1827) is used to connect with analog processes and is actually part of the IBM 1800 computer system (Chap. 33). The other K's are important, though not especially interesting, since they provide the K to T(Teletypes), K(telephone lines), and T(typewriters). The K('2701) and K('2702) are built to transform unsynchronized parallel data from the C into the synchronized serial form required by the telephone line. The K('2701) controls a small number of lines of high data rates; the K('2702) controls a large number of lines at low data

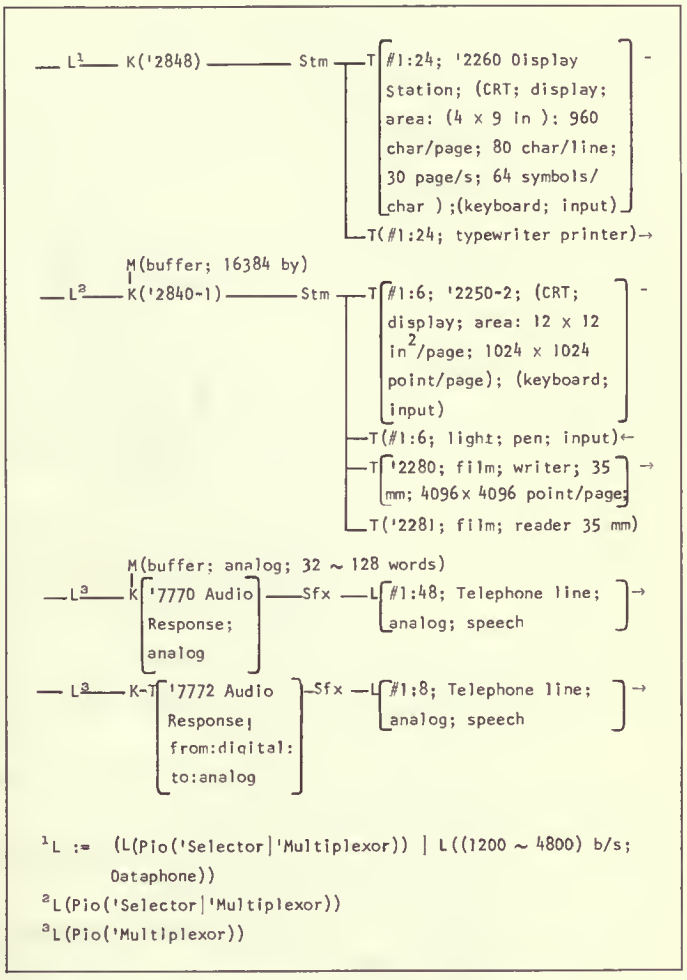


Fig. 14. IBM System/360 T(audio, display) PMS diagrams.

rates. The K('2702) is actually an array of up to 31 K's that are time-multiplexed, using an M.core to hold the state of each K.

Peripheral switching. For performance, communications, and reliability reasons it is necessary to provide access to K's, M's, or T's from several C's or Pio's. A sample structure of a possible configuration, using the above components, is given in Fig. 17. The PMS diagram also shows the physical structure of S(from:Pc; to:K).

Performance and costs

The System/360 series is perhaps the only group of computers for which a valid comparison of performance and cost can be

Fig. 15. IBM System/360 T(printer, reader, punch) PMS diagrams.

made. The models use essentially the same technology, implement the same ISP, and are probably constrained by a common corporate profit goal. Even here, as we noted earlier, comparisons are difficult to make.

In Table 3 we present the costs for various PMS component primitives. From this table, costs (relative to other components) can be obtained. These costs are expressed as dollars per second ($/s) to rent the equipment. They have been derived from the IBM monthly rental prices. The computer prices are based on estimates of minimum, average, and maximum configurations in the *Adams Computer Characteristics Quarterly* [Adams Associates]. The conversion factors are

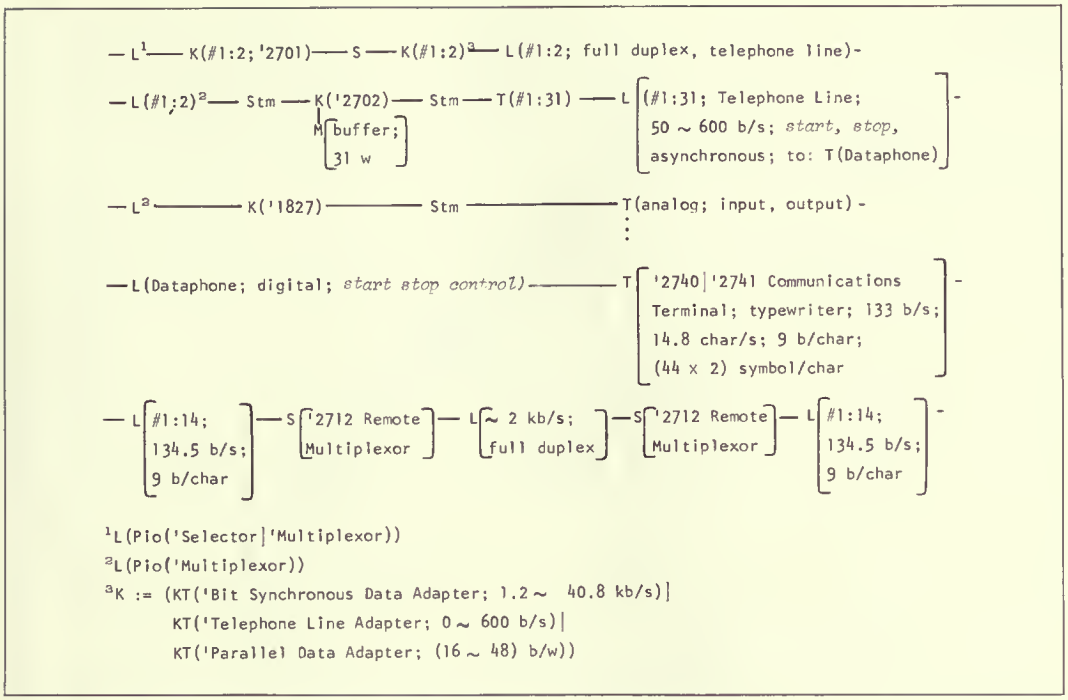


Fig. 16. IBM System/360 T(telephone line, analog, typewriter) PMS diagrams.

$$\$/s = 1/[(173.3 \text{ hour/month}) \times 3{,}600 \text{ s/hour}]$$
$$= 1.6 \times 10^{-6} \$/\text{month}$$
$$\$/\text{month} = 0.625 \times 10^{6} \$/s$$

The cost to buy, in dollars, is approximately

$$\$ = 45 \times (\$/\text{month})$$
$$\$ = 45 \times 0.625 \times 10^{6} (\$/s) = 2.82 \times 10^{7} \times (\$/s)$$

Table 1 is written as a single, large PMS expression, thus, the attributes are:

Pc(cost: ($/s|$)) : = c.Pc : = cost of Pc alone

Mp(cost.avg) : = c.Mp.avg : = cost of average-size Mp for a model

C(cost.min:) : = c.C.min : = cost of minimum-size computer configuration

C(cost.avg:) : = c.C.avg : = cost of average-size computer configuration

Primary memory

The graph of Fig. 18 gives the Mp costs, c, (in $/s) versus memory size (information/i). The line i = $1.43 \times 10^{7} \times c$ is

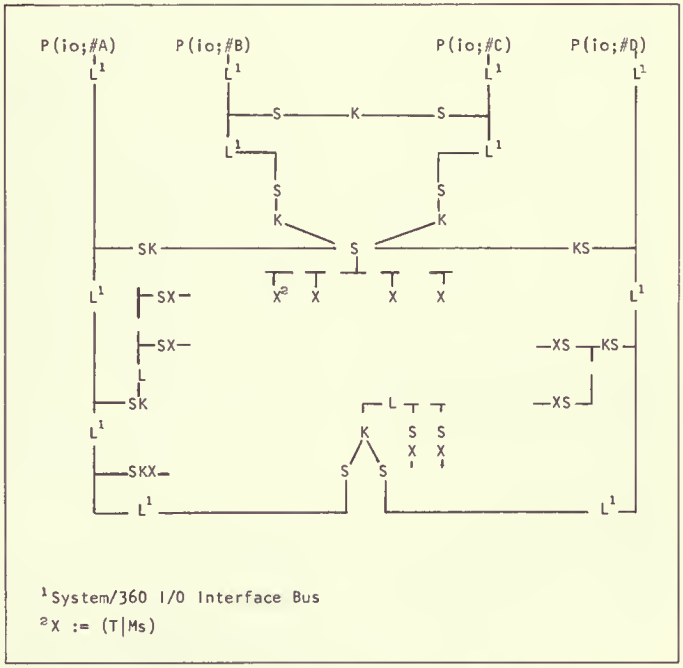
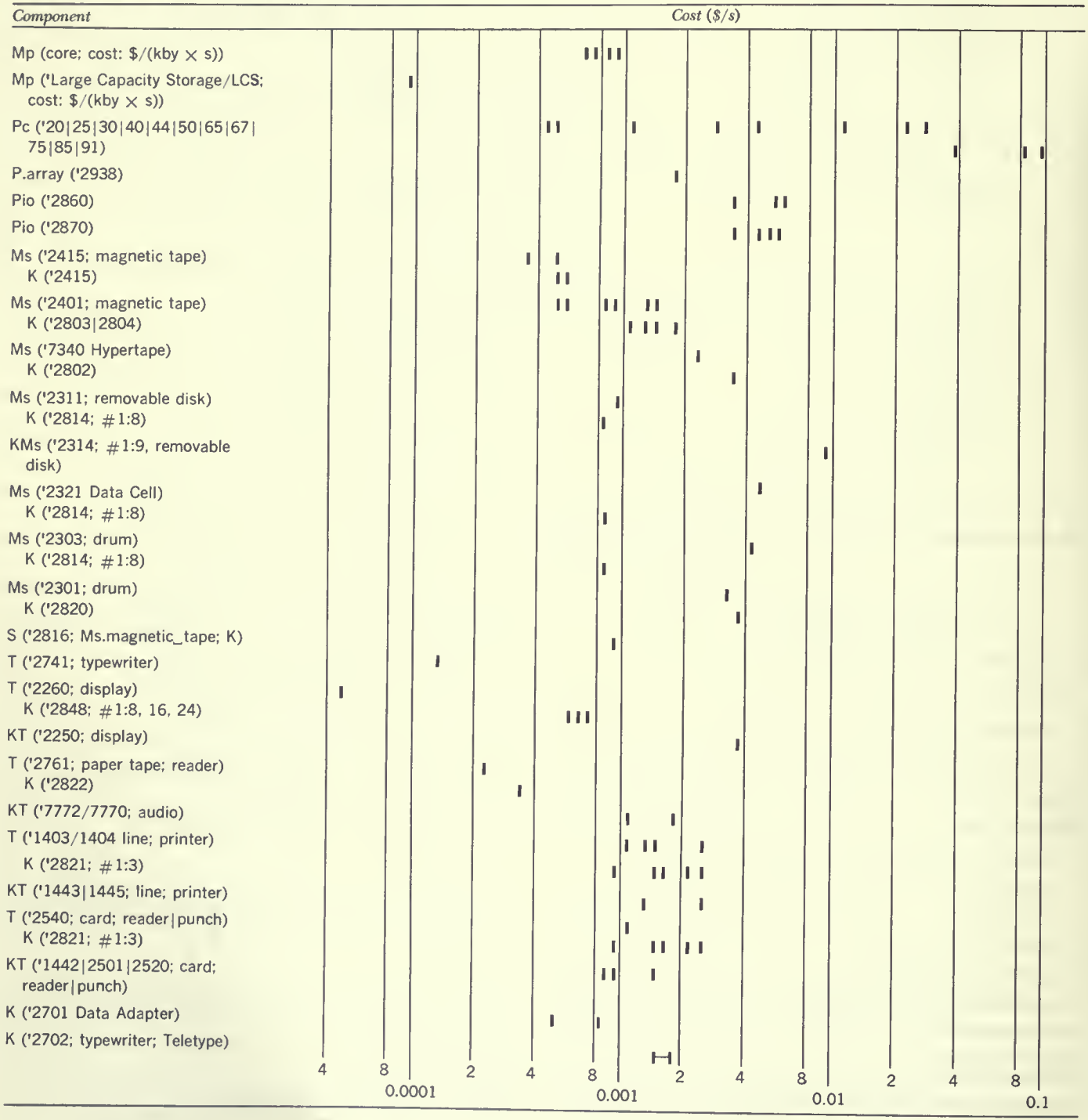


Fig. 17. IBM System/360 peripheral-switching PMS diagram.

Table 3 IBM System/360 component costs



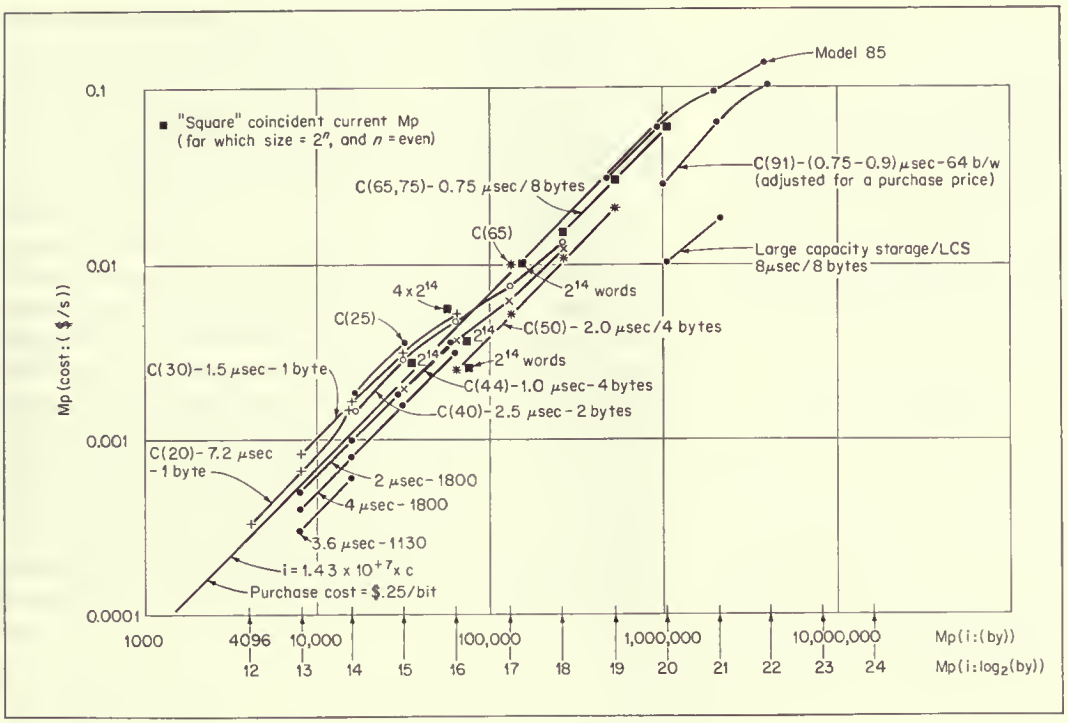


Fig. 18. Graph of IBM System/360 core-memory cost versus core-memory size.

plotted in terms of \$/(by/s) and allows us to compute the purchase cost of a bit. The purchase cost of most Mp.core is \$0.25/bit, according to the line. The 8-μs Large Capacity Storage/LCS cost is \$0.032/bit. There appear to be slight cost savings for large Mp's and a significant saving for lower performance in the case of LCS, a factor of 8. A reasonable formula for Mp cost is: $c = (7 \times 10^6 \times i)/[t.cycle: (\mu s)]$. This formula would account for Model 50 Mp and LCS costs, but not Model 25 and 30 Mp costs. We really need an $i^{1/2}$ term in the formula to make a good fit (and also a constant). The value $i^{1/2}$ should be present, if purchase prices are related to manufacturing costs, because coincident current selection cost is inherently proportional to $i^{1/2}$.

An odd pricing point is the Model 44; it was developed after the other models and is either implemented better or priced differently. The anomalies in Mp('65; 2^{14} words), Mp('30; 2^{14} words), Mp('40; 2^{17} bytes), and Mp('44) are undoubtedly due to pricing-strategy differences. In the case of the Model 30 the incremental cost to increase the Mp size from 2^{13} to 2^{16} bytes is the addition of only a different core array (with no change in electronics), at a small incremental manufacturing cost of goods.

The Mp size range within a model varies by a factor of 8 for Models 30, 40, 44, 50, 65, and 75, although by only a factor of 4 at the ends of the line (Models 20 and 91). The Mp implementation is usually a single common set of electronics to drive 2^{14} (16,384) words in a square or coincident-current-selection system of 2^7 by 2^7. These square points are indicated on the graph, and they should be the most economical memories. Smaller Mp's are implemented simply by using smaller core-memory arrays, but with the same basic electronic configuration, e.g., the Model 30 above. Larger Mp's are obtained by replicating the whole Mp system including the core array and the electronics.

An Mp size range of 8 for a given model presupposes a certain structuring of problems. That is, the models assume a fixed relationship between Pc capacity and Mp size requirements. An ideal system might let Pc power, Pc quantity, Mp power, and Mp size be completely variable. These parameters would all be selected independently to match the work load.

Central processors

The relative Pc powers (in 360 instructions/s) and costs are given in the graph of Fig. 19 and in Table 1. The most significant fact from the graph is that the cost/power ratio is roughly constant for each of the Pc's (especially if we ignore Model 44 and Model 50). Figure 19 gives the relative computing power versus cost for various configurations. Table 1 also shows a number of relationships. One interesting relationship (Table 1) is the ratio of actual Pc power to maximum possible Pc power for a model. This can be based on Mp utilization:

$$\frac{\text{Actual Pc power}}{\text{Maximum Pc power}} = \sim \frac{\text{Mp cycles utilized by Pc}}{\text{Mp cycles available}}$$

This ratio must be less than 1 unless there are many Pc's or a single Pc has more power than Mp. In every case, the Pc is far from fully utilizing the Mp. The technique of buffering instructions in a local Pc memory can increase this ratio to be >1 (although no computers ever do so). In the higher model numbers the utilization is low because a large number of cycles have to be available in order to avoid conflicts when a given cycle is requested—using an Mp with a long t.cycle. In the case of Model 25, the cycles are lost because the microprogram is being executed from Mp. (A ratio of 0.045 indicates 21 cycles are used for microprograms to every 1 of program.)

In the case of the Model 30 the power is limited by holding the general registers in Mp. For example, by using an additional fast M to hold the general registers and working data, the Pc power could increase. Unfortunately, such a change might cause the cost of other parts of the system to be increased, so that it would not be just a simple incremental addition. The C('30) performs well for the field-scan problem [Solomon, 1966] (see Table 1). The data structure for the field-scan problem coincides with the 1-byte Mp organization. C('65) and C('75) perform the worst for field scan because of the mismatch between Mp organization (8 bytes) and program data (1 byte).

C('65) and C('75) have the same Mp structure and hence have the same potential power available from Mp. In the case

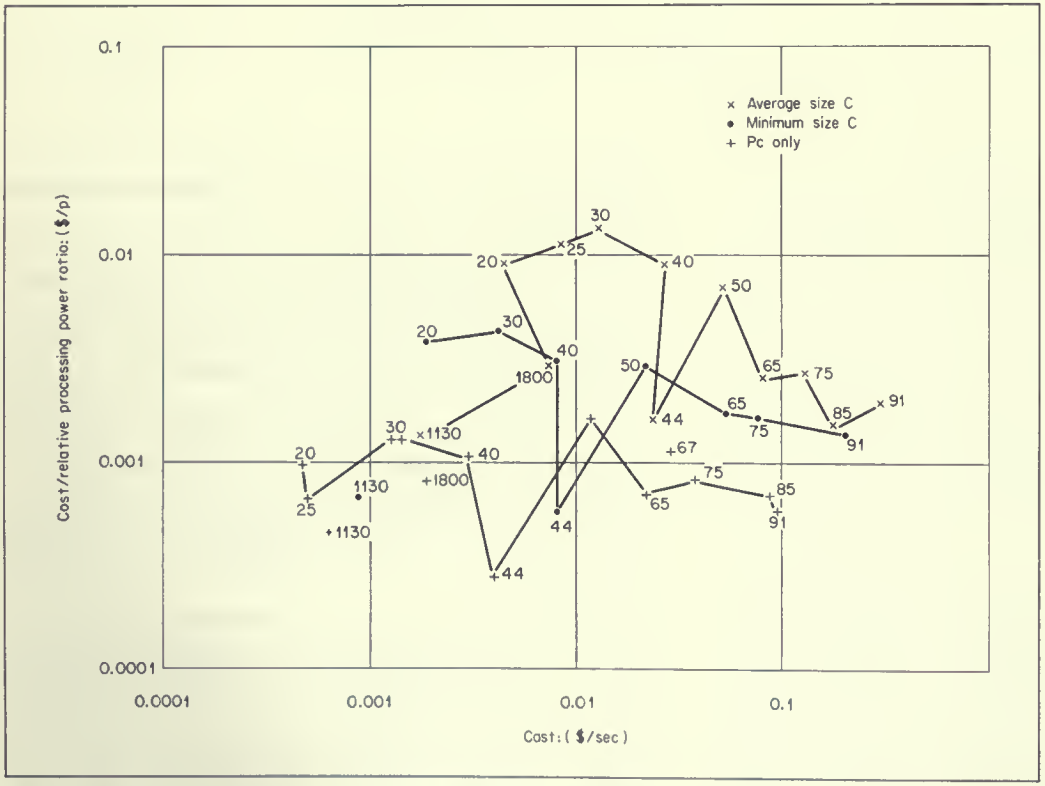


Fig. 19. Graph of IBM System/360 cost/processing power ratio versus cost.

of C('75) the power of the Mp is more nearly utilized. Unfortunately for the more complex Mp structures, which have more potential Mp cycles, the Pc is not able to utilize them. The C('65) and C('75) have several registers concerned with obtaining the next instruction and holding it for execution while other instructions are obtained (look-ahead). The hardwired Model 75 Pc may account for the improvement over the Model 65 P.microprogrammed.

The performance of C('20) is inaccurately high since it is a limited subset of the 360 ISP. (C('20) does not have floating-point or fixed-point multiply and divide instructions, and it has only eight 16-bit general registers.) The hardwired Model 44 has a better cost/power characteristic than any of the other C's, by any measured criteria (see Fig. 19). In the case of the Model 44, the Pc price also includes Ms.disk. Perhaps the Model 44, designed initially for real-time scientific problem solving, is priced more competitively with similar machines (DEC PDP-10 and SDS Sigma 5, 7), whereas the other models compete in a performance-insensitive, competition-free market for general-purpose business data processing. Thus its anomalous position may be due to external market pressures and not manufacturing cost.

The design of the IBM System/360 models is undoubtedly predicated on the basis that performance or computing power is proportional to the cost raised to some power, g, greater than 1: power = k × cost^g; where $g > 1$.[1] Almost all models follow the above relationship with $g > 1$. When $g > 1$ there is an advantage to have large configurations since the cost/computation will decrease. If $g \leq 1$, then an alternative implementation for the 360 C's would simply use multiple C's or Pc's to obtain the same power. Unfortunately, such an approach does not provide for the interconnection of the components to function as a single unit. In many cases a single task cannot be broken into a number of parallel and independent subtasks. If the performance for the system varied by a factor of 100, then 100 Pc's or C's would be placed together. From Table 1 we see a power range of about 314 corresponds to a cost range of 65 to 114 (which tells us $g < 2$).

The following discussion takes computing power to be measured by instructions per second and Mp (size; t.cycle). Costs are measured in dollars per second of rental time. The graph (Fig. 20) shows the relationship to computing power p and costs. The power (actually p.Pc) is taken from the measures of instruction times for certain fixed work. Solomon observed Grosch's law to hold for Models 30, 40, 50, 65, and 75. This line is drawn in Fig. 20 for C(cost.average). Considering Models 20, 25, 44, 85, and 91, a line with a less steep slope might fit the points better. If we consider C(cost.minimum), $g < 2$; considering only Pc, a $g = 1$ might be appropriate (see Fig. 20) in which the power/cost is essentially constant with cost.

Pc(cost)/Mp(cost.avg) : = c.Pc/c.avg.Mp = ~ 1.1, the ratio of processor to memory cost

C(cost.min)/C(cost.avg) : = c.min.C/c.avg.C = ~ 0.47, the ratio of the smallest computer configuration to an average configuration

Pc(cost)/C(cost.avg) : = c.Pc/c.avg.C = ~ 0.23, the ratio of processor to computer cost

These are averages over all the series and can be rather misleading. For example, in higher-numbered models the C(cost.min)/C(cost.avg) : = c.min.C/c.avg.C is about 0.6. whereas in lower-numbered models the ratio is 0.3. We might have expected this, since it indicates that a higher proportion of system cost is in Ms and T on lower-number models.

An alternative computer series based on multiprocessing

In this section we suggest an alternative design providing a wide range of computing power but using multiprocessing. That is, rather than building a higher-performance model, we would have multiple lower-performance models. On the surface, this appears feasible only if the cost of the processor is a relatively small part of the computer, and if for a particular configuration there are memory cycles available in the system (so that a more costly memory system is not required). It is also desirable that the proposed multiprocessor configurations have rather large Mp's so that it can be assumed there will be several jobs in Mp waiting to run; i.e., we should be able to multiprogram rather than do parallel processing. These conditions are satisfied with the System/360 models. Although we do not address the question of development cost, it is clear that a multiprocessor system would have a lower development cost because fewer processors would be required. Within IBM we can assume that the development cost tends to go to zero because of the large production; unfortunately, even for IBM, the training cost for servicemen and salesmen does not go to zero but is proportional to the number of products. Thus, we would anticipate savings by having a smaller line.

The multiprocessor view is presented in Table 4; namely, we suggest dropping Models 20, 30, 40, 50, 65, 75, 85, and 91.

[1] Herb Grosch [Grosch, 1953] first noted this relationship and estimated g to be 2; thus we use g for this exponent. Adams suggested $g = \frac{1}{2}$ [Adams, 1962]. See also The Economics of Computers [Sharpe, 1969].

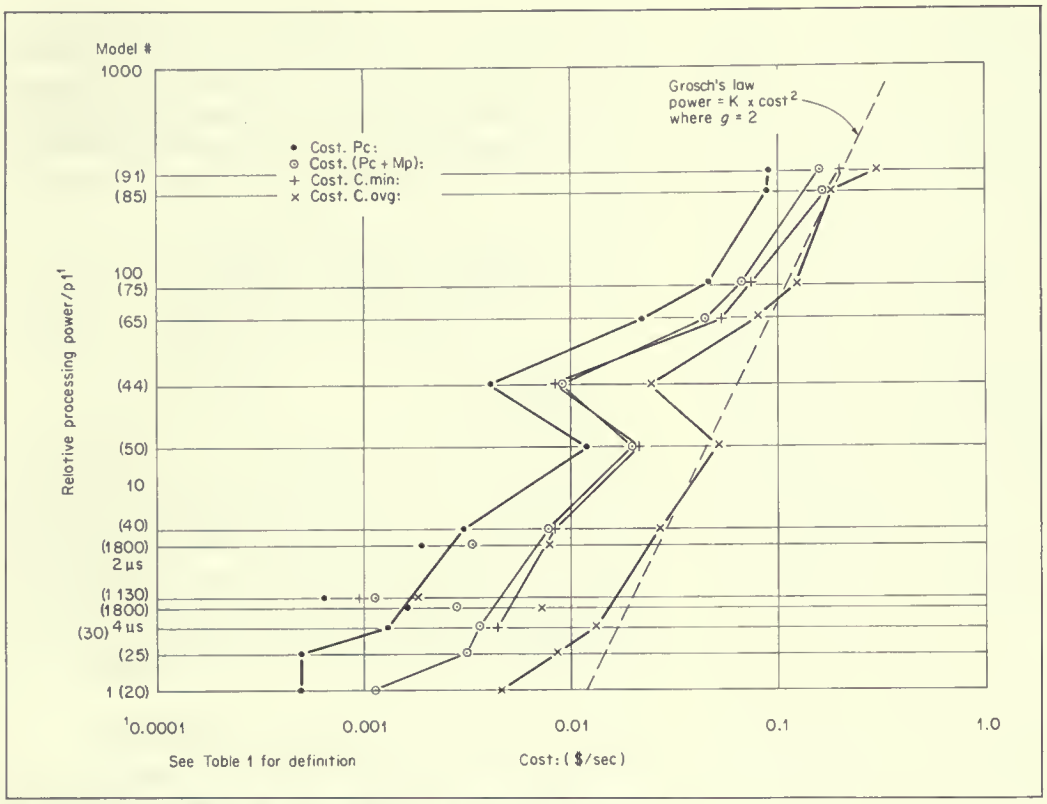


Fig. 20. Graph of IBM System/360 relative processing power versus cost.

These would be replaced with only Models 25 and 44. Note there are Pc's in Table 4 (other than 25 and 44) which when multiprocessed can perform better for lower cost, e.g., 2 Model 65's are >1 Model 75, for about the same cost. Admittedly there are major problems in multiprocessing with 11 Pc's, but other existence proofs [Anderson, 1961] have shown that two to four Pc's can be effective (Chap. 36). If we ignore Models 85 and 91, the worst case is for a maximum of four Pc's needed to obtain the power of model 40. Note that in the above cases the processor cost is about one-half the cost of a single Pc. This factor of 2 might be used to answer critics of the scheme. The reasons against the scheme are: There have to be good switches between Mp and Pc's; there has to be communication among the Pc's (which is about the same as what the Pc-Pio communication should be); and there has to be knowledge of the program environment to split tasks apart to run in parallel.

A less radical suggestion is also presented in Table 4: namely, examining the number of processor models which can be used to provide processing power for the next highest model. Actually, if we carry this view further and were forced to build such a system, the view that the ideal machines are the Model 25 and 44 would undoubtedly change. Model 25 and 44 exist and can be used for the argument. The reader should note that there is a major flaw in our argument using a Model 25. The microprogrammed Model 25 Pc cost should include a 16-kby memory for the microprogram (actually one Mp should be included for each Pc to avoid memory-request conflict). Alternatively, if we use the Model 25 directly without a microprogram, we would lose performance range. With our present knowledge of multiprocessors, a responsible engineer would hardly suggest building a multiprocessor system with 11 processors as a sure-fire money-making venture. A more reasonable alternative would be to use the multiprocessor Model 75 as an alternative to Models 85 and 91. A reasonably safe alternative would be three basic processors and a four-processor multiprocessor structure. For a power range of 320:1, then the processors could be 1, 20, 80, giving powers of 1, 2, 3, 4, 20, 40, 60, 80, 160, 240, 320. This structure would leave a gap of a factor of

Table 4 IBM System/360 Pc (power: cost) and an alternative design based on multiprocessors

Given			*Proposed multiprocessor alternatives*			
Pc.model	*Pc.power*	*Pc.cost*	*Quantity.Pc*	*Pc.model*	*Pc.power*	*Pc.cost*
20	1	0.00049	1	25	1.5	0.0005
25	1.5	0.00050	1	25	1.5	0.0005
30	2	0.0013	2	25	3	0.001
			2	20	2	0.00098
40	6	0.003	4	25	6	0.002
			6	20	6	0.00294
44	30	0.0041	1	44	30	0.0041
50	15	0.012	1	44	30	0.0041
65	63	0.022	2	44	60	0.0082
75	92	0.037	3	44	90	0.012
			2	65	126	0.044
85	252	0.087	8	44	240	0.033
91	314	0.091	11	44	330	0.045

5 between a 4 × 1 power processor and 20 power processor. The largest gap in the System/360 is a factor of 3 between Models 30 and 40.

Conclusions

The IBM System/360, by achieving a production record, has fulfilled its principal design objective. The technical goals, however, are of interest to us here. The most interesting aspect of the design is achieving a performance range of 314 to 1 over a series of models, with a primary-memory size range of 2,048 to 1 for various computer configurations. Thus a user is given a very large set of configuration alternatives. The SLT technology, though not integrated-circuit, is certainly of the third generation. Using SLT the fabrication of the models is superb.

There is a vast array of secondary-memory and terminal devices to couple with almost any other system. The System/360 is the first computer to make extensive use of microprogramming. Microprogramming is used for the definition of the System/360 instruction-set processor, but, more important, microprograms define previous IBM computers so that a user can operate satisfactorily during the interim period when older programs are being updated to use the System/360. There are provisions for multicomputer structures. Within a single computer structure there is adequate means of peripheral switching so that reliable and high-performance structures can be assembled. Early structures do not provide multiprocessing; we have suggested multiprocessing as a technique to achieve the same performance-range objectives. The io processor, though rather elaborate, provides a certain commonality.

The instruction-set processor for the System/360, based on a general-registers structure, appears to be overly complex, yet incomplete, because there are so many data types. The addressing mechanism and lack of multiprogramming ability make the System/360 a hard machine to appreciate fully. Although we praise microprogramming as a means of accomplishing compatibility with the past, it appears to stand in the way of getting the most performance from the hardware. Perhaps of most significance, the System/360 may have a greater lifetime than any past computer.

Selected Bibliography

Architecture and logical structure: AmdaG64a (TeagH65)[1], BlaaG64a[2], BlaaG64b[2]; General implementations: AmdaG64b[2], CartW64, PadeA64[2], StevW64[2]; Microprogramming: GreeJ64, TuckS67, WebeH67; Formal description of Pc[5]; FalkA64[2]; Performance and reviews: HillJ66, SoloM66; Model 40 modifications for multiprogramming: LindA66; Model 67: ArdeB66, FikeR68, GibsC66, LaueH67; Model 85: ContC68[3], LiptJ68[3], PadeA68[3]; Model 91 architecture and technology: AndeD67[4], AndeS67[4], BolaL67[4], FlynM67[4]a, LangJ67[3], LloyR67[4], SechR67[4], TomaR67[4]; Model 92 (proposed): ContC64 (GrimR65a), AmdaG64c (GrimR65b), ChenT64 (GrimR65c); Serviceability: CartW64; Other references: AdamC62, CorbF62, GrosH53, SharW69, WilkM65; IBM reference manuals: IBM System/360 Functional characteristics manuals for each model, IBM System/360 Configurator (diagram) for each model, A22-6821-4 IBM System/360 Principles of Operation, A22-6810-8 IBM System/360 System Summary

[1]( ) denotes the review of previous article.
[2]*IBM Systems Journal*, vol. 3, nos. 2 and 3, 1964.
[3]*IBM Systems Journal*, vol. 7, no. 1, 1968.
[4]*IBM Journal of Research and Development*, vol. 11, no. 1, January, 1967.
[5]Given in A Programming Language/APL [Iverson, 1962].

Chapter 43

The structure of SYSTEM/360[1]

Part I—Outline of the logical structure

G. A. Blaauw / F. P. Brooks, Jr.

Summary A general introductory description of the logical structure of SYSTEM/360 is given. In addition, the functional units, the principal registers and formats, and the basic addressing and sequencing principles of the system are indicated.

In the SYSTEM/360 logical structure, processing efficiency and versatility are served by multiple accumulators, binary addressing, bit-manipulation operations, automatic indexing, fixed and variable field lengths, decimal and hexadecimal radices, and floating-point as well as fixed-point arithmetic. The provisions for program interruption, storage protection, and flexible CPU states contribute to effective operation. Base-register addressing, the standard interface between channels and input/output control units, and the machine-language compatibility among models contribute to flexible configurations and to orderly system expansion.

SYSTEM 360 is distinguished by a design orientation toward very large memories and a hierarchy of memory speeds, a broad spectrum of manipulative functions, and a uniform treatment of input/output functions that facilitates communication with a diversity of input/output devices. The overall structure lends itself to program-compatible embodiments over a wide range of performance levels.

The system, designed for operation with a supervisory program, has comprehensive facilities for storage protection, program relocation, nonstop operation, and program interruption. Privileged instructions associated with a supervisory operating state are included. The supervisory program schedules and governs the execution of multiple programs, handles exceptional conditions, and coordinates and issues input/output (I/O) instructions. Reliability is heightened by supplementing solid-state components with built-in checking and diagnostic aids. Interconnection facilities permit a wide variety of possibilities for multisystem operation.

The purpose of this discussion is to introduce the functional units of the system, as well as formats, codes, and conventions essential to characterization of the system.

[1] *IBM Sys. J*, vol. 3, no. 2, pp. 119–135, 1964.

Functional structure

The SYSTEM/360 structure schematically outlined in Fig. 1 has seven announced embodiments. Six of these, namely, Models 30, 40, 50, 60, 62, and 70, will be treated here.[1] Where requisite I/O devices, optional features, and storage capacity are present, these six models are logically identical for valid programs that contain explicit time dependencies only. Hence, even though the allowable channels or storage capacity may vary from model to model (as discussed in Chap. 44), the logical structure can be discussed without reference to specific models.

Input/output

Direct communication with a large number of low-speed terminals and other I/O devices is provided through a special *multiplexor* channel unit. Communication with high-speed I/O devices is accommodated by the *selector* channel units. Conceptually, the input/output system acts as a set of subchannels that operate concurrently with one another and the processing unit. Each subchannel, instructed by its own control-word sequence, can govern a data transfer operation between storage and a selected I/O device. A multiplexor channel can function either as one or as many subchannels; a selector channel always functions as a single subchannel. The control unit of each I/O device attaches to the channels via a standard mechanical-electrical-programming *interface.*

Processing

The processing unit has sixteen general purpose 32-bit registers used for addressing, indexing, and accumulating. Four 64-bit floating-point accumulators are optionally available. The inclusion of multiple registers permits effective use to be made of small high-speed memories. Four distinct types of processing are pro-

[1] A seventh embodiment, the Model 92, is not discussed in this paper. This model does not provide decimal data handling and has a few minor differences arising from its highly concurrent, speed-oriented organization. A paper on Model 92 is planned for future publication in the *IBM Systems Journal.*

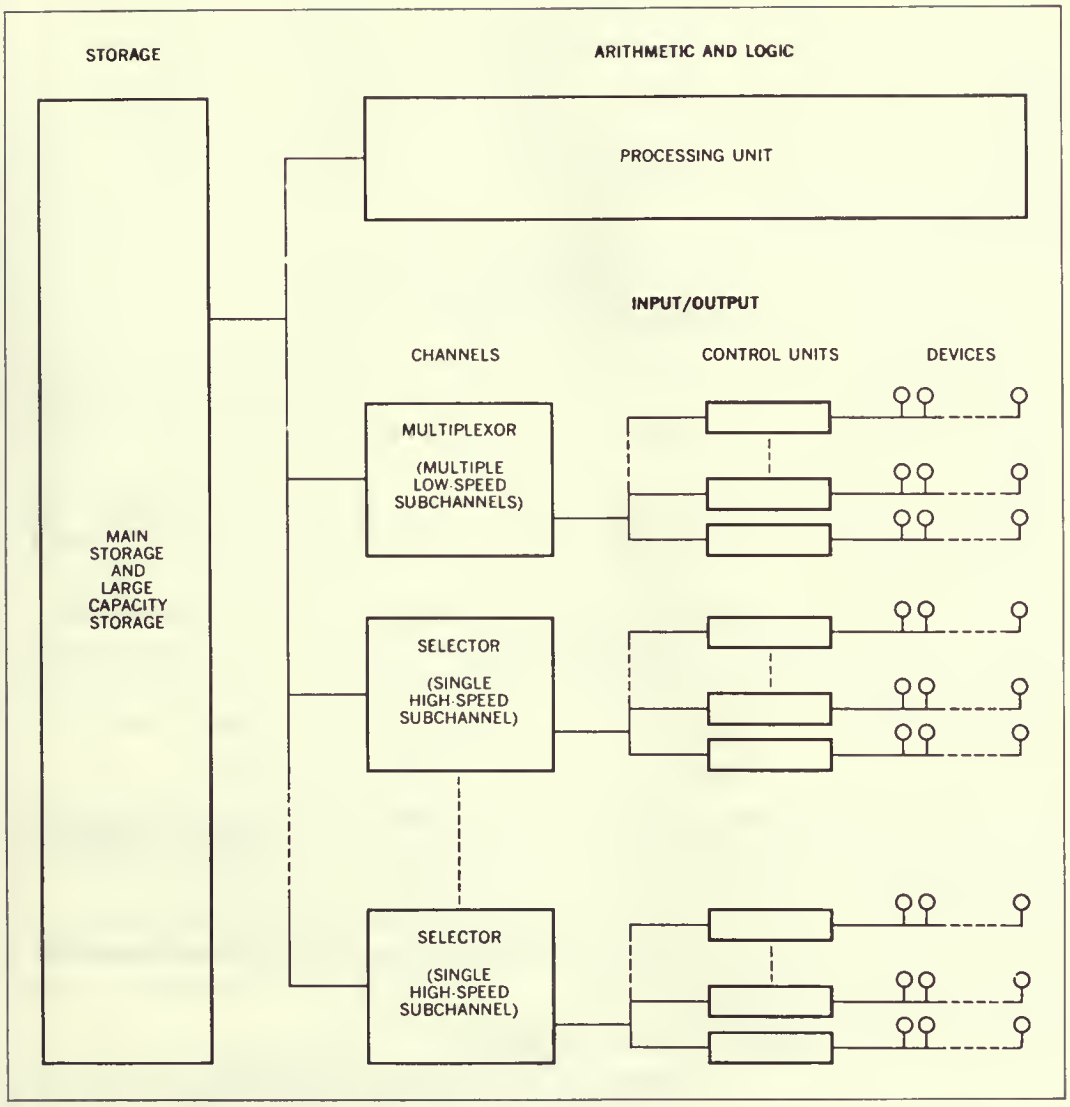


Fig. 1. Functional schematic of System/360.

vided: logical manipulation of individual bits, character strings and fixed words; decimal arithmetic on digit strings; fixed-point binary arithmetic; and floating-point arithmetic. The processing unit, together with the central control function, will be referred to as the central processing unit (CPU). The basic registers and data paths of the CPU are shown in Fig. 2.

The CPU's of the various models yield a substantial range in performance. Relative to the smallest model (Model 30), the internal performance of the largest (Model 70) is approximately 50:1 for scientific computation and 15:1 for commercial data processing.

Control

Because of the extensive instruction set, SYSTEM/360 control is more elaborate than in conventional computers. Control functions include internal sequencing of each operation; sequencing from instruction to instruction (with branching and interruption); governing of many I/O transfers; and the monitoring, signaling, timing, and storage protection essential to total system operation. The control equipment is combined with a programmed supervisor, which coordinates and issues all I/O instructions, handles excep-

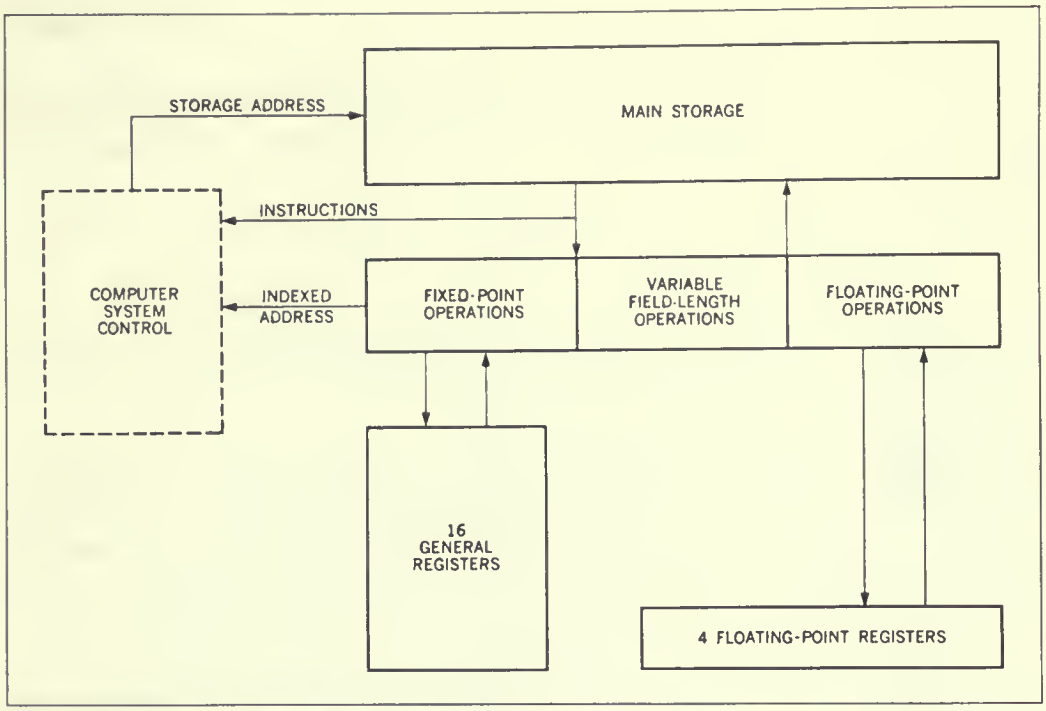


Fig. 2. Schematic of basic registers and data paths.

tional conditions, loads and relocates programs and data, manages storage, and supervises scheduling and execution of multiple programs. To a problem programmer, the supervisory program and the control equipment are indistinguishable.

The functional structure of SYSTEM/360, like that of most computers, is most concisely described by considering the data formats, the types of manipulations performed on them, and the instruction formats by which these manipulations are specified.

Information formats

The several SYSTEM/360 data formats are shown in Fig. 3. An 8-bit unit of information is fundamental to most of the formats. A consecutive group of n such units constitutes a *field of length* n. Fixed-length fields of length one, two, four, and eight are termed *bytes, halfwords, words,* and *double words,* respectively. In many instructions, the operation code implies one of these four fields as the length of the operands. On the other hand, the length is explicit in an instruction that refers to operands of variable length.

The location of a stored field is specified by the address of the leftmost byte of the field. Variable-length fields may start on any byte location, but a fixed-length field of two, four, or eight bytes must have an address that is a multiple of 2, 4, or 8, respectively. Some of the various alignment possibilities are apparent from Fig. 3.

Storage addresses are represented by binary integers in the system. Storage capacities are always expressed as numbers of bytes.

Processing operations

The SYSTEM/360 operations fall into four classes: fixed-point arithmetic, floating-point arithmetic, logical operations, and decimal arithmetic. These classes differ in the data formats used, the registers involved, the operations provided, and the way the field length is stated.

Fixed-point arithmetic

The basic arithmetic operand is the 32-bit fixed-point binary word. Halfword operands may be specified in most operations for the sake of improved speed or storage utilization. Some products and all dividends are 64 bits long, using an even-odd register pair.

Because the 32-bit words accommodate the 24-bit address, the entire fixed-point instruction set, including multiplication, division,

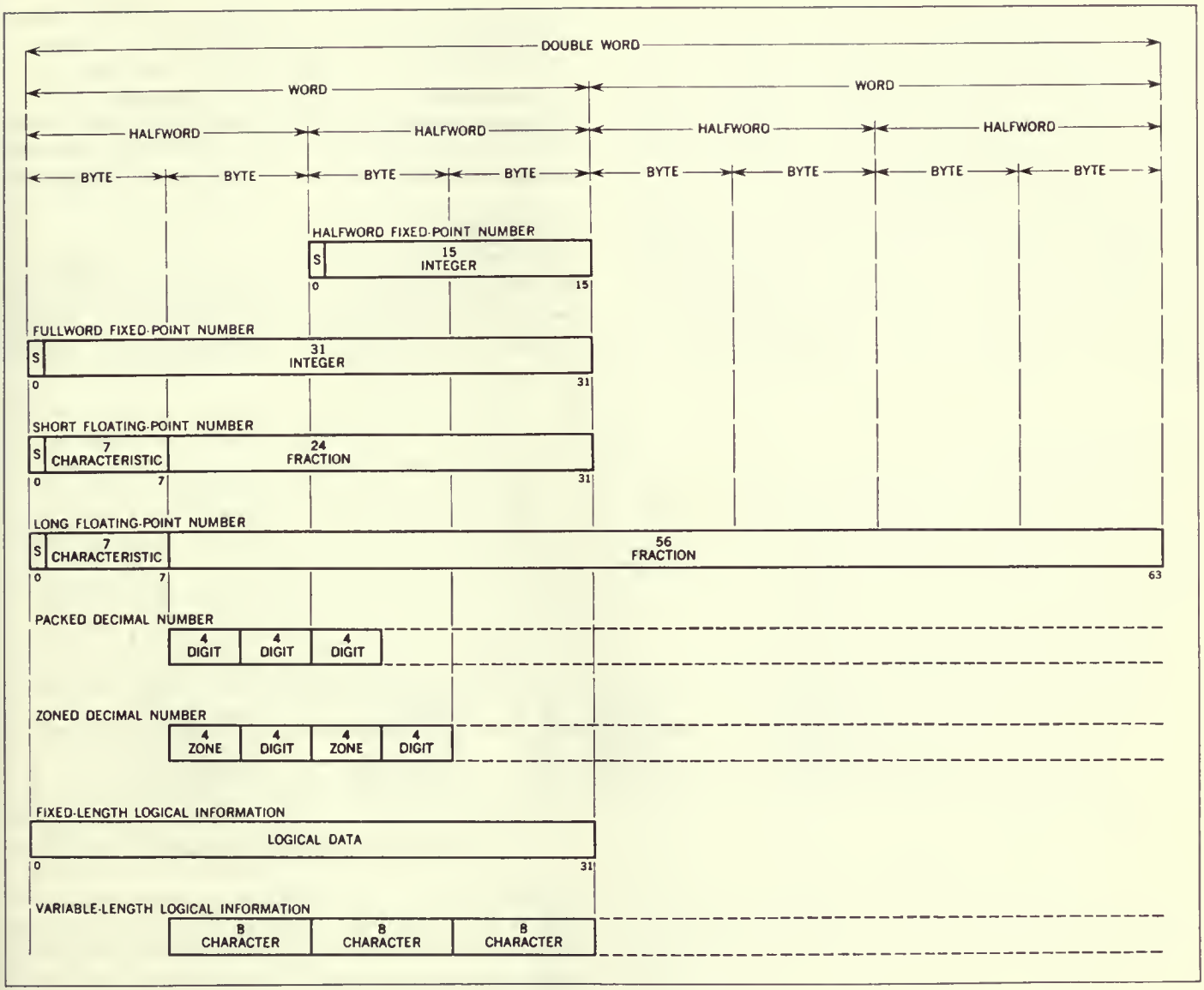


Fig. 3. The data formats.

shifting, and several logical operations, can be used in address computation. A two's complement notation is used for fixed-point operands.

Additions, subtractions, multiplications, divisions, and comparisons take one operand from a register and another from either a register or storage. Multiple-precision arithmetic is made convenient by the two's complement notation and by recognition of the carry from one word to another. A pair of conversion instructions, CONVERT TO BINARY and CONVERT TO DECIMAL, provide transition between decimal and binary radices without the use of tables. Multiple-register loading and storing instructions facilitate subroutine switching.

Floating-point arithmetic

Floating-point numbers may occur in either of two fixed-length formats—short or long. These formats differ only in the length of

the fractions, as indicated in Fig. 3. The fraction of a floating-point number is expressed in 4-bit hexadecimal (base 16) digits. In the short format, the fraction has six hexadecimal digits; in the long format, the fraction has 14 hexadecimal digits. The short length is equivalent to seven decimal places of precision. The long length gives up to 17 decimal places of precision, thus eliminating most requirements for double-precision arithmetic.

The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1 through 7 of both formats, is used to indicate this power. The characteristic is treated as an excess 64 number with a range from −64 through +63, and permits representation of decimal numbers with magnitudes in the range of 10^{-78} to 10^{75}.

Bit position 0 in either format is the fraction sign, S. The fraction of negative numbers is carried in true form.

Floating-point operations are performed with one operand from a register and another from either a register or storage. The result, placed in a register, is generally of the same length as the operands.

Logical operations

Operations for comparison, translation, editing, bit testing, and bit setting are provided for processing logical fields of fixed and variable lengths. Fixed-length logical operands, which consist of one, four, or eight bytes, are processed from the general registers.

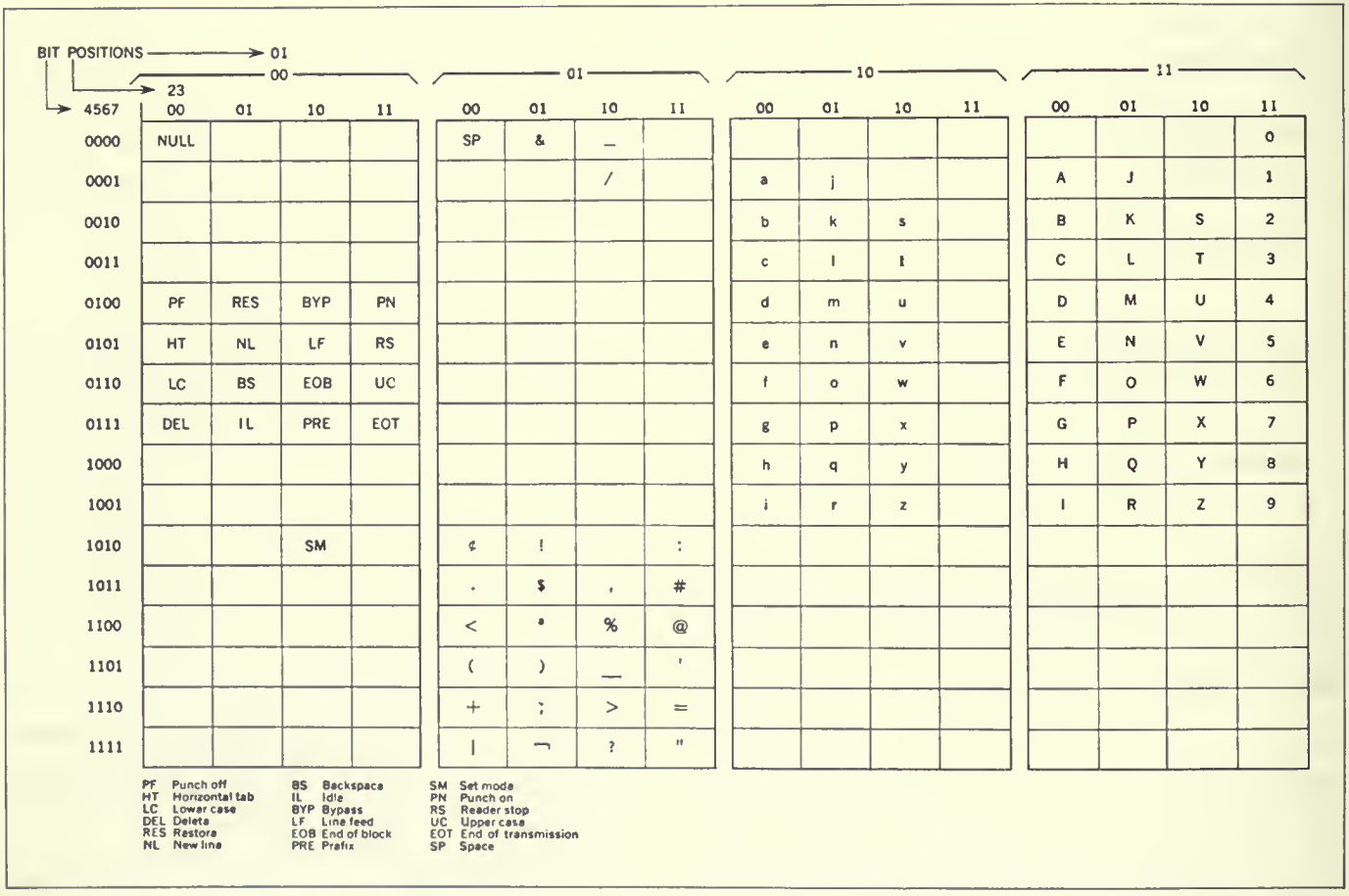
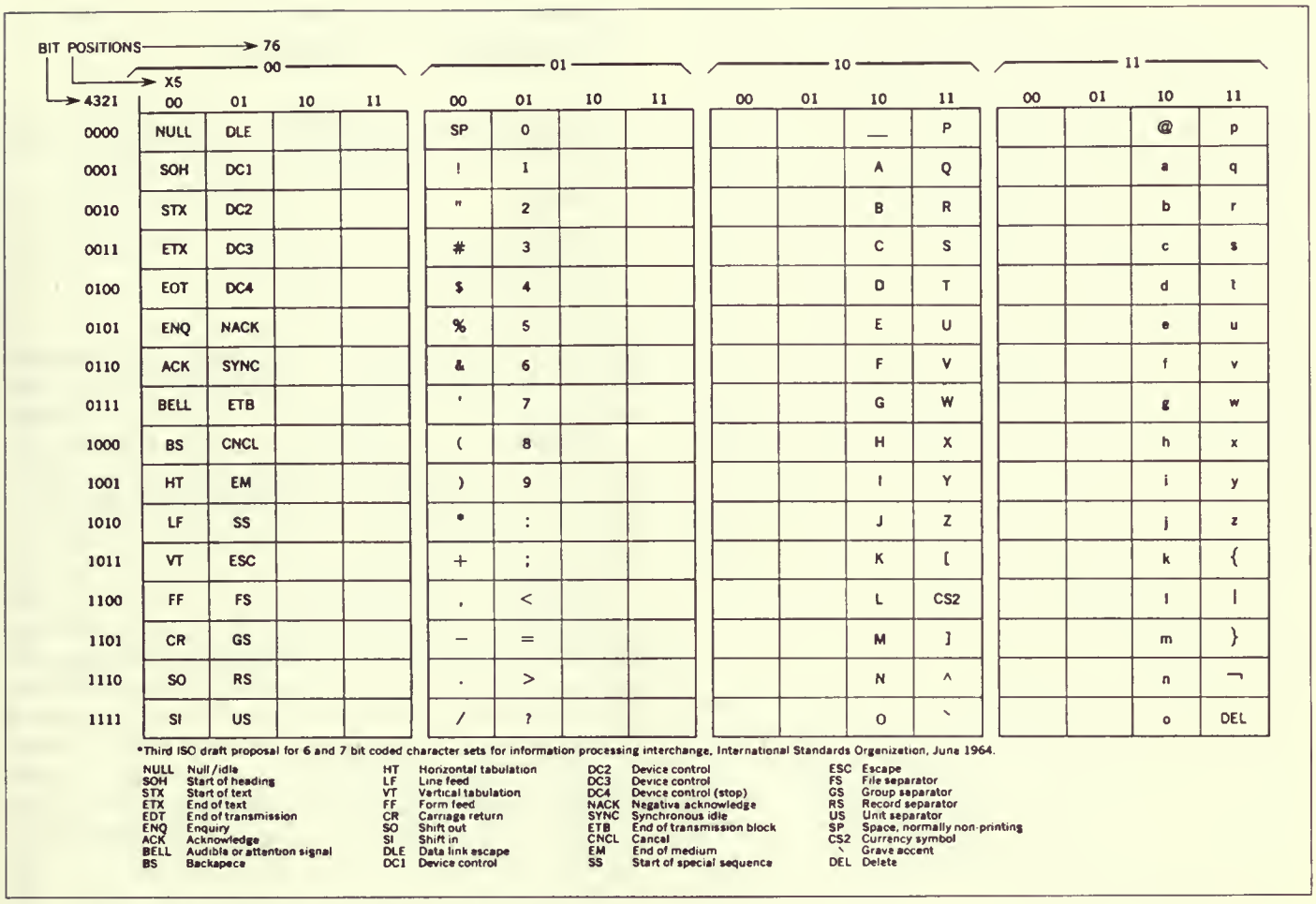


Fig. 4. Extended binary-coded-decimal interchange code.



BIT POSITIONS → 76 / XS → 00 / 4321	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NULL	DLE			SP	0					—	P			@	p
0001	SOH	DC1			!	1					A	Q			a	q
0010	STX	DC2			"	2					B	R			b	r
0011	ETX	DC3			#	3					C	S			c	s
0100	EOT	DC4			$	4					D	T			d	t
0101	ENQ	NACK			%	5					E	U			e	u
0110	ACK	SYNC			&	6					F	V			f	v
0111	BELL	ETB			'	7					G	W			g	w
1000	BS	CNCL			(	8					H	X			h	x
1001	HT	EM			)	9					I	Y			i	y
1010	LF	SS			*	:					J	Z			j	z
1011	VT	ESC			+	;					K	[			k	{
1100	FF	FS			,	<					L	CS2			l	|
1101	CR	GS			-	=					M	]			m	}
1110	SO	RS			.	>					N	^			n	¬
1111	SI	US			/	?					O	`			o	DEL

*Third ISO draft proposal for 6 and 7 bit coded character sets for information processing interchange, International Standards Organization, June 1964.

NULL	Null/idle	HT	Horizontal tabulation	DC2	Device control	ESC	Escape
SOH	Start of heading	LF	Line feed	DC3	Device control	FS	File separator
STX	Start of text	VT	Vertical tabulation	DC4	Device control (stop)	GS	Group separator
ETX	End of text	FF	Form feed	NACK	Negative acknowledge	RS	Record separator
EDT	End of transmission	CR	Carriage return	SYNC	Synchronous idle	US	Unit separator
ENQ	Enquiry	SO	Shift out	ETB	End of transmission block	SP	Space, normally non-printing
ACK	Acknowledge	SI	Shift in	CNCL	Cancel	CS2	Currency symbol
BELL	Audible or attention signal	DLE	Data link escape	EM	End of medium	`	Grave accent
BS	Backspace	DC1	Device control	SS	Start of special sequence	DEL	Delete

Fig. 5. Eight-bit representation for proposed international code.

Logical operations can also be performed on fields of up to 256 bytes, in which case the fields are processed from left to right, one byte at a time. Moreover, two powerful scanning instructions permit byte-by-byte translation and testing via tables. An important special case of variable-length logical operations is the one-byte field, whose individual bits can be tested, set, reset, and inverted as specified by an 8-bit mask in the instruction.

Character codes

Any 8-bit character set can be processed, although certain restrictions are assumed in the decimal arithmetic and editing operations. However, all character-set-sensitive I/O equipment assumes either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) of Fig. 4 or the code of Fig. 5, which is an eight-bit extension of a seven-bit code proposed by the International Standards Organization.

Decimal arithmetic

Decimal arithmetic can improve performance for processes requiring few computational steps per datum between the source input and the output. In these cases, where radix conversion from decimal to binary and back to decimal is not justified, the use of registers for intermediate results usually yields no advantage over storage-to-storage processing. Hence, decimal arithmetic is provided in SYSTEM/360 with operands as well as results located in storage, as in the IBM 1400 series. Decimal arithmetic includes

addition, subtraction, multiplication, division, and comparison.

The decimal digits 0 through 9 are represented in the 4-bit binary-coded-decimal form by 0000 through 1001, respectively. The patterns 1010 through 1111 are not valid as digits and are interpreted as sign codes: 1011 and 1101 represent a minus, the other four a plus. The sign patterns generated in decimal arithmetic depend upon the character set preferred. For EBCDIC, the patterns are 1100 and 1101; for the code of Fig. 5, they are 1010 and 1011. The choice between the two codes is determined by a mode bit.

Decimal digits, packed two to a byte, appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the rightmost four bits of the low-order byte. Operand fields can be located on any byte boundary, and can have lengths up to 31 digits and sign. Operands participating in an operation have independent lengths. Negative numbers are carried in true form. Instructions are provided for packing and unpacking decimal numbers. Packing of digits leads to efficient use of storage, increased arithmetic performance, and improved rates of data transmission. For purely decimal fields, for example, a 90,000-byte/second tape drive reads and writes 180,000 digits/second.

Instruction formats

Instruction formats contain one, two, or three halfwords, depending upon the number of storage addresses necessary for the operation. If no storage address is required of an instruction, one halfword suffices. A two-halfword instruction specifies one address; a three-halfword instruction specifies two addresses. All instructions must be aligned on halfword boundaries.

The five basic instruction formats, denoted by the format mnemonics RR, RX, RS, SI, and SS are shown in Fig. 6. RR denotes a register-to-register operation, RX a register and indexed-storage operation, RS a register and storage operation, SI a storage and immediate-operand operation, and SS a storage-to-storage operation.

In each format, the first instruction halfword consists of two parts. The first byte contains the operation code. The length and format of an instruction are indicated by the first two bits of the operation code.

The second byte is used either as two 4-bit fields or as a single 8-bit field. This byte is specified from among the following:

Four-bit operand register designator (R)

Four-bit index register designator (X)

Four-bit mask (M)

Four-bit field length specification (L)

Eight-bit field length specification

Eight-bit byte of immediate data (I)

The second and third halfwords each specify a 4-bit base register designator (B), followed by a 12-bit displacement (D).

Addressing

An effective storage address E is a 24-bit binary integer given, in the typical case, by

$$E = B + X + D$$

where B and X are 24-bit integers from general registers identified by fields B and X, respectively, and the displacement D is a 12-bit integer contained in every instruction that references storage.

The base B can be used for static relocation of programs and data. In record processing, the base can identify a record; in array calculations, it can specify the location of an array. The index X can provide the relative address of an element within an array. Together, B and X permit double indexing in array processing.

The displacement provides for relative addressing of up to 4095 bytes beyond the element or base address. In array calculations, the displacement can identify one of many items associated with an element. Thus, multiple arrays whose indices move together are best stored in an interleaved manner. In the processing of records, the displacement can identify items within a record.

In forming an effective address, the base and index are treated as unsigned 24-bit positive binary integers and the displacement as a 12-bit positive binary integer. The three are added as 24-bit binary numbers, ignoring overflow. Since every address is formed with the aid of a base, programs can be readily and generally relocated by changing the contents of base registers.

A zero base or index designator implies that a zero quantity must be used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance. Initialization, modification, and testing of bases and indices can be carried out by fixed-point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, or BRANCH ON INDEX instructions. LOAD EFFECTIVE ADDRESS provides not only a convenient housekeeping operation, but also, when the same register is specified for result and operand, an immediate register-incrementing operation.

Sequencing

Normally, the CPU takes instructions in sequence. After an instruction is fetched from a location specified by the instruction

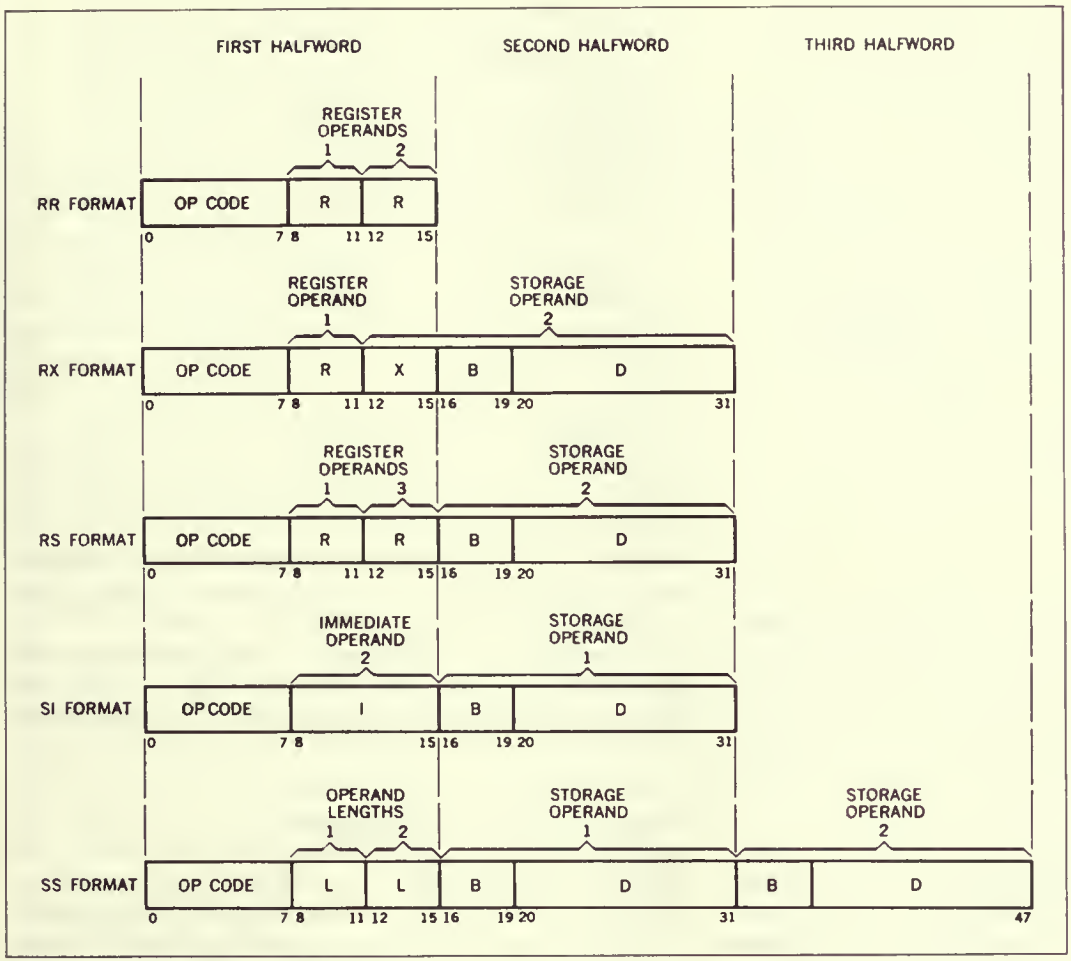


Fig. 6. Five basic instruction formats.

counter, the instruction counter is increased by the number of bytes in the instruction.

Conceptually, all halfwords of an instruction are fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage word size and overlap of instruction execution with storage access may cause the actual instruction fetching to be different. Thus, an instruction can be modified by the instruction that immediately precedes it in the instruction stream, and cannot effectively modify itself during execution.

Branching

Most branching is accomplished by a single BRANCH ON CONDITION operation that inspects a 2-bit *condition register.* Many of the arithmetic, logical, and I/O operations indicate an outcome by setting the condition register to one of its four possible states. Subsequently a conditional branch can select one of the states as a criterion for branching. For example, the condition code reflects such conditions as non-zero result, first operand high, operands equal, overflow, channel busy, zero, etc. Once set, the condition register remains unchanged until modified by an instruction execution that reflects a different condition code.

The outcome of address arithmetic and counting operations can be tested by a conditional branch to effect loop control. Two instructions, BRANCH ON COUNT and BRANCH ON INDEX, provide for one-instruction execution of the most common arithmetic-test combinations.

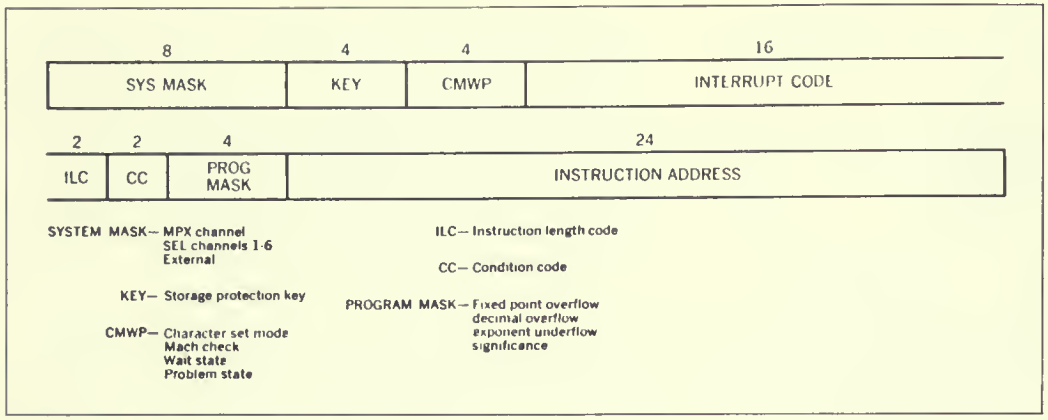


Fig. 7. Program status word format.

Program status word

A program status word (PSW), a double word having the format shown in Fig. 7, contains information required for proper execution of a given program. A PSW includes an instruction address, condition code, and several mask and mode fields. The active or controlling PSW is called the *current* PSW. By storing the current PSW during an interruption, the status of the interrupted program is preserved.

Interruption

Five classes of interruption conditions are distinguished: input/output, program, supervisor call, external, and machine check.

For each class, two PSW's, called *old* and *new*, are maintained in the main-storage locations shown in Table 1. An interruption in a given class stores the current PSW as an old PSW and then takes the corresponding new PSW as the current PSW. If, at the conclusion of the interruption routine, old and current PSW's are interchanged, the system can be restored to its prior state and the interrupted routine can be continued.

The system mask, program mask, and machine-check mask bits in the PSW may be used to control certain interruptions. When masked off, some interruptions remain pending while others are merely ignored. The system mask can keep I/O and external interruptions pending, the program mask can cause four of the 15 program interruptions to be ignored, and the machine-check mask can cause machine-check interruptions to be ignored. Other interruptions cannot be masked off.

Appropriate CPU response to a special condition in the channels and I/O units is facilitated by an I/O *interruption*. The addresses of the channel and I/O unit involved are recorded in the old PSW. Related information is preserved in a channel status word that is stored as a result of the interruption.

Unusual conditions encountered in a program create *program interruptions*. Eight of the fifteen possible conditions involve overflows, improper divides, lost significance, and exponent underflow.

Table 1 Permanent storage assignments

Address	*Byte length*	*Purpose*
0	8	Initial program loading PSW
8	8	Initial program loading CCW 1
16	8	Initial program loading CCW 2
24	8	External old PSW
32	8	Supervisor call old PSW
40	8	Program old PSW
48	8	Machine check old PSW
56	8	Input/output old PSW
64	8	Channel status word
72	4	Channel address word
76	4	Unused
80	4	Timer
84	4	Unused
88	8	External new PSW
96	8	Supervisor call new PSW
104	8	Program new PSW
112	8	Machine check new PSW
120	8	Input/output new PSW
128		Diagnostic scan-out area†

† The size of the diagnostic scan-out area is configuration dependent.

The remaining seven deal with improper addresses, attempted execution of privileged instructions, and similar conditions.

A *supervisor-call interruption* results from execution of the instruction SUPERVISOR CALL. Eight bits from the instruction format are placed in the interruption code of the old PSW, permitting a message to be associated with the interruption. SUPERVISOR CALL permits a problem program to switch CPU control back to the supervisor.

Through an *external interruption,* a CPU can respond to signals from the interruption key on the system control panel, the timer, other CPU's, or special devices. The source of the interruption is identified by an interruption code in bits 24 through 31 of the PSW.

The occurrence of a machine check (if not masked off) terminates the current instruction, initiates a diagnostic procedure, and subsequently effects a *machine-check interruption.* A machine check is occasioned only by a hardware malfunction; it cannot be caused by invalid data or instructions.

Interrupt priority

Interruption requests are honored between instruction executions. When several requests occur during execution of an instruction, they are honored in the following order: (1) machine check, (2) program or supervisor call, (3) external, and (4) input/output. Because the program and supervisor-call interruptions are mutually exclusive, they cannot occur at the same time.

If a machine-check interruption occurs, no other interruptions can be taken until this interruption is fully processed. Otherwise, the execution of the CPU program is delayed while PSW's are appropriately stored and fetched for each interruption. When the last interruption request has been honored, instruction execution is resumed with the PSW last fetched. An interruption subroutine is then serviced for each interruption in the order (1) input/output, (2) external, and (3) program or supervisor call.

Program status

Overall CPU status is determined by four alternatives: (1) *stopped* versus *operating* state, (2) *running* versus *waiting* state, (3) *masked* versus *interruptable* state, and (4) *supervisor* versus *problem* state.

In the stopped state, which is entered and left by manual procedure, instructions are not executed, interruptions are not accepted, and the timer is not updated. In the operating state, the CPU is capable of executing instructions and of being interrupted.

In the running state, instruction fetching and execution proceeds in the normal manner. The wait state is typically entered by the program to await an interruption, for example, an I/O interruption or operator intervention from the console. In the wait state, no instructions are processed, the timer is updated, and I/O and external interruptions are accepted unless masked. Running versus waiting is determined by the setting of a bit in the current PSW.

The CPU may be interruptable or masked for the system, program, and machine interruptions. When the CPU is interruptable for a class of interruptions, these interruptions are accepted. When the CPU is masked, the system interruptions remain pending, but the program and machine-check interruptions are ignored. The interruptable states of the CPU are changed by altering mask bits in the current PSW.

In the problem state, processing instructions are valid, but all I/O instructions and a group of control instructions are invalid. In the supervisor state, all instructions are valid. The choice of problem or supervisor state is determined by a bit in the PSW.

Supervisory facilities

Timer

A timer word in main storage location 80 is counted down at a rate of 50 or 60 cycles per second, depending on power line frequency. The word is treated as a signed integer according to the rules of fixed-point arithmetic. An external interrupt occurs when the value of the timer word goes from positive to negative. The full cycle time of the timer is 15.5 hours.

As an interval timer, the timer may be used to measure elapsed time over relatively short intervals. The timer can be set by a supervisory-mode program to any value at any time.

Direct control

Two instructions, READ DIRECT and WRITE DIRECT, provide for the transfer of a single byte of information between an external device and the main storage of the system. These instructions are intended for use in synchronizing CPU's and special external devices.

Storage protection

For protection purposes, main storage is divided into blocks of 2,048 bytes each. A four-bit *storage key* is associated with each block. When a store operation is attempted by an instruction, the *protection key* of the current PSW is compared with the storage key of the affected block. When storing is specified by a channel operation, a protection key supplied by the channel is used as the

comparand. The keys are said to *match* if equal or if either is zero. A storage key is not part of addressable storage, and can be changed only by privileged instructions. The protection key of the CPU program is held in the current PSW. The protection key of a channel is recorded in a status word that is associated with the channel operation.

When a CPU operation causes a protection mismatch, its execution is suppressed or terminated, and the program execution is altered by an interruption. The protected storage location always remains unchanged. Similarly, protection mismatch due to an I/O operation terminates data transmission in such a way that the protected storage location remains unchanged.

Multisystem operation

Communication between CPU's is made possible by shared control units, interconnected channels, or shared storage. Multisystem operation is supported by provisions for automatic relocation, indication of malfunctions, and CPU initialization.

Automatic relocation applies to the first 4,096 bytes of storage, an area that contains all permanent storage assignments and usually has special significance for supervisory programs. The relocation is accomplished by inserting a 12-bit prefix in each address whose high-order 12 bits are zero. Two manually set prefixes permit the use of an alternate area when storage malfunction occurs; the choice between prefixes is preserved in a trigger that is set during initial program loading.

To alert one CPU to the possible malfunction of another, a machine-check signal from a given CPU can serve as an external interruption to another CPU. By another special provision, initial program loading of a given CPU can be initiated by a signal from another CPU.

Input/output

Devices and control units

Input/output devices include card equipment, magnetic tape units, disk storage, drum storage, typewriter-keyboard devices, printers, teleprocessing devices, and process control equipment. The I/O devices are regulated by control units, which provide the electrical, logical, and buffering capabilities necessary for I/O device operation. From the programming point of view, most control-unit and I/O device functions are indistinguishable. Sometimes the control unit is housed with an I/O device, as in the case of the printer.

A control unit functions only with those I/O devices for which it is designed, but all control units respond to a standard set of signals from the channel. This control-unit-to-channel connection, called the I/O *interface*, enables the CPU to handle all I/O operations with only four instructions.

I/O instructions

Input/output instructions can be executed only while the CPU is in the supervisor state. The four I/O instructions are START I/O, HALT I/O, TEST CHANNEL, and TEST I/O.

START I/O initiates an I/O operation; its address field specifies a channel and an I/O device. If the channel facilities are free, the instruction is accepted and the CPU continues its program. The channel independently selects the specified I/O device. HALT I/O terminates a channel operation. TEST CHANNEL sets the condition code in the PSW to indicate the state of the channel addressed by the instruction. The code then indicates one of the following conditions: channel available, interruption condition in channel, channel working, or channel not operational. TEST I/O sets the PSW condition code to indicate the state of the addressed channel, subchannel, and I/O device.

Channels

Channels provide the data path and control for I/O devices as they communicate with main storage. In the multiplexor channel, the single data path can be time-shared by several low-speed devices (card readers, punches, printers, terminals, etc.) and the channel has the functional character of many subchannels, each of which services one I/O device at a time. On the other hand, the selector channel, which is designed for high-speed devices, has the functional character of a single subchannel. All subchannels respond to the same I/O instructions. Each can fetch its own control word sequence, govern the transfer of data and control signals, count record lengths, and interrupt the CPU on exceptions.

Two modes of operation, *burst* and *multiplex*, are provided for multiplexor channels. In burst mode, the channel facilities are monopolized for the duration of data transfer to or from a particular I/O device. The selector channel functions only in the burst mode. In multiplex mode, the multiplexor channel sustains several simultaneous I/O operations: bytes of data are interleaved and then routed between selected I/O devices and desired locations in main storage.

At the conclusion of an operation launched by START I/O or TEST I/O, an I/O interruption occurs. At this time a channel status word (CSW) is stored in location 64. Figure 8 shows the CSW format. The CSW provides information about the termination of the I/O operation.

Successful execution of START I/O causes the channel to

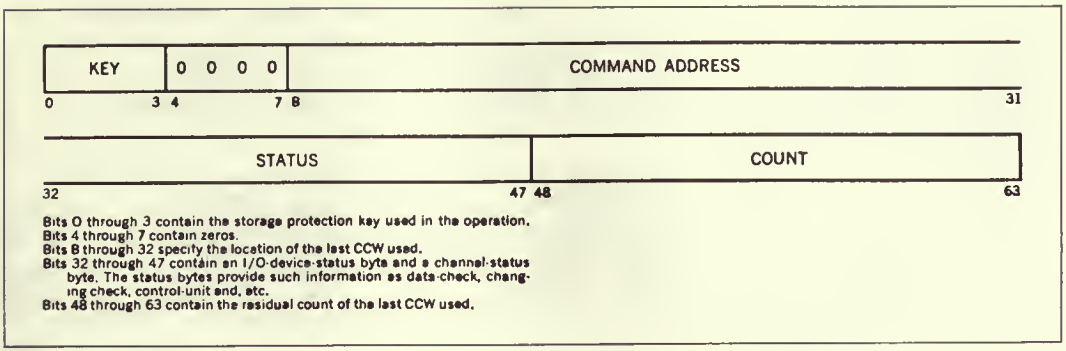


Fig. 8. Channel status word format.

fetch a channel address word from main-storage location 72. This word specifies the storage-protection key that governs the I/O operation, as well as the location of the first eight bytes of information that the channel fetches from main storage. These 64 bits comprise a channel command word (CCW). Figure 9 shows the CCW format.

Channel program

One or more CCW's make up the channel program that directs channel operations. Each CCW points to the next one to be fetched, except for the last in the chain which so identifies itself.

Six channel commands are provided: read, write, read backward, sense, transfer in channel, and control. The read command defines an area in main storage and causes a read operation from the selected I/O device. The write command causes data to be written by the selected device. The read-backward command is akin to the read command, but the external medium is moved in the opposite direction and bytes read backward are placed in descending main storage locations.

The control command contains information, called an *order*, that is used to control the selected I/O device. Orders, peculiar to the particular I/O device in use, can specify such functions as rewinding a tape unit, searching for a particular track in disk storage, or line skipping on a printer. In a functional sense, the CPU executes I/O instructions, the channels execute commands, and the control units and devices execute orders.

The sense command specifies a main storage location and transfers one or more bytes of status information from the selected control unit. It provides details concerning the selected I/O device, such as a stacker-full condition of a card reader or a file-protected condition of a magnetic-tape reel.

A channel program normally obtains CCW's from a consecutive string of storage locations. The string can be broken by a transfer-in-channel command that specifies the location of the next CCW to be used by the channel. External documents, such as punched cards or magnetic tape, may carry CCW's that can be used by the channel to govern the reading of the documents.

The input/output interruptions caused by termination of an

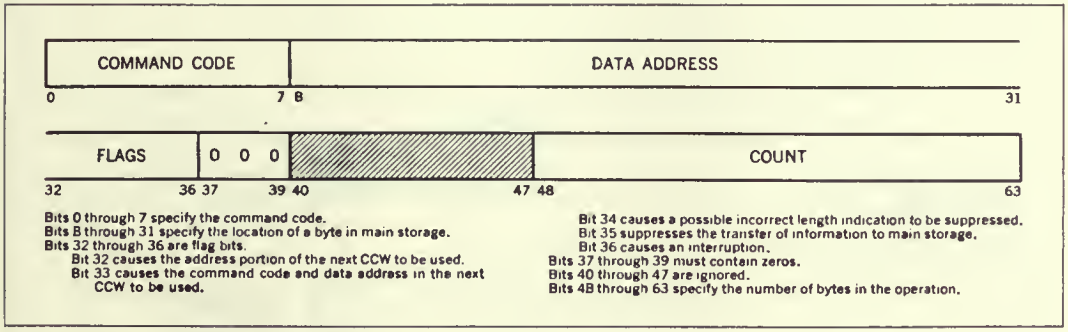


Fig. 9. Channel command word format.

Table 2 System/360 instructions

RR Format

xxxx	*Branching and status switching* 0000xxxx	*Fixed-point fullword and logical* 0001xxxx	*Floating-point long* 0010xxxx	*Floating-point short* 0011xxxx
0000		LPR LOAD POSITIVE	LPDR LOAD POSITIVE	LPER LOAD POSITIVE
0001		LNR LOAD NEGATIVE	LNDR LOAD NEGATIVE	LNER LOAD NEGATIVE
0010		LTR LOAD AND TEST	LTDR LOAD AND TEST	LTER LOAD AND TEST
0011		LCR LOAD COMPLEMENT	LCDR LOAD COMPLEMENT	LCER LOAD COMPLEMENT
0100	SPM SET PROGRAM MASK	NR AND	HDR HALVE	HER HALVE
0101	BALR BRANCH AND LINK	CLR COMPARE LOGICAL		
0110	BCTR BRANCH ON COUNT	OR OR		
0111	BCR BRANCH/CONDITION	XR EXCLUSIVE OR		
1000	SSK SET KEY	LR LOAD	LDR LOAD	LER LOAD
1001	ISK INSERT KEY	CR COMPARE	CDR COMPARE	CER COMPARE
1010	SVC SUPERVISOR CALL	AR ADD	ADR ADD N	ALR ADD N
1011		SR SUBTRACT	SDR SUBTRACT N	SER SUBTRACT N
1100		MR MULTIPLY	MDR MULTIPLY	MER MULTIPLY
1101		DR DIVIDE	DDR DIVIDE	DER DIVIDE
1110		ALR ADD LOGICAL	AWR ADD U	AUR ADD U
1111		SLR SUBTRACT LOGICAL	SWR SUBTRACT U	SUR SUBTRACT U

RX Format

xxxx	*Fixed-point halfword and branching* 0100xxxx	*Fixed-point fullword and logical* 0101xxxx	*Floating-point long* 0110xxxx	*Floating-point short* 0111xxxx
0000	STH STORE	ST STORE	STD STORE	STE STORE
0001	LA LOAD ADDRESS			
0010	STC STORE CHARACTER			
0011	IC INSERT CHARACTER			
0100	EX EXECUTE	N AND		
0101	BAL BRANCH AND LINK	CL COMPARE LOGICAL		
0110	BCT BRANCH ON COUNT	O OR		
0111	BC BRANCH/CONDITION	X EXCLUSIVE OR		
1000	LH LOAD	L LOAD	LD LOAD	LE LOAD
1001	CH COMPARE	C COMPARE	CD COMPARE	CE COMPARE
1010	AH ADD	A ADD	AD ADD N	AE ADD N
1011	SH SUBTRACT	S SUBTRACT	SD SUBTRACT N	SE SUBTRACT N
1100	MH MULTIPLY	M MULTIPLY	MD MULTIPLY	ME MULTIPLY
1101		D DIVIDE	DD DIVIDE	DE DIVIDE
1110	CVD CONVERT-DECIMAL	AL ADD LOGICAL	AW ADD U	AU ADD U
1111	CVB CONVERT-BINARY	SL SUBTRACT LOGICAL	SW SUBTRACT U	SU SUBTRACT U

RS, SI Format

xxxx	*Branching status switching and shifting* 1000xxxx	*Fixed-point logical and input/output* 1001xxxx	1010xxxx	1011xxxx
0000	SSM SET SYSTEM MASK	STM STORE MULTIPLE		
0001		TM TEST UNDER MASK		
0010	LPSW LOAD PSW	MVI MOVE		
0011	DIAGNOSE	TS TEST AND SET		
0100	WRD WRITE DIRECT	NI AND		
0101	RDD READ DIRECT	CLI COMPARE LOGICAL		
0110	BXH BRANCH/HIGH	OI OR		
0111	BXLE BRANCH/LOW-EQUAL	XI EXCLUSIVE OR		
1000	SRL SHIFT RIGHT SL	LM LOAD MULTIPLE		
1001	SLL SHIFT LEFT SL			
1010	SRA SHIFT RIGHT S			
1011	SLA SHIFT LEFT S			
1100	SRDL SHIFT RIGHT DL	SIO START I/O		
1101	SLDL SHIFT LEFT DL	TIO TEST I/O		
1110	SRDA SHIFT RIGHT D	HIO HALT I/O		
1111	SLDA SHIFT LEFT D	TCH TEST CHANNEL		

SS Format

xxxx	1100xxxx	*Logical* 1101xxxx	1110xxxx	*Decimal* 1111xxxx
0000				
0001		MVN MOVE NUMERIC		MVO MOVE WITH OFFSET
0010		MVC MOVE		PACK PACK
0011		MVZ MOVE ZONE		UNPK UNPACK
0100		NC AND		
0101		CLC COMPARE LOGICAL		
0110		OC OR		
0111		XC EXCLUSIVE OR		
1000				ZAP ZERO AND ADD
1001				CP COMPARE
1010				AP ADD
1011				SP SUBTRACT
1100		TR TRANSLATE		MP MULTIPLY
1101		TRT TRANSLATE AND TEST		DP DIVIDE
1110		ED EDIT		
1111		EDMK EDIT AND MARK		

NOTE: N = NORMALIZED DL = DOUBLE LOGICAL S = SINGLE
SL = SINGLE LOGICAL U = UNNORMALIZED D = DOUBLE

I/O operation, or by operator intervention at the I/O device, enable the CPU to provide appropriate programmed response to conditions as they occur in I/O devices or channels. Conditions responsible for I/O interruption requests are preserved in the I/O devices or channels until recognized by the CPU.

During execution of START I/O, a command can be rejected by a busy condition, program check, etc. Rejection is indicated in the condition code of the PSW, and additional detail on the conditions that precluded initiation of the I/O operation is provided in a CSW.

Manual control

The need for manual control is minimal because of the design of the system and supervisory program. A control panel provides the ability to reset the system; store and display information in main storage, in registers, and in the PSW; and load initial program information. After an input device is selected with the load unit switches, depressing a load key causes a read from the selected input device. The six words of information that are read into main storage provide the PSW and the CCW's required for subsequent operation.

Instruction set

The SYSTEM/360 instructions, classified by format and function, are displayed in Table 2. Operation codes and mnemonic abbreviations are also shown. With the previously described formats in mind, much of the generality provided by the system is apparent in this listing.

Chapter 44

The structure of SYSTEM/360[1]

Part II—System implementations

W. Y. Stevens

Summary The performance range desired of SYSTEM/360 is obtained by variations in the storage, processing, control, and channel functions of the several models. The systematic variations in speed, size, and degree of simultaneity that characterize the functional components and elements of each model are discussed.

A primary goal in the SYSTEM/360 design effort was a wide range of processing unit performances coupled with complete program compatibility. In keeping with this goal, the logical structure of the resultant system lends itself to a wide choice of components and techniques in the engineering of models for desired performance levels.

This paper discusses basic choices made in implementing six SYSTEM/360 models spanning a performance range of fifty to one. It should be emphasized that the problems of model implementation were studied throughout the design period, and many of the decisions concerning logical structure were influenced by difficulties anticipated or encountered in implementation.

Performance adjustment

The choices made in arriving at the desired performances fall into four areas:

Main storage

Central processing unit (CPU) registers and data paths

Sequence control

Input/output (I/O) channels

Each of the adjustable parameters of these areas can be subordinated, for present purposes, to one of three general factors: basic speed, size, and degree of simultaneity.

[1] *IBM Sys. J*, vol. 3, no. 2, 136–143, 1964.

Main storage

Storage speed and size

The interaction of the general factors is most obvious in the area of main storage. Here the basic speeds vary over a relatively small range: from a 2.5-μsec cycle for the Model 40 to a 1.0-μsec cycle for Models 62 and 70. However, in combination with the other two factors, a 32:1 range in overall storage data rate is obtained, as shown in Table 1.

Most important of the three factors is size. The width of main storage, i.e., the amount of data obtained with one storage access, ranges from one byte for the Model 30, two bytes for the Model 40, and four bytes for the Model 50, to 8 bytes for Models 60, 62, and 70.

Another size factor, less direct in its effect, is the total number of bytes in main storage, which can make a large difference in system throughput by reducing the number of references to external storage media. This number ranges from a minimum of 8192 bytes on Model 30 to a maximum of 524,288 bytes on Models 60, 62, and 70. An option of up to eight million more bytes of slower-speed, large-capacity core storage can further increase the throughput in some applications.

Interleaved storage

Simultaneity in the core storage of Models 60 and 70 is obtained by overlapping the cycles of two storage units. Addresses are staggered in the two units, and a series of requests for successive words activates the two units alternately, thus doubling the maximum rate. For increased system performance, this technique is less effective than doubling the basic speed of a single unit, since the access time to a single word is not improved, and successive references frequently occur to the same unit. This is illustrated by comparing the performances of Models 60 and 62, whose only difference is the choice between two overlapped 2.0-μsec storage units and one single 1.0-μsec storage unit, respectively. The performance of Model 62 is approximately 1.5 times that of Model 60.

Table 1 System/360 main storage characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60*	*Model 62*	*Model 70*
Cycle time (μsec)	2.0	2.5	2.0	2.0	1.0	1.0
Width (bytes)	1	2	4	8	8	8
Interleaved access	no	no	no	yes	no	yes
Maximum data rate (bytes/μsec)	0.5	0.8	2.0	8.0	8.0	16.0
Minimum storage size (bytes)	8,192	16,384	65,536	131,072	262,144	262,144
Maximum storage size (bytes)	65,536	262,144	262,144	524,288	524,288	524,288
Large capacity storage attachable	no	no	yes	yes	yes	yes

CPU registers and data paths

Circuit speed

SYSTEM/360 has three families of logic circuits, as shown in Table 2, each using the same solid-logic technology. One family, having a nominal delay of 30 nsec per logical stage or level, is used in the data paths of Models 30, 40, and 50. A second and faster family with a nominal delay of 10 nsec per level is used in Models 60 and 62. The fastest family, with a delay of 6 nsec, is used in Model 70.

The fundamental determinant of CPU speed is the time required to take data from the internal registers, process the data through the adder or other logical unit, and return the result to a register. This cycle time is determined by the delay per logical circuit level and the number of levels in the register-to-adder path, the adder, and the adder-to-register return path. The number of levels varies because of the trade-off that can usually be made between the number of circuit modules and the number of logical levels. Thus, the cycle time of the system varies from 1.0 μsec for Model 30 (with 30-nsec circuits, a relatively small number of modules, and more logic levels) and 0.5 μsec for Model 50 (also with 30-nsec circuits, but with more modules and fewer levels) to 0.2 μsec for Model 70 (with 6-nsec circuits).

Local storage

The speed of the CPU depends also on the speed of the general and floating-point registers. In Model 30, these registers are located in an extension to the main core storage and have a read-write

Table 2 System/360 CPU characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Circuit family: nominal delay per logic level (nsec)	30	30	30	10	6
Cycle time (μsec)	1.0	0.625	0.5	0.25	0.2
Location of general and floating registers	main core storage	local core storage	local core storage	local transistor storage	transistor registers
Width of general and floating register storage (bytes)	1	2	4	4	4 or 8
Speed of general and floating register storage (μsec)	2.0	1.25	0.5	0.25	
Width of main adder path (bits)	8	8	32	56	64
Width of auxiliary transfer path (bits)		16	8		
Widths of auxiliary adder paths (bits)				8	8, 8, and 24
Approximate number of bytes of register storage	12	15	30	50	100
Approximate number of bytes of working locations in local storage	45 (main storage)	48	60	4	
Relative computing speed	1	3.5	10	21/30	50

time of 2.0 μsec. In Model 40, the registers are located in a small core-storage unit, called *local storage,* with a read-write time of 1.25 μsec. Here, the operation of the local storage may be overlapped with main storage. In Model 50, the registers are in a local storage with a read-write time of only 0.5 μsec. In Model 60/62, the local storage has the logical characteristics of a core storage with nondestructive read-out; however, it is actually constructed as an array of registers using the 30-nsec family of logic circuits, and has a read-write time of 0.25 μsec. In Model 70, the general and floating-point registers are implemented with 6-nsec logic circuits and communicate directly with the adder and other data paths.

The two principal measures of size in the CPU are the width of the data paths and the number of bytes of high-speed working registers.

Data path organization

Model 30 has an 8-bit wide (plus parity) adder path, through which all data transfers are made, and approximately 12 bytes of working registers.

Model 40 also has an 8-bit wide adder path, but has an additional 16-bit wide data transfer path. Approximately 15 bytes of working registers are used, plus about 48 bytes of working locations in the local storage, exclusive of the general and floating-point registers.

Model 50 has a 32-bit wide adder path, an 8-bit wide data path used for handling individual bytes, approximately 30 bytes of working registers, plus about 60 bytes of working locations in the local storage.

Model 60/62 has a 56-bit wide main adder path, an 8-bit wide serial adder path, and approximately 50 bytes of working registers.

Model 70 has a 64-bit wide main adder, an 8-bit wide exponent adder, an 8-bit wide decimal adder, a 24-bit wide addressing adder, and several other data transfer paths, some of which have incrementing ability. The model has about 100 bytes of working registers plus the 96 bytes of floating point and general registers which, in Model 70, are directly associated with the data paths.

The models of SYSTEM/360 differ considerably in the number of relatively independent operations that can occur simultaneously in the CPU. Model 30, for example, operates serially: virtually all data transfers must pass through the adder, one byte at a time. Model 70, however, can have many operations taking place at the same time. The CPU of this model is divided into three units that operate somewhat independently. The instruction preparation unit fetches instructions from storage, prepares them by computing their effective addresses, and initiates the fetching of the required data. The execution unit performs the execution of the instruction prepared by the instruction unit. The third unit is a storage bus control which coordinates the various requests by the other units and by the channels for core-storage cycles. All three units normally operate simultaneously, and together provide a large degree of instruction overlap. Since each of the units contains a number of different data paths, several data transfers may be occurring on the same cycle in a single unit.

The operations of other SYSTEM/360 models fall between those mentioned. Model 50, for example, can have simultaneous data transfers through the main adder, through an auxiliary byte transfer path, and to or from local storage.

Sequence control

Complex instruction sequences

Since the SYSTEM/360 has an extensive instruction set, the CPU's must be capable of executing a large number of different sequences of basic operations. Furthermore, many instructions require sequences that are dependent on the data or addresses used. As shown in Table 3, these sequences of operations can be controlled by two methods; either by a conventional sequential logic circuit that uses the same types of circuit modules as used in the data paths or by a read-only storage device that contains a microprogram specifying the sequences to be performed for the different instructions.

Model 70 makes use of conventional sequential logic control mainly because of the high degree of simultaneity required. Also, a sufficiently fast read-only storage unit was not available at the time of development. The sequences to be performed in each of the Model 70 data paths have a considerable degree of independence. The read-only storage method of control does not easily lend itself to controlling these independent sequences, but is well adapted where the actions in each of the data paths are highly coordinated.

Read-only storage control

The read-only storage method of control is described elsewhere [Peacock, 19??]. This microprogram control, used in all but the fastest model of SYSTEM/360, is the only method known by which an extensive instruction set may be economically realized in a small system. This was demonstrated during the design of Model 60/62. Conventional logic control was originally planned for this model, but it became evident during the design period that too many circuit modules were required to implement the instruction set, even for this rather large system. Because a sufficiently fast read-only storage became available, it was adopted for sequence control at a substantial cost reduction.

Table 3 System/360 sequence control characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Type	read-only storage	read-only storage	read-only storage	read-only storage	sequential logic
Cycle time (μsec)	1.0	0.625	0.5	0.25	0.2
Width of read-only storage word (available bits)	60	60	90	100	
Number of read-only storage words available	4096	4096	2816	2816	
Number of gate-control fields in read-only storage word	9	10	15	16	

The three factors of speed, size, and simultaneity are applicable to the read-only storage controls of the various SYSTEM/360 models. The speed of the read-only storage units corresponds to the cycle time of the CPU, and hence varies from 1.0 μsec per access for Model 30 down to 0.25 μsec for Models 60 and 62.

The size of read-only storage can vary in two ways—in width (number of bits per word) and in number of words. Since the bits of a word are used to control gates in the data paths, the width of storage is indirectly related to the complexity of the data paths. The widths of the read-only storages in SYSTEM/360 range from 60 bits for Models 30 and 40 to 100 bits for Models 60 and 62. The number of words is affected by several factors. First, of course, is the number and complexity of the control sequences to be executed. This is the same for all models except that Model 60/62 read-only storage contains no sequences for channel functions. The number of words tends to be greater for the smaller models, since these models require more cycles to accomplish the same function. Partially offsetting this is the fact that the greater degree of simultaneity in the larger systems often prevents the sharing of microprogram sequences between similar functions.

SYSTEM/360 employs no read-only storage simultaneity in the sense that more than one access is in progress at a given time. However, a single read-only storage word simultaneously controls several independent actions. The number of different gate control fields in a word provides some measure of this simultaneity. Model 30 has 9 such fields. Model 60/62 has 16.

Input/output channels

Channel design

The SYSTEM/360 input/output channels may be considered from two viewpoints: the design of a channel itself, or the relationship of a channel to the whole system.

From the viewpoint of channel design, the raw speed of the components does not vary, since all channels use the 30-nsec family of circuits. However, the different channels do have access to different speeds of main storage and, in the three smaller models, different speeds of local storage.

The channels differ markedly in the amount of hardware devoted exclusively to channel use, as shown in Table 4. In the Model 30 multiplexor channel, this hardware amounts only to three 1-byte wide data paths, 11 latch bits for control, and a simple interface polling circuit. The channel used in Models 60, 62, and 70 contains about 300 bits of register storage, a 24-bit wide adder, and a complete set of sequential control circuits. The amount of hardware provided for other channels is somewhere in between these extremes.

The disparity in the amount of channel hardware reflects the extent to which the channels share CPU hardware in accomplishing their functions. Such sharing is done at the expense of increased interference with the CPU, of course. This interference ranges from complete lock-out of CPU operations at high data rates on some of the smaller models, to interference only in essential references to main storage by the channel in the large models.

Channel/system relationship

When the channels are viewed in their relationship to the whole system, the three factors of speed, size, and simultaneity take on a different aspect. The channel is viewed as a system component, and its effect on system throughput and other system capabilities is of concern. The speeds of the channels vary from a maximum rate of about 16 thousand bytes per second (byte interleaved mode) on the multiplexor channel of Model 30 to a maximum rate of about 1250 thousand bytes per second on the channels of Models 60, 62, and 70. The size of each of the channels is the same, in the sense that each handles an 8-bit byte at a time and each can connect to eight different control units. A slight size difference exists among multiplexor channels in terms of the maximum number of subchannels.

The degree of channel simultaneity differs considerably among the various models of SYSTEM/360. For example, operation of the Model 30 or 40 multiplexor channels in burst mode inhibits all

Table 4 System/360 channel characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Selector channels					
Maximum number attachable	2	2	3	6	6
Approximate maximum data rate on one channel in Kbyps†	250	400	800 (1250 on high speed)	1250	1250
Uses CPU data paths for:					
initiation and termination	yes	yes	yes	yes	yes
byte transfers	no	no	no	no	nc
storage word transfers	no	low speed only	yes	no	no
chaining	yes	yes	yes	no	no
CPU and I/O overlap possible	yes	yes	regular—yes high speed—no	yes	yes
Multiplexor channels					
Maximum number attachable	1	1	1	0	0
Minimum number of subchannels	32	16	64		
Maximum number of subchannels	96	128	256		
Maximum data rate in byte interleaved mode (Kbyps)	16	30	40		
Maximum data rate in burst mode (Kbyps)	200	200	200		
Uses CPU data paths for all functions	yes	yes	yes		
CPU and I/O overlap possible in byte mode	yes	yes	yes		
CPU and I/O overlap possible in burst mode	no	no	yes		

† Thousand bytes per second.

other activity on the system, as does operation of the special high-speed channel on Model 50. At the other extreme, as many as six selector channels can be operating concurrently with the CPU on Models 60, 62, or 70. A second type of simultaneity is present in the multiplexor channels available on Models 30, 40, and 50. When operating in byte interleaved mode, one of these channels can control a number of concurrently operating input/output devices, and the CPU can also continue operation.

Differences in application emphasis

The models of SYSTEM/360 differ not only in throughput but also in the relative speeds of the various operations. Some of these relative differences are simply a result of the design choices described in this paper, made to achieve the desired overall performance. The more basic differences in relative performance of the various operations, however, were intentional. These differences in emphasis suit each model to those applications expected to comprise its largest usage.

Thus the smallest system is particularly aimed at traditional commercial data processing applications. These are characterized by extensive input/output operations in relation to the internal processing, and by more character handling than arithmetic. The fast selector channels and character-oriented data paths of Model 30 result from this emphasis. But despite this emphasis, the general-purpose instruction set of SYSTEM/360 results in much better scientific application performance for Model 30 than for its comparable predecessors.

On the other hand, the large systems are expected to find particularly heavy use in scientific computation, where the emphasis is on rapid floating-point arithmetic. Thus Models 60, 62, and 70 contain registers and adders that can handle the full length of a long format floating-point operand, yet do character operations one byte at a time.

No particular emphasis on either commercial or scientific applications characterizes the intermediate models. However, Models 40 and 50 are intended to be particularly suitable for communication-oriented and real-time applications. For example, Model 50 includes a multiplexor channel, storage protection, and a timer as standard features, and also provides the ability to share main storages between two CPU's in a multiprocessing arrangement.

References

PeacA??

Chapter 40

The Structure of SYSTEM/360[1]

Part I—Outline of the Logical Structure

G. A. Blaauw / F. P. Brooks, Jr.

Summary A general introductory description of the logical structure of SYSTEM/360 is given. In addition, the functional units, the principal registers and formats, and the basic addressing and sequencing principles of the system are indicated.

In the SYSTEM/360 logical structure, processing efficiency and versatility are served by multiple accumulators, binary addressing, bit-manipulation operations, automatic indexing, fixed and variable field lengths, decimal and hexadecimal radices, and floating-point as well as fixed-point arithmetic. The provisions for program interruption, storage protection, and flexible CPU states contribute to effective operation. Base-register addressing, the standard interface between channels and input/output control units, and the machine-language compatibility among models contribute to flexible configurations and to orderly system expansion.

SYSTEM/360 is distinguished by a design orientation toward very large memories and a hierarchy of memory speeds, a broad spectrum of manipulative functions, and a uniform treatment of input/output functions that facilitates communication with a diversity of input/output devices. The overall structure lends itself to program-compatible embodiments over a wide range of performance levels.

The system, designed for operation with a supervisory program, has comprehensive facilities for storage protection, program relocation, nonstop operation, and program interruption. Privileged instructions associated with a supervisory operating state are included. The supervisory program schedules and governs the execution of multiple programs, handles exceptional conditions, and coordinates and issues input/output (I/O) instructions. Reliability is heightened by supplementing solid-state components with built-in checking and diagnostic aids. Interconnection facilities permit a wide variety of possibilities for multisystem operation.

The purpose of this discussion is to introduce the functional units of the system, as well as formats, codes, and conventions essential to characterization of the system.

[1]*IBM Sys. J*, vol. 3, no. 2, 1964, pp. 119–135.

Functional Structure

The SYSTEM/360 structure schematically outlined in Fig. 1 has seven announced embodiments. Six of these, namely, Models 30, 40, 50, 60, 62, and 70, will be treated here.[2] Where requisite I/O devices, optional features, and storage capacity are present, these six models are logically identical for valid programs that contain explicit time dependencies only. Hence, even though the allowable channels or storage capacity may vary from model to model (as discussed in Chap. 41), the logical structure can be discussed without reference to specific models.

Input/Output

Direct communication with a large number of low-speed terminals and other I/O devices is provided through a special *multiplexor* channel unit. Communication with high-speed I/O devices is accommodated by the *selector* channel units. Conceptually, the input/output system acts as a set of subchannels that operate concurrently with one another and the processing unit. Each subchannel, instructed by its own control-word sequence, can govern a data transfer operation between storage and a selected I/O device. A multiplexor channel can function either as one or as many subchannels; a selector channel always functions as a single subchannel. The control unit of each I/O device attaches to the channels via a standard mechanical-electrical-programming *interface.*

Processing

The processing unit has sixteen general purpose 32-bit registers used for addressing, indexing, and accumulating. Four 64-bit floating-point accumulators are optionally available. The inclusion of multiple registers permits effective use to be made of small high-speed memories. Four distinct types of processing are provided: logical manipulation of individual bits, character strings and fixed words; decimal arithmetic on digit strings; fixed-point binary arithmetic; and floating-point arithmetic. The processing unit, together with the central control function, will be referred to as the central processing unit (CPU). The basic registers and data paths of the CPU are shown in Fig. 2.

The CPU's of the various models yield a substantial range in performance. Relative to the smallest model (Model 30), the internal performance of the largest (Model 70) is approximately 50:1 for scientific computation and 15:1 for commercial data processing.

[2]A seventh embodiment, the Model 92, is not discussed in this paper. This model does not provide decimal data handling and has a few minor differences arising from its highly concurrent, speed-oriented organization. A paper on Model 92 is planned for future publication in the *IBM Systems Journal.*

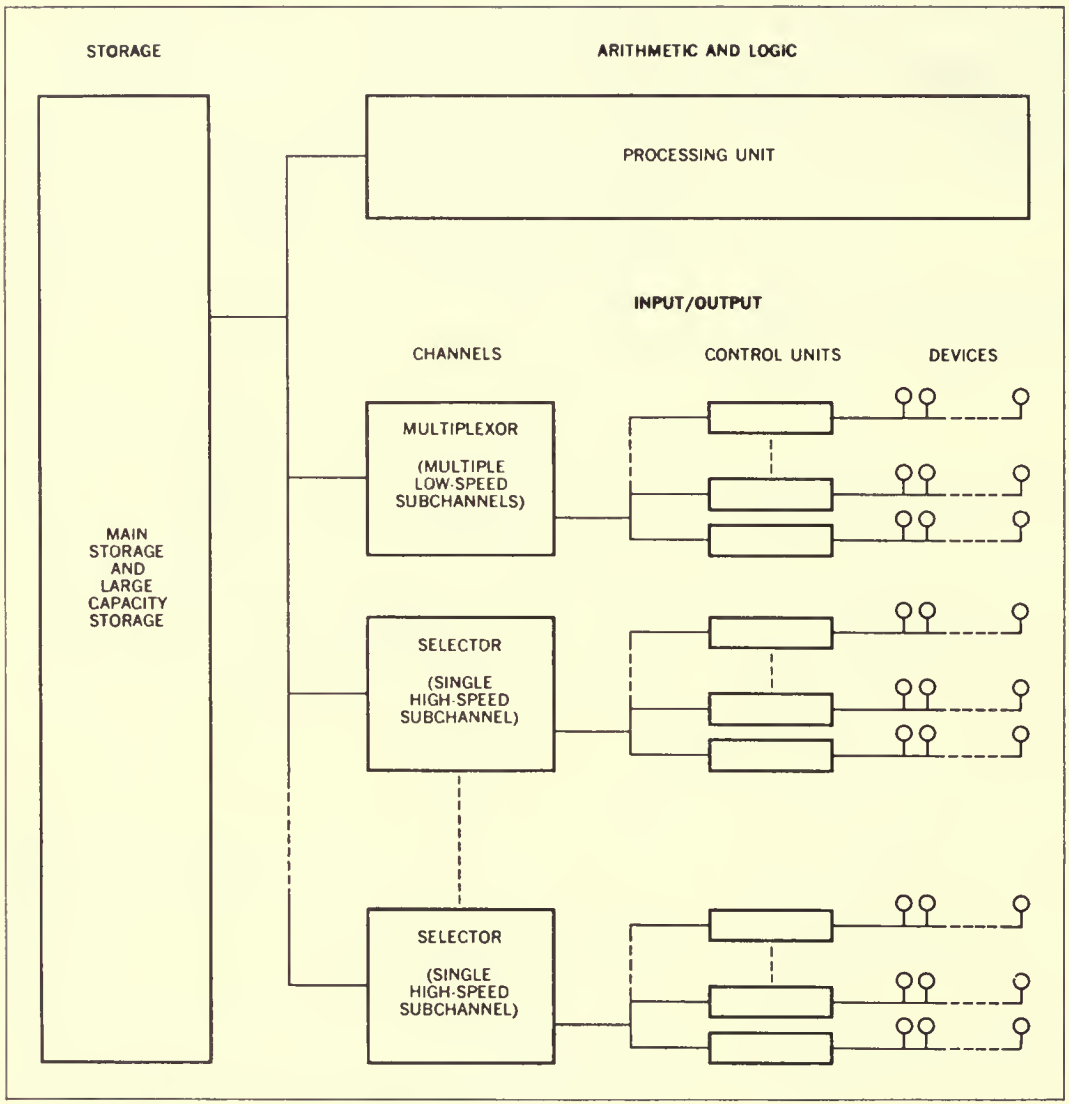


Fig. 1. Functional schematic of SYSTEM/360.

Control

Because of the extensive instruction set, SYSTEM/360 control is more elaborate than in conventional computers. Control functions include internal sequencing of each operation; sequencing from instruction to instruction (with branching and interruption); governing of many I/O transfers; and the monitoring, signaling, timing, and storage protection essential to total system operation. The control equipment is combined with a programmed supervisor, which coordinates and issues all I/O instructions, handles exceptional conditions, loads and relocates programs and data, manages storage, and supervises scheduling and execution of multiple programs. To a problem programmer, the supervisory program and the control equipment are indistinguishable.

The functional structure of SYSTEM/360, like that of most computers, is most concisely described by considering the data formats, the types of manipulations performed on them, and the instruction formats by which these manipulations are specified.

Information Formats

The several SYSTEM/360 data formats are shown in Fig. 3. An 8-bit unit of information is fundamental to most of the formats. A

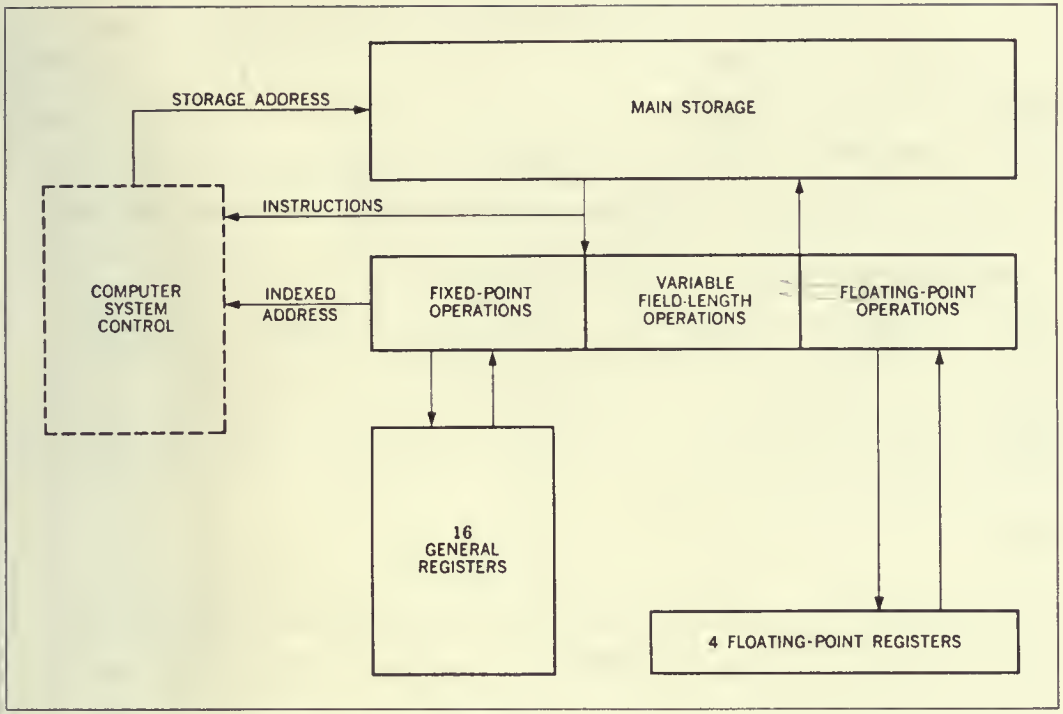


Fig. 2. Schematic of basic registers and data paths.

consecutive group of n such units constitutes a *field of length n*. Fixed-length fields of length one, two, four, and eight are termed *bytes*, *halfwords*, *words*, and *double words*, respectively. In many instructions, the operation code implies one of these four fields as the length of the operands. On the other hand, the length is explicit in an instruction that refers to operands of variable length.

The location of a stored field is specified by the address of the leftmost byte of the field. Variable-length fields may start on any byte location, but a fixed-length field of two, four, or eight bytes must have an address that is a multiple of 2, 4, or 8, respectively. Some of the various alignment possibilities are apparent from Fig. 3.

Storage addresses are represented by binary integers in the system. Storage capacities are always expressed as numbers of bytes.

Processing Operations

The SYSTEM/360 operations fall into four classes: fixed-point arithmetic, floating-point arithmetic, logical operations, and decimal arithmetic. These classes differ in the data formats used, the registers involved, the operations provided, and the way the field length is stated.

Fixed-Point Arithmetic

The basic arithmetic operand is the 32-bit fixed-point binary word. Halfword operands may be specified in most operations for the sake of improved speed or storage utilization. Some products and all dividends are 64 bits long, using an even-odd register pair.

Because the 32-bit words accommodate the 24-bit address, the entire fixed-point instruction set, including multiplication, division, shifting, and several logical operations, can be used in address computation. A two's complement notation is used for fixed-point operands.

Additions, subtractions, multiplications, divisions, and comparisons take one operand from a register and another from either a register or storage. Multiple-precision arithmetic is made convenient by the two's complement notation and by recognition of the carry from one word to another. A pair of conversion instructions, CONVERT TO BINARY and CONVERT TO DECIMAL, provide transition between decimal and binary radices without the use of tables. Multiple-register loading and storing instructions facilitate subroutine switching.

Floating-Point Arithmetic

Floating-point numbers may occur in either of two fixed-length formats—short or long. These formats differ only in the length of the fractions, as indicated in Fig. 3. The fraction of a floating-point

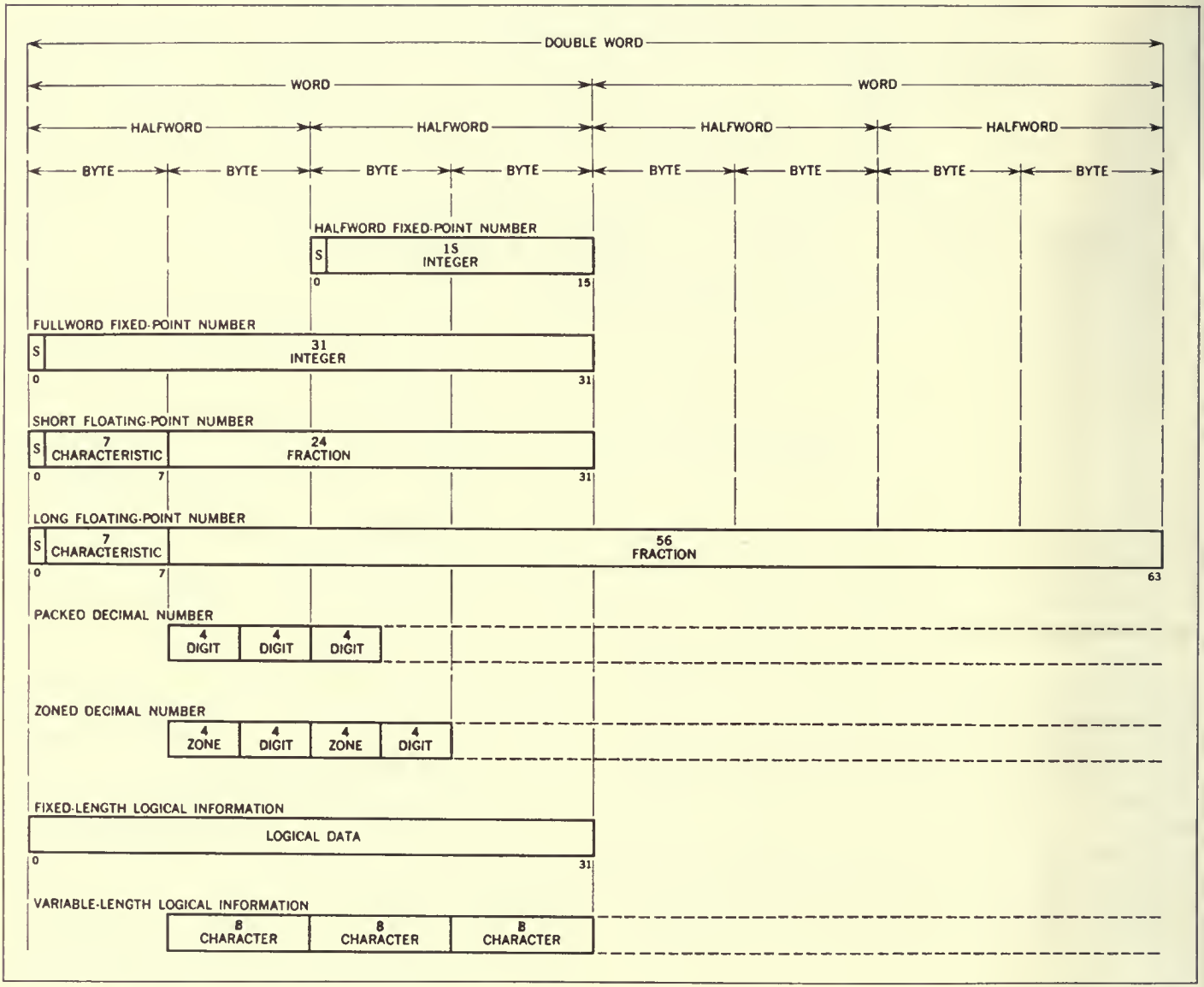


Fig. 3. The data formats.

number is expressed in 4-bit hexadecimal (base 16) digits. In the short format, the fraction has six hexadecimal digits; in the long format, the fraction has 14 hexadecimal digits. The short length is equivalent to seven decimal places of precision. The long length gives up to 17 decimal places of precision, thus eliminating most requirements for double-precision arithmetic.

The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1 through 7 of both formats, is used to indicate this power. The characteristic is treated as an excess 64 number with a range from −64 through +63, and permits representation of decimal numbers with magnitudes in the range of 10^{-78} to 10^{75}.

Bit position 0 in either format is the fraction sign, S. The fraction of negative numbers is carried in true form.

Floating-point operations are performed with one operand from a register and another from either a register or storage. The

result, placed in a register, is generally of the same length as the operands.

Logical Operations

Operations for comparison, translation, editing, bit testing, and bit setting are provided for processing logical fields of fixed and variable lengths. Fixed-length logical operands, which consist of one, four, or eight bytes, are processed from the general registers. Logical operations can also be performed on fields of up to 256 bytes, in which case the fields are processed from left to right, one byte at a time. Moreover, two powerful scanning instructions permit byte-by-byte translation and testing via tables. An important special case of variable-length logical operations is the one-byte field, whose individual bits can be tested, set, reset, and inverted as specified by an 8-bit mask in the instruction.

Character Codes

Any 8-bit character set can be processed, although certain restrictions are assumed in the decimal arithmetic and editing operations. However, all character-set-sensitive, I/O equipment assumes either the Extended Binary-Coded-Decimal Interchange Code (EBCDIC) of Fig. 4 or the code of Fig. 5, which is an eight-bit extension of a seven-bit code proposed by the International Standards Organization.

Decimal Arithmetic

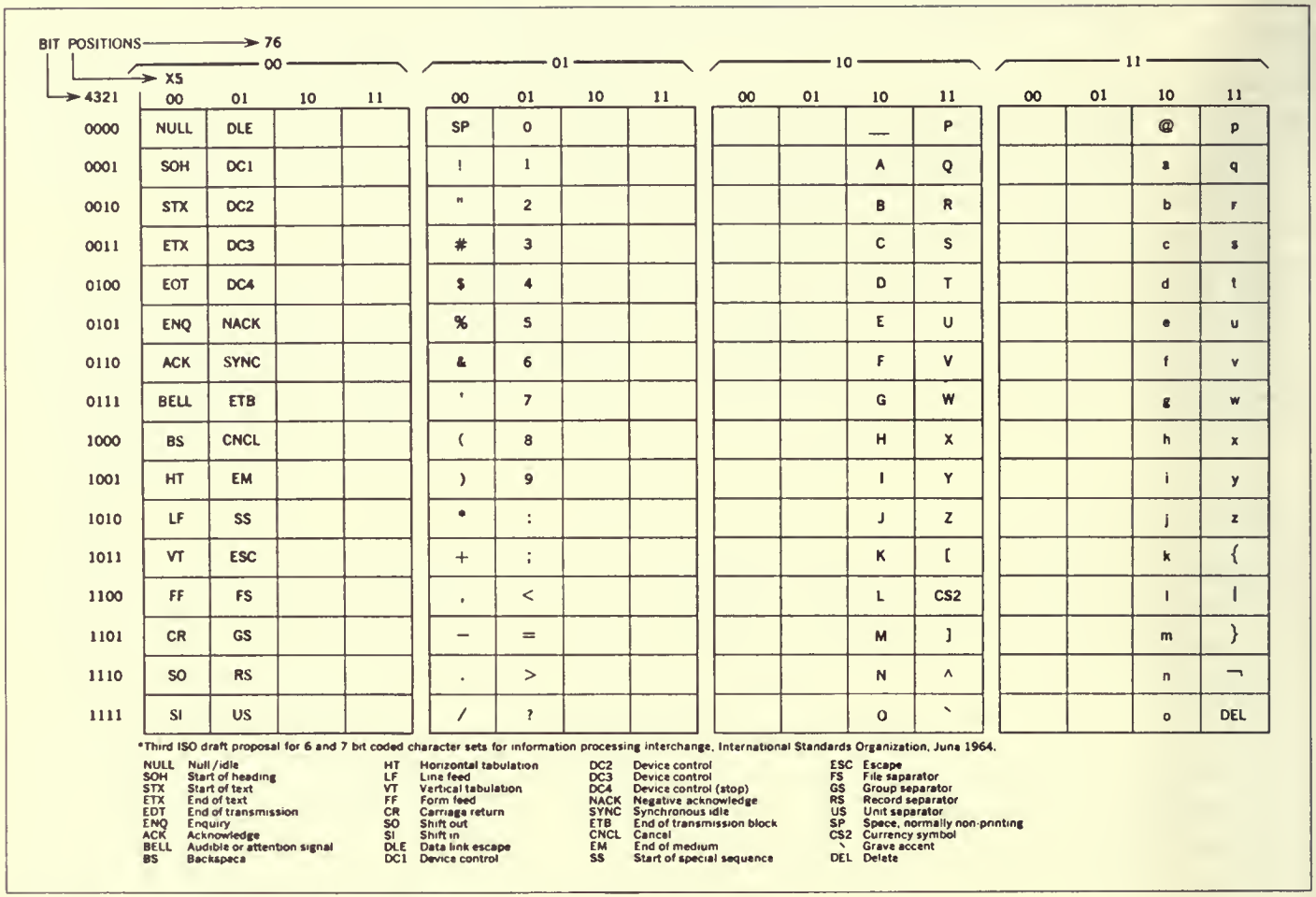
Decimal arithmetic can improve performance for processes requiring few computational steps per datum between the source input and the output. In these cases, where radix conversion from decimal to binary and back to decimal is not justified, the use of registers for intermediate results usually yields no advantage over storage-to-storage processing. Hence, decimal arithmetic is pro-

BIT POSITIONS → 01 / 23 / 4567

	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NULL				SP	&	–									0
0001							/		a	j			A	J		1
0010									b	k	s		B	K	S	2
0011									c	l	t		C	L	T	3
0100	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	EOB	UC					f	o	w		F	O	W	6
0111	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
1000									h	q	y		H	Q	Y	8
1001									i	r	z		I	R	Z	9
1010			SM		¢	!		:								
1011					.	$	,	#								
1100					<	*	%	@								
1101					(	)	_	'								
1110					+	;	>	=								
1111					|	¬	?	"								

PF Punch off
HT Horizontal tab
LC Lower case
DEL Delete
RES Restore
NL New line
BS Backspace
IL Idle
BYP Bypass
LF Line feed
EOB End of block
PRE Prefix
SM Set mode
PN Punch on
RS Reader stop
UC Upper case
EOT End of transmission
SP Space

Fig. 4. Extended binary-coded-decimal interchange code.



BIT POSITIONS → 76 / X5 / 4321	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NULL	DLE			SP	0					—	P			@	p
0001	SOH	DC1			!	1					A	Q			a	q
0010	STX	DC2			"	2					B	R			b	r
0011	ETX	DC3			#	3					C	S			c	s
0100	EOT	DC4			$	4					D	T			d	t
0101	ENQ	NACK			%	5					E	U			e	u
0110	ACK	SYNC			&	6					F	V			f	v
0111	BELL	ETB			'	7					G	W			g	w
1000	BS	CNCL			(	8					H	X			h	x
1001	HT	EM			)	9					I	Y			i	y
1010	LF	SS			*	:					J	Z			j	z
1011	VT	ESC			+	;					K	[			k	{
1100	FF	FS			,	<					L	CS2			l	|
1101	CR	GS			-	=					M	]			m	}
1110	SO	RS			.	>					N	^			n	¬
1111	SI	US			/	?					O	`			o	DEL

*Third ISO draft proposal for 6 and 7 bit coded character sets for information processing interchange, International Standards Organization, June 1964.

NULL	Null/idle	HT	Horizontal tabulation	DC2	Device control	ESC	Escape
SOH	Start of heading	LF	Line feed	DC3	Device control	FS	File separator
STX	Start of text	VT	Vertical tabulation	DC4	Device control (stop)	GS	Group separator
ETX	End of text	FF	Form feed	NACK	Negative acknowledge	RS	Record separator
EDT	End of transmission	CR	Carriage return	SYNC	Synchronous idle	US	Unit separator
ENQ	Enquiry	SO	Shift out	ETB	End of transmission block	SP	Space, normally non-printing
ACK	Acknowledge	SI	Shift in	CNCL	Cancel	CS2	Currency symbol
BELL	Audible or attention signal	DLE	Data link escape	EM	End of medium	`	Grave accent
BS	Backspace	DC1	Device control	SS	Start of special sequence	DEL	Delete

Fig. 5. Eight-bit representation for proposed international code.

vided in SYSTEM/360 with operands as well as results located in storage, as in the IBM 1400 series. Decimal arithmetic includes addition, subtraction, multiplication, division, and comparison.

The decimal digits 0 through 9 are represented in the 4-bit binary-coded-decimal form by 0000 through 1001, respectively. The patterns 1010 through 1111 are not valid as digits and are interpreted as sign codes: 1011 and 1101 represent a minus, the other four a plus. The sign patterns generated in decimal arithmetic depend upon the character set preferred. For EBCDIC, the patterns are 1100 and 1101; for the code of Fig. 5, they are 1010 and 1011. The choice between the two codes is determined by a mode bit.

Decimal digits, packed two to a byte, appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the rightmost four bits of the low-order byte. Operand fields can be located on any byte boundary, and can have lengths up to 31 digits and sign. Operands participating in an operation have independent lengths. Negative numbers are carried in true form. Instructions are provided for packing and unpacking decimal numbers. Packing of digits leads to efficient use of storage, increased arithmetic performance, and improved rates of data transmission. For purely decimal fields, for example, a 90,000-byte/second tape drive reads and writes 180,000 digits/second.

Instruction Formats

Instruction formats contain one, two, or three halfwords, depending upon the number of storage addresses necessary for the operation. If no storage address is required of an instruction, one halfword suffices. A two-halfword instruction specifies one address; a three-halfword instruction specifies two addresses. All instructions must be aligned on halfword boundaries.

The five basic instruction formats, denoted by the format

mnemonics RR, RX, RS, SI, and SS are shown in Fig. 6. RR denotes a register-to-register operation, RX a register and indexed-storage operation, RS a register and storage operation, SI a storage and immediate-operand operation, and SS a storage-to-storage operation.

In each format, the first instruction halfword consists of two parts. The first byte contains the operation code. The length and format of an instruction are indicated by the first two bits of the operation code.

The second byte is used either as two 4-bit fields or as a single 8-bit field. This byte is specified from among the following:

Four-bit operand register designator (R)

Four-bit index register designator (X)

Four-bit mask (M)

Four-bit field length specification (L)

Eight-bit field length specification

Eight-bit byte of immediate data (I)

The second and third halfwords each specify a 4-bit base register designator (B), followed by a 12-bit displacement (D).

Addressing

An effective storage address E is a 24-bit binary integer given, in the typical case, by

$$E = B + X + D$$

where B and X are 24-bit integers from general registers

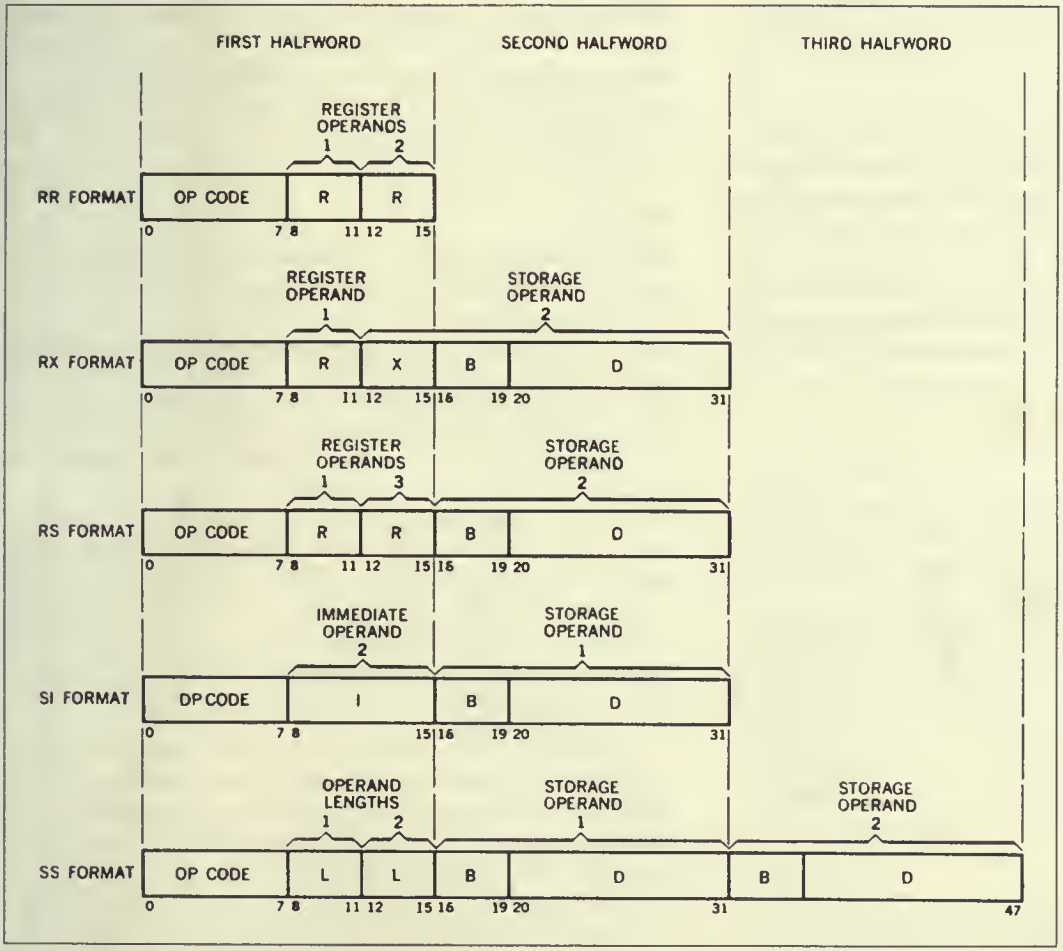


Fig. 6. Five basic instruction formats.

identified by fields B and X, respectively, and the displacement D is a 12-bit integer contained in every instruction that references storage.

The base *B* can be used for static relocation of programs and data. In record processing, the base can identify a record; in array calculations, it can specify the location of an array. The index *X* can provide the relative address of an element within an array. Together, *B* and *X* permit double indexing in array processing.

The displacement provides for relative addressing of up to 4095 bytes beyond the element or base address. In array calculations, the displacement can identify one of many items associated with an element. Thus, multiple arrays whose indices move together are best stored in an interleaved manner. In the processing of records, the displacement can identify items within a record.

In forming an effective address, the base and index are treated as unsigned 24-bit positive binary integers and the displacement as a 12-bit positive binary integer. The three are added as 24-bit binary numbers, ignoring overflow. Since every address is formed with the aid of a base, programs can be readily and generally relocated by changing the contents of base registers.

A zero base or index designator implies that a zero quantity must be used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance. Initialization, modification, and testing of bases and indices can be carried out by fixed-point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, or BRANCH ON INDEX instructions. LOAD EFFECTIVE ADDRESS provides not only a convenient housekeeping operation, but also, when the same register is specified for result and operand, an immediate register-incrementing operation.

Sequencing

Normally, the CPU takes instructions in sequence. After an instruction is fetched from a location specified by the instruction counter, the instruction counter is increased by the number of bytes in the instruction.

Conceptually, all halfwords of an instruction are fetched from storage after the preceding operation is completed and before execution of the current operation, even though physical storage word size and overlap of instruction execution with storage access may cause the actual instruction fetching to be different. Thus, an instruction can be modified by the instruction that immediately precedes it in the instruction stream, and cannot effectively modify itself during execution.

Branching

Most branching is accomplished by a single BRANCH ON CONDITION operation that inspects a 2-bit *condition register*. Many of the arithmetic, logical, and I/O operations indicate an outcome by setting the condition register to one of its four possible states. Subsequently a conditional branch can select one of the states as a criterion for branching. For example, the condition code reflects such conditions as non-zero result, first operand high, operands equal, overflow, channel busy, zero, etc. Once set, the condition register remains unchanged until modified by an instruction execution that reflects a different condition code.

The outcome of address arithmetic and counting operations can be tested by a conditional branch to effect loop control. Two instructions, BRANCH ON COUNT and BRANCH ON INDEX, provide for one-instruction execution of the most common arithmetic-test combinations.

Program Status Word

A program status word (PSW), a double word having the format shown in Fig. 7, contains information required for proper execution of a given program. A PSW includes an instruction address, condition code, and several mask and mode fields. The active or controlling PSW is called the *current* PSW. By storing the current PSW during an interruption, the status of the interrupted program is preserved.

Interruption

Five classes of interruption conditions are distinguished: input/output, program, supervisor call, external, and machine check.

For each class, two PSW's, called *old* and *new*, are maintained in the main-storage locations shown in Table 1. An interruption in a given class stores the current PSW as an old PSW and then takes the corresponding new PSW as the current PSW. If, at the conclusion of the interruption routine, old and current PSW's are interchanged, the system can be restored to its prior state and the interrupted routine can be continued.

The system mask, program mask, and machine-check mask bits in the PSW may be used to control certain interruptions. When masked off, some interruptions remain pending while others are merely ignored. The system mask can keep I/O and external interruptions pending, the program mask can cause four of the 15 program interruptions to be ignored, and the machine-check mask can cause machine-check interruptions to be ignored. Other interruptions cannot be masked off.

Appropriate CPU response to a special condition in the channels and I/O units is facilitated by an I/O *interruption*. The addresses of the channel and I/O unit involved are recorded in the old PSW. Related information is preserved in a channel status word that is stored as a result of the interruption.

Unusual conditions encountered in a program create *program interruptions*. Eight of the fifteen possible conditions involve overflows, improper divides, lost significance, and exponent underflow. The remaining seven deal with improper addresses,

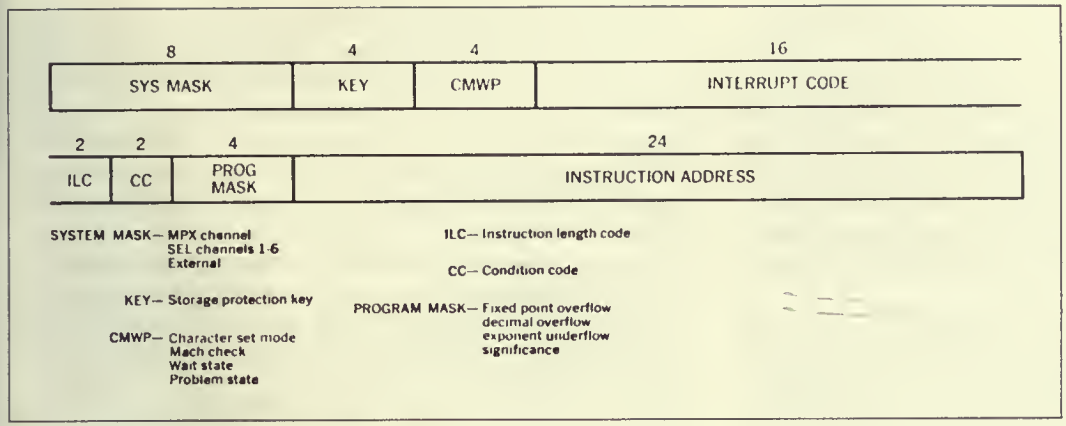


Fig. 7. Program status word format.

attempted execution of privileged instructions, and similar conditions.

A *supervisor-call interruption* results from execution of the instruction SUPERVISOR CALL. Eight bits from the instruction format are placed in the interruption code of the old PSW, permitting a message to be associated with the interruption. SUPERVISOR CALL permits a problem program to switch CPU control back to the supervisor.

Through an *external interruption*, a CPU can respond to signals from the interruption key on the system control panel, the timer, other CPU's, or special devices. The source of the interruption is identified by an interruption code in bits 24 through 31 of the PSW.

The occurrence of a machine check (if not masked off) terminates the current instruction, initiates a diagnostic procedure, and subsequently effects a *machine-check interruption*. A machine check is occasioned only by a hardware malfunction; it cannot be caused by invalid data or instructions.

Table 1 Permanent Storage Assignments

Address	*Byte length*	*Purpose*
0	8	Initial program loading PSW
8	8	Initial program loading CCW 1
16	8	Initial program loading CCW 2
24	8	External old PSW
32	8	Supervisor call old PSW
40	8	Program old PSW
48	8	Machine check old PSW
56	8	Input/output old PSW
64	8	Channel status word
72	4	Channel address word
76	4	Unused
80	4	Timer
84	4	Unused
88	8	External new PSW
96	8	Supervisor call new PSW
104	8	Program new PSW
112	8	Machine check new PSW
120	8	Input/output new PSW
128		Diagnostic scan-out area†

†The size of the diagnostic scan-out area is configuration dependent.

Interrupt Priority

Interruption requests are honored between instruction executions. When several requests occur during execution of an instruction, they are honored in the following order: (1) machine check, (2) program or supervisor call, (3) external, and (4) input/output. Because the program and supervisor-call interruptions are mutually exclusive, they cannot occur at the same time.

If a machine-check interruption occurs, no other interruptions can be taken until this interruption is fully processed. Otherwise, the execution of the CPU program is delayed while PSW's are appropriately stored and fetched for each interruption. When the last interruption request has been honored, instruction execution is resumed with the PSW last fetched. An interruption subroutine is then serviced for each interruption in the order (1) input/output, (2) external, and (3) program or supervisor call.

Program Status

Overall CPU status is determined by four alternatives: (1) *stopped* versus *operating* state, (2) *running* versus *waiting* state, (3) *masked* versus *interruptable* state, and (4) *supervisor* versus *problem* state.

In the stopped state, which is entered and left by manual procedure, instructions are not executed, interruptions are not accepted, and the timer is not updated. In the operating

state, the CPU is capable of executing instructions and of being interrupted.

In the running state, instruction fetching and execution proceeds in the normal manner. The wait state is typically entered by the program to await an interruption, for example, an I/O interruption or operator intervention from the console. In the wait state, no instructions are processed, the timer is updated, and I/O and external interruptions are accepted unless masked. Running versus waiting is determined by the setting of a bit in the current PSW.

The CPU may be interruptable or masked for the system, program, and machine interruptions. When the CPU is interruptable for a class of interruptions, these interruptions are accepted. When the CPU is masked, the system interruptions remain pending, but the program and machine-check interruptions are ignored. The interruptable states of the CPU are changed by altering mask bits in the current PSW.

In the problem state, processing instructions are valid, but all I/O instructions and a group of control instructions are invalid. In the supervisor state, all instructions are valid. The choice of problem or supervisor state is determined by a bit in the PSW.

Supervisory Facilities

Timer

A timer word in main storage location 80 is counted down at a rate of 50 or 60 cycles per second, depending on power line frequency. The word is treated as a signed integer according to the rules of fixed-point arithmetic. An external interrupt occurs when the value of the timer word goes from positive to negative. The full cycle time of the timer is 15.5 hours.

As an interval timer, the timer may be used to measure elapsed time over relatively short intervals. The timer can be set by a supervisory-mode program to any value at any time.

Direct Control

Two instructions, READ DIRECT and WRITE DIRECT, provide for the transfer of a single byte of information between an external device and the main storage of the system. These instructions are intended for use in synchronizing CPU's and special external devices.

Storage Protection

For protection purposes, main storage is divided into blocks of 2,048 bytes each. A four-bit *storage key* is associated with each block. When a store operation is attempted by an instruction, the *protection key* of the current PSW is compared with the storage key of the affected block. When storing is specified by a channel operation, a protection key supplied by the channel is used as the comparand. The keys are said to *match* if equal or if either is zero. A storage key is not part of addressable storage, and can be changed only by privileged instructions. The protection key of the CPU program is held in the current PSW. The protection key of a channel is recorded in a status word that is associated with the channel operation.

When a CPU operation causes a protection mismatch, its execution is suppressed or terminated, and the program execution is altered by an interruption. The protected storage location always remains unchanged. Similarly, protection mismatch due to an I/O operation terminates data transmission in such a way that the protected storage location remains unchanged.

Multisystem Operation

Communication between CPU's is made possible by shared control units, interconnected channels, or shared storage. Multisystem operation is supported by provisions for automatic relocation, indication of malfunctions, and CPU initialization.

Automatic relocation applies to the first 4,096 bytes of storage, an area that contains all permanent storage assignments and usually has special significance for supervisory programs. The relocation is accomplished by inserting a 12-bit prefix in each address whose high-order 12 bits are zero. Two manually set prefixes permit the use of an alternate area when storage malfunction occurs; the choice between prefixes is preserved in a trigger that is set during initial program loading.

To alert one CPU to the possible malfunction of another, a machine-check signal from a given CPU can serve as an external interruption to another CPU. By another special provision, initial program loading of a given CPU can be initiated by a signal from another CPU.

Input/Output

Devices and Control Units

Input/output devices include card equipment, magnetic tape units, disk storage, drum storage, typewriter-keyboard devices, printers, teleprocessing devices, and process control equipment. The I/O devices are regulated by control units, which provide the electrical, logical, and buffering capabilities necessary for I/O device operation. From the programming point of view, most control-unit and I/O device functions are indistinguishable. Sometimes the control unit is housed with an I/O device, as in the case of the printer.

A control unit functions only with those I/O devices for which it is designed, but all control units respond to a standard set of

signals from the channel. This control-unit-to-channel connection, called the I/O *interface*, enables the CPU to handle all I/O operations with only four instructions.

I/O Instructions

Input/output instructions can be executed only while the CPU is in the supervisor state. The four I/O instructions are START I/O, HALT I/O, TEST CHANNEL, and TEST I/O.

START I/O initiates an I/O operation; its address field specifies a channel and an I/O device. If the channel facilities are free, the instruction is accepted and the CPU continues its program. The channel independently selects the specified I/O device. HALT I/O terminates a channel operation. TEST CHANNEL sets the condition code in the PSW to indicate the state of the channel addressed by the instruction. The code then indicates one of the following conditions: channel available, interruption condition in channel, channel working, or channel not operational. TEST I/O sets the PSW condition code to indicate the state of the addressed channel, subchannel, and I/O device.

Channels

Channels provide the data path and control for I/O devices as they communicate with main storage. In the multiplexor channel, the single data path can be time-shared by several low-speed devices (card readers, punches, printers, terminals, etc.) and the channel has the functional character of many subchannels, each of which services one I/O device at a time. On the other hand, the selector channel, which is designed for high-speed devices, has the functional character of a single subchannel. All subchannels respond to the same I/O instructions. Each can fetch its own control word sequence, govern the transfer of data and control signals, count record lengths, and interrupt the CPU on exceptions.

Two modes of operation, *burst* and *multiplex*, are provided for multiplexor channels. In burst mode, the channel facilities are monopolized for the duration of data transfer to or from a particular I/O device. The selector channel functions only in the burst mode. In multiplex mode, the multiplexor channel sustains several simultaneous I/O operations: bytes of data are interleaved and then routed between selection I/O devices and desired locations in main storage.

At the conclusion of an operation launched by START I/O or TEST I/O, an I/O interruption occurs. At this time a channel status word (CSW) is stored in location 64. Figure 8 shows the CSW format. The CSW provides information about the termination of the I/O operation.

Successful execution of START I/O causes the channel to fetch a channel address word from main-storage location 72. This word specifies the storage-protection key that governs the I/O operation, as well as the location of the first eight bytes of information that the channel fetches from main storage. These 64 bits comprise a channel command word (CCW). Figure 9 shows the CCW format.

Channel Program

One or more CCW's make up the channel program that directs channel operations. Each CCW points to the next one to be fetched, except for the last in the chain which so identifies itself.

Six channel commands are provided: read, write, read backward, sense, transfer in channel, and control. The read command defines an area in main storage and causes a read operation from the selected I/O device. The write command causes data to be written by the selected device. The read-backward command is akin to the read command, but the external medium is moved in the opposite direction and bytes read backward are placed in descending main storage locations.

The control command contains information, called an *order*, that is used to control the selected I/O device. Orders, peculiar to the particular I/O device in use, can specify such functions as rewinding a tape unit, searching for a particular track in disk

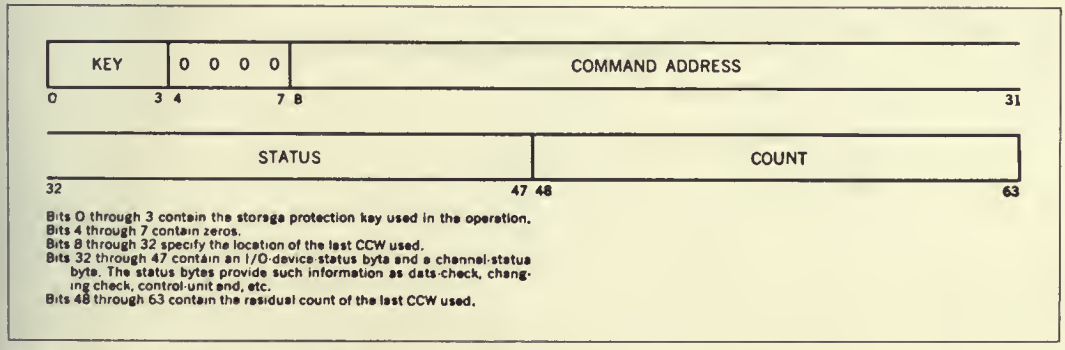


Fig. 8. Channel status word format.

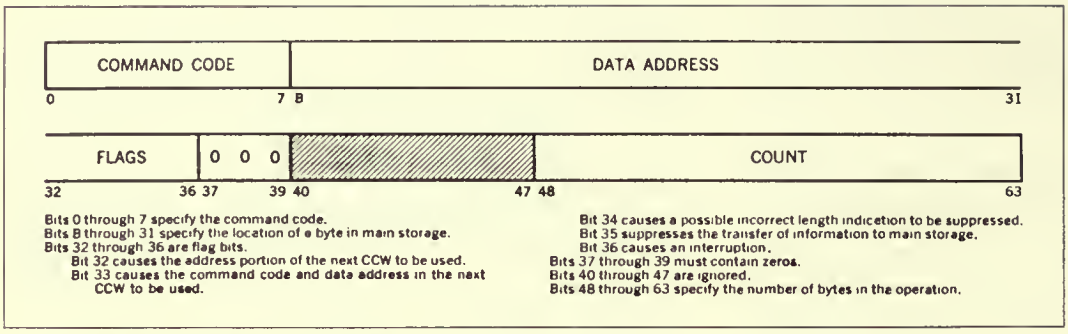


Fig. 9. Channel command word format.

storage, or line skipping on a printer. In a functional sense, the CPU executes I/O instructions, the channels execute commands, and the control units and devices execute orders.

The sense command specifies a main storage location and transfers one or more bytes of status information from the selected control unit. It provides details concerning the selected I/O device, such as a stacker-full condition of a card reader or a file-protected condition of a magnetic-tape reel.

A channel program normally obtains CCW's from a consecutive string of storage locations. The string can be broken by a transfer-in-channel command that specifies the location of the next CCW to be used by the channel. External documents, such as punched cards or magnetic tape, may carry CCW's that can be used by the channel to govern the reading of the documents.

The input/output interruptions caused by termination of an I/O operation, or by operator intervention at the I/O device, enable the CPU to provide appropriate programmed response to conditions as they occur in I/O devices or channels. Conditions responsible for I/O interruption requests are preserved in the I/O devices or channels until recognized by the CPU.

During execution of START I/O, a command can be rejected by a busy condition, program check, etc. Rejection is indicated in the condition code of the PSW, and additional detail on the conditions that precluded initiation of the I/O operation is provided in a CSW.

Manual Control

The need for manual control is minimal because of the design of the system and supervisory program. A control panel provides the ability to reset the system; store and display information in main storage, in registers, and in the PSW; and load initial program information. After an input device is selected with the load unit switches, depressing a load key causes a read from the selected input device. The six words of information that are read into main storage provide the PSW and the CCW's required for subsequent operation.

Instruction Set

The SYSTEM/360 instructions, classified by format and function, are displayed in Table 2. Operation codes and mnemonic abbreviations are also shown. With the previously described formats in mind, much of the generality provided by the system is apparent in this listing.

Table 2 (*opposite*) System/360 instructions

RR Format

xxxx	*Branching and status switching* 0000xxxx	*Fixed-point fullword and logical* 0001xxxx	*Floating-point long* 0010xxxx	*Floating-point short* 0011xxxx
0000		LPR LOAD POSITIVE	LPDR LOAD POSITIVE	LPER LOAD POSITIVE
0001		LNR LOAD NEGATIVE	LNDR LOAD NEGATIVE	LNER LOAD NEGATIVE
0010		LTR LOAD AND TEST	LTDR LOAD AND TEST	LTER LOAD AND TEST
0011		LCR LOAD COMPLEMENT	LCDR LOAD COMPLEMENT	LCER LOAD COMPLEMENT
0100	SPM SET PROGRAM MASK	NR AND	HDR HALVE	HER HALVE
0101	BALR BRANCH AND LINK	CLR COMPARE LOGICAL		
0110	BCTR BRANCH ON COUNT	OR OR		
0111	BCR BRANCH/CONDITION	XR EXCLUSIVE OR		
1000	SSK SET KEY	LR LOAD	LDR LOAD	LER LOAD
1001	ISK INSERT KEY	CR COMPARE	CDR COMPARE	CER COMPARE
1010	SVC SUPERVISOR CALL	AR ADD	ADR ADD N	ALR ADD N
1011		SR SUBTRACT	SDR SUBTRACT N	SER SUBTRACT N
1100		MR MULTIPLY	MDR MULTIPLY	MER MULTIPLY
1101		DR DIVIDE	DDR DIVIDE	DER DIVIDE
1110		ALR ADD LOGICAL	AWR ADD U	AUR ADD U
1111		SLR SUBTRACT LOGICAL	SWR SUBTRACT U	SUR SUBTRACT U

RX Format

xxxx	*Fixed-point halfword and branching* 0100xxxx	*Fixed-point fullword and logical* 0101xxxx	*Floating-point long* 0110xxxx	*Floating-point short* 0111xxxx
0000	STH STORE	ST STORE	STD STORE	STE STORE
0001	LA LOAD ADDRESS			
0010	STC STORE CHARACTER			
0011	IC INSERT CHARACTER			
0100	EX EXECUTE	N AND		
0101	BAL BRANCH AND LINK	CL COMPARE LOGICAL		
0110	BCT BRANCH ON COUNT	O OR		
0111	BC BRANCH/CONDITION	X EXCLUSIVE OR		
1000	LH LOAD	L LOAD	LD LOAD	LE LOAD
1001	CH COMPARE	C COMPARE	CD COMPARE	CE COMPARE
1010	AH ADD	A ADD	AD ADD N	AE ADD N
1011	SH SUBTRACT	S SUBTRACT	SD SUBTRACT N	SE SUBTRACT N
1100	MH MULTIPLY	M MULTIPLY	MD MULTIPLY	ME MULTIPLY
1101		D DIVIDE	DD DIVIDE	DE DIVIDE
1110	CVD CONVERT-DECIMAL	AL ADD LOGICAL	AW ADD U	AU ADD U
1111	CVB CONVERT-BINARY	SL SUBTRACT LOGICAL	SW SUBTRACT U	SU SUBTRACT U

RS, SI Format

xxxx	*Branching status switching and shifting* 1000xxxx	*Fixed-point logical and input/output* 1001xxxx	1010xxxx	1011xxxx
0000	SSM SET SYSTEM MASK	STM STORE MULTIPLE		
0001		TM TEST UNDER MASK		
0010	LPSW LOAD PSW	MVI MOVE		
0011	DIAGNOSE	TS TEST AND SET		
0100	WRD WRITE DIRECT	NI AND		
0101	RDD READ DIRECT	CLI COMPARE LOGICAL		
0110	BXH BRANCH/HIGH	OI OR		
0111	BXLE BRANCH/LOW-EQUAL	XI EXCLUSIVE OR		
1000	SRL SHIFT RIGHT SL	LM LOAD MULTIPLE		
1001	SLL SHIFT LEFT SL			
1010	SRA SHIFT RIGHT S			
1011	SLA SHIFT LEFT S			
1100	SRDL SHIFT RIGHT DL	SIO START I/O		
1101	SLDL SHIFT LEFT DL	TIO TEST I/O		
1110	SRDA SHIFT RIGHT D	HIO HALT I/O		
1111	SLDA SHIFT LEFT D	TCH TEST CHANNEL		

SS Format

xxxx	1100xxxx	*Logical* 1101xxxx	1110xxxx	*Decimal* 1111xxxx
0000				
0001		MVN MOVE NUMERIC		MVO MOVE WITH OFFSET
0010		MVC MOVE		PACK PACK
0011		MVZ MOVE ZONE		UNPK UNPACK
0100		NC AND		
0101		CLC COMPARE LOGICAL		
0110		OC OR		
0111		XC EXCLUSIVE OR		
1000				ZAP ZERO AND ADD
1001				CP COMPARE
1010				AP ADD
1011				SP SUBTRACT
1100		TR TRANSLATE		MP MULTIPLY
1101		TRT TRANSLATE AND TEST		DP DIVIDE
1110		ED EDIT		
1111		EDMK EDIT AND MARK		

NOTE: N = NORMALIZED DL = DOUBLE LOGICAL S = SINGLE
SL = SINGLE LOGICAL U = UNNORMALIZED D = DOUBLE

APPENDIX 1 IBM System/370 ISP

```
S370 :=
    BEGIN

! ISP summary description of IBM System/370 architecture.

! This summary gives an overview of the major architectural
! features of the S370.

! Instruction fetch and execution cycles are fully described,
! but the actual execution of individual instructions is limited
! to one or two examples from each group (RR, RX, RS/SI, and SS).

! The summary is fully compatible with the ISPS compiler and simulator.

    **MP.State**

        macro maxb   := |16383 |,
        macro maxkey :=     |7 |,

        MB[0:maxb]<0:7>,                    ! Byte memory
        MH[0:maxb]<0:15>{INCREMENT:2} :=    ! Half word memory
                MB[0:maxb]<0:7>,
        MW[0:maxb]<0:31>{INCREMENT:4} :=    ! Word memory
                MB[0:maxb]<0:7>,
        MDW[0:maxb]<0:63>{INCREMENT:8} :=   ! Doubleword memory
                MB[0:maxb]<0:7>,

        MAR<0:23>,                          ! Memory address reg
        MBR<0:31>,                          ! Memory buffer reg

        STKEYS[0:maxkey]<0:4>,              ! Storage key array

    ! Permanent Storage Assignments

        iplpsw<0:63> := MB[0:7]<0:7>,       ! IPL PSW
        iplcw1<0:63> := MB[8:15]<0:7>,      ! IPL CCW #1
        iplcw2<0:63> := MB[16:23]<0:7>,     ! IPL CCW #2
        exopsw<0:63> := MB[24:31]<0:7>,     ! External Old PSW
        svcpsw<0:63> := MB[32:39]<0:7>,     ! SVC Old PSW
        propsw<0:63> := MB[40:47]<0:7>,     ! Program Old PSW
        mkopsw<0:63> := MB[48:55]<0:7>,     ! Machine check Old PSW
        ioopsw<0:63> := MB[56:63]<0:7>,     ! I/O Old PSW
        chstwd<0:63> := MB[64:71]<0:7>,     ! Channel Status Word
        chadwd<0:31> := MB[72:75]<0:7>,     ! Channel Address Word
        timer<0:23>  := MB[80:82]<0:7>,     ! Timer cell
        exnpsw<0:63> := MB[88:95]<0:7>,     ! External New PSW
        svnpsw<0:63> := MB[96:103]<0:7>,    ! SVC New PSW
        prnpsw<0:63> := MB[104:111]<0:7>,   ! Program New PSW
        mknpsw<0:63> := MB[112:119]<0:7>,   ! Machine check New PSW
        ionpsw<0:63> := MB[120:127]<0:7>,   ! I/O New PSW
        scnuut<0:63> := MB[128:135]<0:7>    ! Diagnostic scan out
    **PC.State**

        R[0:15]<0:31>,                      ! General purpose registers
        PSW<0:63>,                          ! Program Status Word

        psww[0:3]<0:15>  := PSW<0:63>,      ! Alternate PSW definition
            CHAMSK<0:7>  := PSW<0:7>,       ! Channel Mask
            PROTKY<0:3>  := PSW<8:11>,      ! Protection Xey
            ASCMSK<>     := PSW<12>,        ! USASCII Mask
            MCHKMK<>     := PSW<13>,        ! Machine check mask
            WAITST<>     := PSW<14>,        ! Wait state
            PRODST<>     := PSW<15>,        ! Problem state
            INTCDE<0:15> := PSW<16:31>,     ! Interrupt code
            ILC<0:1>     := PSW<32:33>,     ! Instruction length code
            CC<0:1>      := PSW<34:35>,     ! Condition code
            FPOPMS<>     := PSW<36>,        ! Fixed point overflow mask
            DOFMSK<>     := PSW<37>,        ! Decimal overflow mask
            EXOFMS<>     := PSW<38>,        ! Exponent underflow mask
            SIGMSK<>     := PSW<39>,        ! Significance mask
            PC<0:23>     := PSW<40:83>      ! Program counter-24 bits

**Implementation.Declarations**

    amar1<0:23>,                            ! Auxillary memory address reg.(1)
    amar2<0:23>,                            ! Auxillary memory address reg.(2)
    lobyte<8:7>     := MBR<0:7>,            ! Left byte in MBR
    hibyte<0:7>     := MBR<24:31>,          ! Right byte in MBR
    leux1<0:7>,                             ! Byte count register 1
    laux2<0:7>,                             ! Byte count register 2
    exrf<>,                                 ! Execute recursion flag
    t2<0:1>,                                ! 2 bit temp
    t4<8:3>,                                ! 4 bit temp
    t24<8:23>,                              ! 24 bit temp
    ovf<>,                                  ! Overflow
    stopbit<>,                              ! Stop switch
    intvec<8:4>,                            ! Interrupt vector:
                                            !   Bit 8 = machine check
                                            !   Bit 1 = svc
                                            !   Bit 2 = prog check
                                            !   Bit 3 = timer interrupt
                                            !   Bit 4 = I/O interrupt
    iomsk<0:7>,                             ! Channel mask register
    chrls<>,                                ! Channel release
    chsel<>,                                ! Channel select register
    chancc<0:1>,                            ! Channel condition code
    chinst[0:3]<>,                          ! Channel instruction line
                                            !   0 => SIO
                                            !   1 => TIO
                                            !   2 => HIO
                                            !   3 => TCH
    chareg<0:7>,                            ! Channel address register
    devreg<0:7>,                            ! Device register
                                            ! Holds device address (0-255)
    extreg<0:7>,                            ! External register:
                                            !   Bit 0 = timer interrupt
                                            !   Bit 1 = console interrupt
    iodreg<0:7>,                            ! Holds data byte for direct I/O
                                            ! its meaning (command or data)
                                            ! is implementation dependent
    sigout<0:9>,                            ! Signal out for direct I/O

**Instruction.Format**

    IR<0:47>,                               ! Instruction register
    irw[0:2]<0:15>  := IR<0:47>,            ! 1/2 word address for IR (execute)
        OpCode<0:7> := IR<0:7>,             ! RR, RX, RS, SI, SS
        R1<0:3>     := IR<8:11>,            ! RR, RX, RS
        R2<0:3>     := IR<12:15>,           ! RR
        X2<0:3>     := IR<12:15>,           ! RX
        B1<0:3>     := IR<16:19>,           ! RX, RS, SI, SS
        D1<0:11>    := IR<20:31>,           ! RX, RS, SI, SS
        R3<0:3>     := IR<12:15>,           ! RS
        M1[0:3]<>   := IR<8:11>,            ! Mask 1
        I2<0:7>     := IR<8:15>,            ! SI
        LFLD<0:7>   := IR<8:15>,            ! SS
        L1<0:3>     := IR<8:11>,            ! SS
        L2<0:3>     := IR<12:15>,           ! SS
        B2<0:3>     := IR<32:35>,           ! SS
        D2<0:11>    := IR<36:47>            ! SS

**Service.Facilities**(us)

! Interrupt code 5 implies addressing error
! Interrupt code 6 implies specification (alignment error)
! Interrupt code 4 implies protection
! The order of setting these codes may be implementation dependent.
! Tests on a model 75 show code 6 is first.

    ckpr :=                     ! Check routine for storage protection
        begin
        checker(STKEYS[MAR<0:21>]<1:4> neq PROTKY, 4)
        end,

    ckrdpr :=                   ! Check routine for read protection
        begin
        IF STKEYS[MAR<0:12>]<0> => ckpr()
        end,

    keyck :=                    ! Check routine for ssk & isk instructions
        begin
        checker(PRODST, 2) next         ! Privileged state check
        checker(R[R2]<28:31> neq 0, 6) next
        checker(R[R2]<8:20> gtr 7, 5)
        end,
    ckhwad :=                   ! Check 1/2 word address routine
        begin
        checker(MAR<23>, 6) next
        checker(MAR<0:22> gtr maxb/2, 5)
        end,

    ckwdad :=                   ! Check word address routine
        begin
        checker(MAR<22:23> neq 0, 6) next
        checker(MAR<0:21> gtr maxb/4, 5)
        end,

    ckdwad :=                   ! Check double word address routine
        begin
        checker(MAR<21:23> neq 0, 6) next
        checker(MAR<0:20> gtr maxb/8, 5)
        end,

    checker(condition<>, intcode<0:15>) :=
        begin
        IF condition =>
            begin
            INTCDE = intcode; intvec<2> = 1 next
            LEAVE icycle
            end
        end,

    rdbyte :=                   ! Read a byte routine
        begin
        checker(MAR gtr maxb, 5) next   ! Check valid byte address
        ckrdpr() next
        MBR<24:31> = MB[MAR]
        end,

    wrbyte :=                   ! Write a byte routine
        begin
        checker(MAR gtr maxb, 5) next   ! Check valid byte address
        ckpr() next
        MB[MAR] = MBR<24:31>
        end,

    readhw :=                   ! Read a 1/2 word routine
        begin
        ckhwad() next
        ckrdpr() next
        MBR<16:31> = MH[MAR]
        end,
```

```
    wrhw :=                         ! Write a 1/2 word routine
        begin
        ckhwad() next
        ckpr() next
        MH[MAR] = MBR<16:31>
        end,

    readwd :=                       ! Read a word routine
        begin
        ckwdad() next
        ckrdpr() next
        MBR = MW[MAR]
        end,

    wrwd :=                         ! Write a word routine
        begin
        ckwdad() next
        ckpr() next
        MW[MAR] = MBR
        end,

    l2fch :=                        ! Fetch of L2 operand if possible or a load
        begin                       ! of zero into the MBR if L2 field is exhausted.
        DECODE laux2 eql 0 =>
            begin
            0 := begin
                 laux2 = laux2 -(tc) 1 next
                 MAR = amar2 + laux2 next
                 rdbyte()
                 end,
            1 := MBR = 0
            end
        end,

    adrio :=                        ! Device addressing for I/O instructions
        begin
        chareg = (D1 + R[B1])<15:8>; devreg = (D1 + R[B1])<7:0>
        end,

    setfcc :=                       ! Set fixed point condition codes
        begin
        CC = 0 next
        DECODE R[R1]<0> =>
            begin
            0 := IF R[R1] => CC = 2,
            1 := CC = 1
            end  next
        IF ovf => CC = 3 next
        checker(ovf and FPOPMS, 8)
        end,

    opex := (checker('1, 1)),       ! Illegal op-code
    int :=
        begin
        t2 = ILC next                       ! Save instruction length

        IF intvec<0> and MCHKMK =>          ! Handle priority (1) interrupts
            begin
            mkopsw = PSW next
            mkopsw<16:31> = 0 next
            scnout = PSW next
            PSW = mknpsw; intvec<0:2> = 0
            end  next

        IF intvec<1> =>                     ! Handle priority (2) interrupts
            begin
            svcpsw = PSW next
            PSW = svnpsw; intvec<1> = 0
            end  next

        IF intvec<2> =>
            begin
            propsw = PSW next
            PSW = prnpsw; intvec<2> = 0
            end  next

        IF intvec<3> and CHAMSK =>          ! Handle priority (3) interrupts
            begin
            INTCDE = extreg next
            exopsw = PSW next
            PSW = exnpsw; intvec<3> = 0
            end  next

        IF intvec<4> and iomsk =>           ! Handle priority (4) interrupts
            begin
            INTCDE = devreg next
            ioopsw = PSW next
            PSW = ionpsw; intvec<4> = 0
            end  next

        INTCDE = 0; ILC = t2                ! Reset ILC & interrupt code
        end
**Instruction.Interpretation**{us}

    start(main) :=
        begin
        stopbit = 0 next
        run()
        end,

    run :=
        begin
        IF not stopbit =>
            begin
            IF not WAITST => icycle() next
            int() next
            RESTART run
            end
        end,

    icycle :=                       ! Instruction interpretation cycle
        begin
        ifetch() next
        iexec() next
        IF exrf => (iexec() next exrf = 0)
        end,

    ifetch :=                       ! Instruction fetch
        begin
        MAR = PC next
        readhw() next
        IR<0:15> = MBR<16:31>; ILC = (MBR<18> + MBR<17>) + 1;
        PC = PC + ((MBR<18> + MBR<17>) + 1) * 2; ovf = 0 next
        IF ILC gtr 1 =>
            begin
            MAR = MAR + 2 next
            readhw() next
            IR<16:31> = MBR<16:31> next
            IF ILC gtr 2 =>
                begin
                MAR = MAR + 2 next
                readhw() next
                IR<32:47> = MBR<16:31>
                end
            end
        end

**Instruction.Execution**{us}

    iexec :=
        begin
        DECODE OpCode<0:1> =>
            begin
            '00 := RR(),
            '01 := RX(),
            '10 := RS.SI(),
            '11 := SS()
            end
        end,
    RR :=       !  RR instruction decode table
        begin
        DECODE OpCode =>
            begin
            "04 := SPM(),               ! Set program mask
            "06 := BALR(),              ! Branch and link

        "06:"3F := no.op(),             ! Opcodes not shown in this summary

      otherwise := opex()
            end
        end,

    RX :=       ! RX instruction decode table
        begin
        MAR = D1 next                   ! Effective address calculation
        IF B1 => MAR = MAR + R[B1] next
        IF X2 => MAR = MAR + R[X2] next

        DECODE OpCode =>                ! Opcode decode for RX
            begin
            "40 := STH(),               ! Store halfword

        "41:"74 := no.op(),             ! Opcodes not shown in this summary

      otherwise := opex()
            end
        end,

    RS.SI :=    ! RS, SI instruction decode table
        begin

        MAR = D1 next                   ! Effective address calculation
        IF B1 => MAR = MAR + R[B1] next

        DECODE OpCode =>                ! Opcode decoding
            begin
            "80 := SSM(),               ! Set system mask

        "81:"9F := no.op(),             ! Opcodes not shown in this summary

      otherwise := opex()
            end
        end,

    SS :=       ! SS instruction decode table
        begin

        amar1 = D1; amar2 = D2 next     ! Effective address calculation
        IF B1 => amar1 = amar1 + R[B1];
        IF B2 => amar2 = amar2 + R[B2] next
```

APPENDIX 1 (Cont'd.)

```
    DECODE OpCode =>                ! Opcode decoding
        begin
        "01 := MVN(),               ! Move numerics

    "D2:"FF := no.op(),             ! Opcodes not shown in this summary

  otherwise := opex()
        end
    end,
! RR instructions

    SPM :=              ! Set program mask
        begin
        PSW<34:39> = R[R1]<2:7>
        end,

    BALR :=             ! Branch and link register
        begin
        t24 = R[R2]<8:31> next
        R[R1] = PSW<32:63> next
        IF R2 => PC = t24
        end,

!       . . . .                             ! Instruction descriptions not
!       . . . .                             ! included in this summary.

! RX instructions

    STH :=              ! Store halfword
        begin
        MBR = R[R1]<16:31> next
        wrhw()
        end,

!       . . . .
!       . . . .

! RS, SI instructions

    SSM :=              ! Set system mask
        begin
        checker(PROBST, 2) next         ! Privileged stete check
        rdbyte() next
        PSW<0:7> = hibyte
        end,

!       . . . .
!       . . . .

! SS instructions

    MVN :=              ! Move numerics
        begin
        laux1 = 0; laux2 = 0 next
        mvn1 := begin
                MAR = amar2 + laux2 next
                rdbyte() next
                MAR = amar1 + laux1; t4 = MBR<28:31> next
                rdbyte() next
                MDR<28:31> = t4 next
                wrbyte() next
                IF LFLD gtr laux2 =>
                    begin
                    laux1 = laux1 + 1; laux2 = laux2 + 1 next
                    RESTART mvn1
                    end
                end
        end,

    end

!       . . . .
!       . . . .

! End of S370 summary description
```

Chapter 41

The Structure of SYSTEM/360[1]

Part II—System Implementations

W. Y. Stevens

Summary The performance range desired of SYSTEM/360 is obtained by variations in the storage, processing, control, and channel functions of the several models. The systematic variations in speed, size, and degree of simultaneity that characterize the functional components and elements of each model are discussed.

A primary goal in the SYSTEM/360 design effort was a wide range of processing unit performances coupled with complete program compatibility. In keeping with this goal, the logical structure of the resultant system lends itself to a wide choice of components and techniques in the engineering of models for desired performance levels.

This paper discusses basic choices made in implementing six SYSTEM/360 models spanning a performance range of fifty to one. It should be emphasized that the problems of model implementation were studied throughout the design period, and many of the decisions concerning logical structure were influenced by difficulties anticipated or encountered in implementation.

Performance Adjustment

The choices made in arriving at the desired performances fall into four areas:

Main storage

Central processing unit (CPU) registers and data paths

Sequence control

Input/output (I/O channels)

Each of the adjustable parameters of these areas can be subordinated, for present purposes, to one of three general factors: basic speed, size, and degree of simultaneity.

Main Storage

Storage Speed and Size

The interaction of the general factors is most obvious in the area of main storage. Here the basic speeds vary over a relatively small range: from a 2.5-μsec cycle for the Model 40 to a 1.0-μsec cycle for Models 62 and 70. However, in combination with the other two factors, a 32:1 range in overall storage data rate is obtained, as shown in Table 1.

Most important of the three factors is size. The width of main storage, i.e., the amount of data obtained with one storage access, ranges from one byte for the Model 30, two bytes for the Model 40, and four bytes for the Model 50, to eight bytes for Models 60, 62, and 70.

Another size factor, less direct in its effect, is the total number of bytes in main storage, which can make a large difference in system throughput by reducing the number of references to external storage media. This number ranges from a minimum of 8192 bytes on Model 30 to a maximum of 524,288 bytes on Models 60, 62, and 70. An option of up to eight million more bytes of slower-speed, large-capacity core storage can further increase the throughput in some applications.

Interleaved Storage

Simultaneity in the core storage of Models 60 and 70 is obtained by overlapping the cycles of two storage units. Addresses are staggered in the two units, and a series of requests for successive words activates the two units alternately, thus doubling the maximum rate. For increased system performance, this technique is less effective than doubling the basic speed of a single unit, since the access time to a single word is not improved, and successive references frequently occur to the same unit. This is illustrated by comparing the performances of Models 60 and 62, whose only difference is the choice between two overlapped 2.0-μsec storage units and one single 1.0-μsec storage unit, respectively. The performance of Model 62 is approximately 1.5 times that of Model 60.

CPU Registers and Data Paths

Circuit Speed

SYSTEM/360 has three families of logic circuits, as shown in Table 2, each using the same solid-logic technology. One family, having a nominal delay of 30 nsec per logical stage or level, is used in the data paths of Models 30, 40, and 50. A second and faster family with a nominal delay of 10 nsec per level is used in Models 60 and 62. The fastest family, with a delay of 6 nsec, is used in Model 70.

The fundamental determinant of CPU speed is the time required to take data from the internal registers, process the data through the adder or other logical unit, and return the result to a register. This cycle time is determined by the delay per logical circuit level and the number of levels in the register-to-adder path, the adder, and the adder-to-register return path. The

[1]*IBM Sys. J*, vol. 3, no. 2, 1964, pp. 136–143.

Table 1 System/360 Main Storage Characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60*	*Model 62*	*Model 70*
Cycle time (μsec)	2.0	2.5	2.0	2.0	1.0	1.0
Width (bytes)	1	2	4	8	8	8
Interleaved access	no	no	no	yes	no	yes
Maximum data rate (bytes/μsec)	0.5	0.8	2.0	8.0	8.0	16.0
Minimum storage size (bytes)	8,192	16,384	65,536	131,072	262,144	262,144
Maximum storage size (bytes)	65,536	262,144	262,144	524,288	524,288	524,288
Large capacity storage attachable	no	no	yes	yes	yes	yes

number of levels varies because of the trade-off that can usually be made between the number of circuit modules and the number of logical levels. Thus, the cycle time of the system varies from 1.0 μsec for Model 30 (with 30-nsec circuits, a relatively small number of modules, and more logic levels) and 0.5 μsec for Model 50 (also with 30-nsec circuits, but with more modules and fewer levels) to 0.2 μsec for Model 70 (with 6-nsec circuits).

Local Storage

The speed of the CPU depends also on the speed of the general and floating-point registers. In Model 30, these registers are located in an extension to the main core storage and have a read-write time of 2.0 μsec. In Model 40, the registers are located in a small core-storage unit, called *local storage*, with a read-write time of 1.25 μsec. Here, the operation of the local storage may be overlapped with main storage. In Model 50, the registers are in a local storage with a read-write time of only 0.5 μsec. In Model 60/62, the local storage has the logical characteristics of a core storage with nondestructive read-out; however, it is actually constructed as an array of registers using the 30-nsec family of logic circuits, and has a read-write time of 0.25 μsec. In Model 70, the general and floating-point registers are implemented with 6-nsec logic circuits and communicate directly with the adder and other data paths.

The two principal measures of size in the CPU are the width of the data paths and the number of bytes of high-speed working registers.

Data Path Organization

Model 30 has an 8-bit wide (plus parity) adder path, through which all data transfers are made, and approximately 12 bytes of working registers.

Model 40 also has an 8-bit wide adder path, but has an additional 16-bit wide data transfer path. Approximately 15 bytes

Table 2 System/360 CPU Characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Circuit family: nominal delay per logic level (nsec)	30	30	30	10	6
Cycle time (μsec)	1.0	0.625	0.5	0.25	0.2
Location of general and floating registers	main core storage	local core storage	local core storage	local transistor storage	transistor registers
Width of general and floating register storage (bytes)	1	2	4	4	4 or 8
Speed of general and floating register storage (μsec)	2.0	1.25	0.5	0.25	
Width of main adder path (bits)	8	8	32	56	64
Width of auxiliary transfer path (bits)		16	8		
Widths of auxiliary adder paths (bits)				8	8, 8, and 24
Approximate number of bytes of register storage	12	15	30	50	100
Approximate number of bytes of working locations in local storage	45 (main storage)	48	60	4	
Relative computing speed	1	3.5	10	21/30	50

of working registers are used, plus about 48 bytes of working locations in the local storage, exclusive of the general and floating-point registers.

Model 50 has a 32-bit wide adder path, an 8-bit wide data path used for handling individual bytes, approximately 30 bytes of working registers, plus about 60 bytes of working locations in the local storage.

Model 60/62 has a 56-bit wide main adder path, an 8-bit wide serial adder path, and approximately 50 bytes of working registers.

Model 70 has a 64-bit wide main adder, an 8-bit wide exponent adder, an 8-bit wide decimal adder, a 24-bit wide addressing adder, and several other data transfer paths, some of which have incrementing ability. The model has about 100 bytes of working registers plus the 96 bytes of floating point and general registers which, in Model 70, are directly associated with the data paths.

The models of SYSTEM/360 differ considerably in the number of relatively independent operations that can occur simultaneously in the CPU. Model 30, for example, operates serially: virtually all data transfers must pass through the adder, one byte at a time. Model 70, however, can have many operations taking place at the same time. The CPU of this model is divided into three units that operate somewhat independently. The instruction preparation unit fetches instructions from storage, prepares them by computing their effective addresses, and initiates the fetching of the required data. The execution unit performs the execution of the instruction prepared by the instruction unit. The third unit is a storage bus control which coordinates the various requests by the other units and by the channels for core-storage cycles. All three units normally operate simultaneously, and together provide a large degree of instruction overlap. Since each of the units contains a number of different data paths, several data transfers may be occurring on the same cycle in a single unit.

The operations of other SYSTEM/360 models fall between those mentioned. Model 50, for example, can have simultaneous data transfers through the main adder, through an auxiliary byte transfer path, and to or from local storage.

Sequence Control

Complex Instruction Sequences

Since the SYSTEM/360 has an extensive instruction set, the CPU's must be capable of executing a large number of different sequences of basic operations. Furthermore, many instructions require sequences that are dependent on the data or addresses used. As shown in Table 3, these sequences of operations can be controlled by two methods; either by a conventional sequential logic circuit that uses the same types of circuit modules as used in the data paths or by a read-only storage device that contains a microprogram specifying the sequences to be performed for the different instructions.

Model 70 makes use of conventional sequential logic control mainly because of the high degree of simultaneity required. Also, a sufficiently fast read-only storage unit was not available at the time of development. The sequences to be performed in each of the Model 70 data paths have a considerable degree of independence. The read-only storage method of control does not easily lend itself to controlling these independent sequences, but is well adapted where the actions in each of the data paths are highly coordinated.

Read-Only Storage Control

The read-only storage method of control is described elsewhere [Peacock, n.d.]. This microprogram control, used in all but the fastest model of SYSTEM/360, is the only method known by which an extensive instruction set may be economically realized in a small system. This was demonstrated during the design of Model 60/62. Conventional logic control was originally planned for this model, but it became evident during the design period that too many circuit modules were required to implement the instruction set, even for this rather large system. Because a sufficiently fast read-only storage became available, it was adopted for sequence control at a substantial cost reduction.

The three factors of speed, size, and simultaneity are applicable

Table 3 System/360 Sequence Control Characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Type	read-only storage	read-only storage	read-only storage	read-only storage	sequential logic
Cycle time (μsec)	1.0	0.625	0.5	0.25	0.2
Width of read-only storage word (available bits)	60	60	90	100	
Number of read-only storage words available	4096	4096	2816	2816	
Number of gate-control fields in read-only storage word	9	10	15	16	

to the read-only storage controls of the various SYSTEM/360 models. The speed of the read-only storage units corresponds to the cycle time of the CPU, and hence varies from 1.0 μsec per access for Model 30 down to 0.25 μsec for Models 60 and 62.

The size of read-only storage can vary in two ways—in width (number of bits per word) and in number of words. Since the bits of a word are used to control gates in the data paths, the width of storage is indirectly related to the complexity of the data paths. The widths of the read-only storages in SYSTEM/360 range from 60 bits for Models 30 and 40 to 100 bits for Models 60 and 62. The number of words is affected by several factors. First, of course, is the number and complexity of the control sequences to be executed. This is the same for all models except that Model 60/62 read-only storage contains no sequences for channel functions. The number of words tends to be greater for the smaller models, since these models require more cycles to accomplish the same function. Partially offsetting this is the fact that the greater degree of simultaneity in the larger systems often prevents the sharing of microprogram sequences between similar functions.

SYSTEM/360 employs no read-only storage simultaneity in the sense that more than one access is in progress at a given time. However, a single read-only storage word simultaneously controls several independent actions. The number of different gate control fields in a word provides some measure of this simultaneity. Model 30 has 9 such fields. Model 60/62 has 16.

Input/Output Channels

Channel Design

The SYSTEM/360 input/output channels may be considered from two viewpoints: the design of a channel itself, or the relationship of a channel to the whole system.

From the viewpoint of channel design, the raw speed of the components does not vary, since all channels use the 30-nsec family of circuits. However, the different channels do have access to different speeds of main storage and, in the three smaller models, different speeds of local storage.

The channels differ markedly in the amount of hardware devoted exclusively to channel use, as shown in Table 4. In the Model 30 multiplexor channel, this hardware amounts only to three 1-byte wide data paths, 11 latch bits for control, and a simple interface polling circuit. The channel used in Models 60, 62, and 70 contains about 300 bits of register storage, a 24-bit wide adder, and a complete set of sequential control circuits. The

Table 4 System/360 Channel Characteristics

	Model 30	*Model 40*	*Model 50*	*Model 60/62*	*Model 70*
Selector channels					
Maximum number attachable	2	2	3	6	6
Approximate maximum data rate on one channel in Kbyps†	250	400	800 (1250 on high speed)	1250	1250
Uses CPU data paths for:					
iniation and termination	yes	yes	yes	yes	yes
byte transfers	no	no	no	no	no
storage word transfers	no	low speed only	yes	no	no
chaining	yes	yes	yes	no	no
CPU and I/O overlap possible	yes	yes	regular—yes high speed—no	yes	yes
Multiplexor channels					
Maximum number attachable	1	1	1	0	0
Minimum number of subchannels	32	16	64		
Maximum number of subchannels	96	128	256		
Maximum data rate in byte interleaved mode (Kbyps)	16	30	40		
Maximum data rate in burst mode (Kbyps)	200	200	200		
Uses CPU data paths for all functions	yes	yes	yes		
CPU and I/O overlap possible in byte mode	yes	yes	yes		
CPU and I/O overlap possible in burst mode	no	no	yes		

†Thousand bytes per second.

amount of hardware provided for other channels is somewhere in between these extremes.

The disparity in the amount of channel hardware reflects the extent to which the channels share CPU hardware in accomplishing their functions. Such sharing is done at the expense of increased interference with the CPU, of course. This interference ranges from complete lock-out of CPU operations at high data rates on some of the smaller models, to interference only in essential references to main storage by the channel in the large models.

Channel/System Relationship

When the channels are viewed in their relationship to the whole system, the three factors of speed, size, and simultaneity take on a different aspect. The channel is viewed as a system component, and its effect on system throughput and other system capabilities is of concern. The speeds of the channels vary from a maximum rate of about 16 thousand bytes per second (byte interleaved mode) on the multiplexor channel of Model 30 to a maximum rate of about 1250 thousand bytes per second on the channels of Models 60, 62, and 70. The size of each of the channels is the same, in the sense that each handles an 8-bit byte at a time and each can connect to eight different control units. A slight size difference exists among multiplexor channels in terms of the maximum number of subchannels.

The degree of channel simultaneity differs considerably among the various models of SYSTEM/360. For example, operation of the Model 30 or 40 multiplexor channels in burst mode inhibits all other activity on the system, as does operation of the special high-speed channel on Model 50. At the other extreme, as many as six selector channels can be operating concurrently with the CPU on Models 60, 62, or 70. A second type of simultaneity is present in the multiplexor channels available on Models 30, 40, and 50. When operating in byte interleaved mode, one of these channels can control a number of concurrently operating input/output devices, and the CPU can also continue operation.

Differences in Application Emphasis

The models of SYSTEM/360 differ not only in throughput but also in the relative speeds of the various operations. Some of these relative differences are simply a result of the design choices described in this paper, made to achieve the desired overall performance. The more basic differences in relative performance of the various operations, however, were intentional. These differences in emphasis suit each model to those applications expected to comprise its largest usage.

Thus the smallest system is particularly aimed at traditional commercial data processing applications. These are characterized by extensive input/output operations in relation to the internal processing, and by more character handling than arithmetic. The fast selector channels and character-oriented data paths of Model 30 result from this emphasis. But despite this emphasis, the general-purpose instruction set of SYSTEM/360 results in much better scientific application performance for Model 30 than for its comparable predecessors.

On the other hand, the large systems are expected to find particularly heavy use in scientific computation, where the emphasis is on rapid floating-point arithmetic. Thus Models 60, 62, and 70 contain registers and adders that can handle the full length of a long format floating-point operand, yet do character operations one byte at a time.

No particular emphasis on either commercial or scientific applications characterizes the intermediate models. However, Models 40 and 50 are intended to be particularly suitable for communication-oriented and real-time applications. For example, Model 50 includes a multiplexor channel, storage protection, and a timer as standard features, and also provides the ability to share main storages between two CPU's in a multiprocessing arrangement.

References

Peacock [n.d.].