A Survey and Taxonomy of I/O Systems

Mark Smotherman
Last updated: July 2006

Summary: The range of different approaches to starting I/O and synchronizing CPU execution with I/O completions is more vast than what is found in typically textbook presentations.

Introduction

Fred Brooks held the opinion that the further away you go out from the CPU, the more varied the design details. He saw this exemplified in the various schemes used for input/output on computers. I agree with his assessment, and this essay attempts to give a taxonomy to these schemes and classify historical computers accordingly.

This work is adapted from my earlier paper, entitled "A Sequencing-Based Taxonomy of I/O Systems and Review of Historical Machines," which is Paper 1 of Chapter 7, in M. Hill, N. Jouppi, and G. Sohi (eds.), Readings in Computer Architecture, Morgan Kaufmann, San Francisco, 2000. That paper originally appeared in Computer Architecture News, vol. 17, no. 5, pp. 5-15, September 1989, in the special issue on input/output architecture.

Sidebar: A Summary of Historical Firsts in I/O

- IBM 702 (1953) - concept of control unit

  Note (added July 25, 2006): In 1952, Werner Bucholz suggested that overlapped I/O and CPU execution be added to the TPM II design, which became the IBM 702. "The TPM II engineering team was unable to justify the additional development time, expense, and product complexity involved." [Ba81]

- NBS DYSEAC (1954) - first use of I/O interrupts

- IBM NORC (1954) - I/O channel

  Note 1 (added May 10, 2005): "IBM's Naval Ordnance Research Calculator (NORC) ... [was built] between 1950 and 1954 at Columbia University's Watson Scientific Computing Laboratory .... The NORC also included the first input-output channel, which synchronized the flow of data into and out of the computer while computation was in progress, relieving the central processor of that task, a concept that was quickly adopted across the industry." http://www.columbia.edu/acis/history/norc.html

  Note 2 (added May 10, 2005): In an interview, Gene Amdahl told me that he designed the I/O channel architecture for the IBM 709, although he shortly thereafter left IBM and credit was given to others. (See US Patent 3,812,475, Christiansen, Kanter, and Monroe, "Data Synchronizer," filed on Dec, 26, 1957, granted on May 21, 1974.) Gene Amdahl's brother, Lowell, was one of the logic designers on the NORC.

Mea culpa

- In the 1989 version of this paper, I assigned the innovation of DMA to the IBM SAGE, but I am unable to date its use in SAGE to earlier than the publication of the concept in the 1954 DYSEAC papers. DYSEAC thus seems to be the first to use DMA.

- Also in that version, I assigned the innovation of interrupt vectors to the Lincoln Labs TX-2. A case can also be made for tracing this innovation to Fred Brooks' work (1957 paper and patent) on an interrupt system for the IBM Stretch.

See a Short Review of the History of Interrupts for more details.

Input/Output Taxonomies

Typical textbook presentations of I/O systems identify only four methods of data transfer:

1. program-controlled I/O (i.e., polling),
2. interrupt-driven I/O,
3. DMA, and
4. channel I/O.

Blaauw and Brooks go beyond this narrow view, but they identify essentially only seven different categories in their textbook, Computer Architecture: Concepts and Evolution,

1. synchronous I/O using CPU registers
2. asynchronous I/O using private device buffers
3. asynchronous I/O using shared device buffers
4. asynchronous I/O using main memory buffers
5. channel I/O
6. heterogeneous multiprocessor I/O (PPUs)
7. homogeneous multiprocessor I/O

The first category above is the case where I/O completely involves the CPU, the next three categories allow some form of overlap between a single I/O instruction and the CPU, and the final three categories involve some form of autonomous processing of multiple I/O transfers using a processor with an identifiable program counter.

Based on Brooks' design details dictum above, I think a different, and much wider taxonomy should be used to partition the I/O design space. In particular, I would adopt a programming, or sequence-based, view of the parallelism between CPU processing and I/O processing. For uniprocessors, I believe that the major issue is the method of initiating the I/O transfer; and, for multiprocessors, the major issue is the symmetry of I/O.

The method of initiating a transfer is an indication of the level of functionality of the I/O subsystem, specifically, how often and how much the CPU is involved in the actual transfer. This ranges from the CPU doing it all to autonomous peripheral processing units. Within this range, I draw a distinction between a controller that can transfer only one block before requiring CPU intervention and a controller that can transfer multiple blocks in a scatter/gather type of operation (in which the blocks are identified to the controller by a chain of descriptors). Some designers and authors call a controller with the latter capability an I/O channel. Indeed, Bell and Newell categorize controllers with scatter/gather capability as Pios, since they consider the chain of block descriptors to be a series of jump instructions [Be71]. However, in this taxonomy I reserve the term I/O channel for a specialized I/O processor that fetches instructions with identifiable opcode fields. Moreover, I also use the distinction made by Blaauw and Brooks between I/O channels and I/O processors, which is the general ability to count. That is, an I/O processor should have the ability to maintain a loop or event count that is unrelated to the transfer of a given number of words or characters per block.

Under each approach to transfer initiation, I subdivide the category according to the method of completion reporting. This includes the common methods of polling and interruption, as well as higher-level, device-driver-like queueing for the Honeywell Series 60 Level 64 minicomputer, ELXSI 6400, and Intel 432.

For multiprocessors, the method of initiation is not as important as the symmetry of the initiation; therefore, this symmetry (or lack of it) becomes the basis of the major categories. The method of completion reporting is the basis of subcategories, and symmetry in interruption is explicitly identified.

A classification of historical machines serves to demonstrate the usefulness of the proposed taxonomy and also serves as a guided tour of the history of I/O systems. For older machines with multiple I/O options, I have chosen to classify them according to their established use. For example, the IBM S/360 had a synchronous I/O capability, but it was rarely used. Therefore, the S/360 appears as an example in the conditional-instruction/interrupt subcategory. Some multiprocessors also appear in the first section; this is because they represent the first use of a given transfer initiation method.

Not all categories are populated with machines. This may be the result of omissions on my part, or the category may indeed be unfruitful.

A Sequencing-Based Taxonomy

I. CPU - I/O INTERACTION
- A. Synchronous transfer
- B. Asynchronous transfer
  1. interlocked instruction to start transfer
     - a. synchronization by interlock
     - b. synchronization by polling
       - i. separate instructions to poll and transfer data
       - ii. controller transfers words of block (i.e., DMA)
       - iii. controller with scatter/gather capability (often called an I/O channel)
       - iv. I/O channel (with specialized I/O instruction set)
       - v. I/O processor
     - c. synchronization by interrupt
       - i. separate instruction to transfer data
       - ii. controller transfers words of block (i.e., DMA)
       - iii. controller with scatter/gather capability (often called an I/O channel)
       - iv. I/O channel (with specialized I/O instruction set)
       - v. I/O processor
  2. conditional instruction to start transfer
     - a. synchronization by polling
     - b. synchronization by interrupt
  3. mailbox deposit to start transfer (i.e., single entry)
     - a. synchronization by polling
     - b. synchronization by interrupt
  4. queue insert to start transfer (i.e., multiple entries)
     - a. synchronization by polling
     - b. synchronization by queueing
     - c. synchronization by interrupt
  5. asynchronous instruction to start transfer
     - a. synchronization by polling
     - b. synchronization by interrupt

II. MULTIPROCESSOR I/O
- A. Asymmetric initiation

1. synchronization by polling
2. synchronization by asymmetric interrupt
3. synchronization by symmetric interrupt

B. Symmetric initiation

1. synchronization by polling
2. synchronization by queueing
3. synchronization by asymmetric interrupt
4. synchronization by symmetric interrupt

A Review of Historical Machines

I. *CPU - I/O INTERACTION*

A. *Synchronous transfer*

ERA 1103 (1953) - word-at-a-time interlocked I/O [Be71]. Bell and Newell incorrectly designated the 1103 as the first computer to use interrupts. The Univac 1103A did include an interrupt capability, but this was after the NBS DYSEAC had already done so. (See history of interrupts for more details.)

IBM 702 (1953) - block interlocked I/O [Ba81,Ba86]. The CPU stalled while a block of characters was transferred from an I/O device buffer. The 702 introduced the control unit concept.

IBM 1401 (1959) [Ba81,Ba86]. This machine was originally designed as a printer controller but found widespread use as a small business computer. It used one opcode per I/O device, and these included reading a card into memory locations 0 to 79, punching a card from other locations, and printing from a third set of locations. The CPU stalled while the characters were transferred. An example of its ease of use is that a card duplicating program can be written in about 20 characters and punched onto one card.

B. *Asynchronous transfer*

1. *interlocked instruction to start transfer*

a. *synchronization by interlock*

UNIVAC I (1951) - buffered I/O [Be71,Ec52,We52]. This machine had one 60-word tape buffer for input and one for output. An initial input instruction started the transfer to the buffer and then released the CPU for overlapped instruction execution. A subsequent input instruction would dump the buffer to memory, start the next transfer, and then release the CPU. If a subsequent input instruction was issued too early, an interlock stalled the CPU. I/O errors halted the CPU, and the operator had to diagnose the problem.

IBM 701 (1952) - "copy logic" [Ba81,Ba86,Bl83,Bu53,St52]. After an initial prepare to read (or write) instruction, the program had to issue a copy instruction for each word in the transfer. A loop would be coded to update the memory addresses and issue the copies; the loop could also perform some superficial processing, such as character code conversion. The copy instruction was interlocked so that an early issue was stalled until the I/O device could provide or accept the next word. At end of file, the copy instruction caused a one-instruction skip; and, at the end of block, it caused a two-instruction skip.

b. *synchronization by polling*

i. *separate instructions to poll and transfer data*

PDP-1 (1959) [Be78]. This machine provided conditional skips on I/O buffer register contents, which were apparently used to poll for transfer completion.

PDP-8 (1965) [Be78,Be71]. Conditional skips on control unit status registers were used for polling.

ii. *controller transfers words of block (i.e., DMA)*

(?) Whirlwind I (1951) [Ev51]. Everett states, "In general the computer continues to run during terminal equipment wait times," but explains no further.

(see NBS DYSEAC)

IBM SAGE (or AN/FSQ-7, started 1952, operational 1955) - DMA operation [As57]. I/O operations started block transfers of data to/from drum buffers that proceed in parallel with further CPU operations. A controller generated the sequential memory addresses for a block and decremented a counter, while the CPU used a conditional branch to test for completion of the transfer. Transfers were interlocked so that the CPU was stalled if a second transfer was attempted before the previous one ended. Jacobs states "the input/output (I/O) break, or memory cycle stealing," was introduced in SAGE [Ja86], and Serrell, et al., identify "computation in parallel with I/O" as a significant new feature of SAGE [Se62]. However, I would be more inclined to give that credit to the NBS DYSEAC. (See history of interrupts for more details.)

iii. *controller with scatter/gather capability (often called an I/O channel)*

Honeywell 800 (1963) - hardware-assisted multiprogramming [Bo64,Ha60,Ha68,Lo59,Ma64]. This machine implemented eight virtual processors. On each memory cycle the hardware scanned on a priority basis for activity on up to eight input controllers, then up to eight output controllers, and then the CPU. Within the CPU the hardware scanned the virtual processors in a cyclic manner. (Once an instruction was started, it executed until completion.) Papers are unclear about program I/O synchronization. (See also H-800 sketch for more details.)

iv. *I/O channel (with specialized I/O instruction set)*

(see IBM 709)

v. *I/O processor*

(example unknown)

c. *synchronization by interrupt*

i. *separate instruction to transfer data*

Lincoln TX-2 (1957 paper) - "multiple sequence" [Fo57]. This machine contained 33 program counters; each I/O device had a dedicated PC and operated at a fixed priority. Each instruction had break and dismiss bits: break bits were used to indicate points at which higher-priority sequences could take over, while dismiss bits were used to allow lower-priority sequences to resume. Blaauw and Brooks classify this machine as having PPUs [Bl83], but I see the explicit instruction bits as a recognition of the sharing of a single CPU. Thus, I consider this machine closer to interrupt vectoring than to virtual PPUs.

PDP-1 (1959) [Be78]. Bell, et al., credit the "16-channel sequence break system" to TX-2 influence (actual operation not described).

ii. *controller transfers words of block (i.e., DMA)*

NBS DYSEAC (1954) - introduced I/O interrupt [Le54a,Le54b]. This machine had two program counters; an I/O signal caused the CPU to switch PCs. A bit in each instruction could force the switch back between PCs. Codd states, "in the NBS DYSEAC the very significant step was made of extending interruption to input-output operations" [Co62]. (See history of Interrupts for more details.)

UNIVAC 1107 (1962) [Bo78,Bo79]. Controller uses a single I/O control word, which contained a memory address, address increment/decrement flag, and a word count. Interrupt occurred on zero count when specified by a Load Channel command.

iii. *controller with scatter/gather capability (often called an I/O channel)*

IBM 7070 (1958) - "priority processing" (I/O interrupt) [Sv59]. An I/O completion caused the CPU to store the PC in a register and switch to an uninterruptible "priority routine". The 7070 also provided scatter/gather capability using chains of record definition words.

IBM STRETCH (started 1954, delivered 1961) [Bl83,Bu62,Du56]. The I/O exchange acted as a byte multiplexor. I/O completion was part of a comprehensive interrupt vector facility, in which each vector entry contained a single instruction to be executed outside the normal instruction cycle. These instructions could be single-instruction fixups or subroutine calls. In general, interrupt nesting was supported; however, I/O was treated as a single cause.

iv. *I/O channel (with specialized I/O instruction set)*

IBM 709 (1957) - introduction of channel I/O [Ba81,Ba86,Gr57]. The 709 had to execute two instructions in sequence to start I/O. A read select or write select instruction was first used to select a given device, and then a channel-specific instruction was used to reset and start any of the maximum of six channels (766s). The address field of the reset and start instruction was used to carry the channel program address. Some device select instructions were interlocked so that the CPU was stalled if a second select was issued before a previous one ended. The reset and select instructions, however, immediately affected the channels, which were much more sophisticated than the later IBM S/360 channels. Polling could be used for I/O completion, while interruption ("data-

channel trap") was available as an extra cost feature. Apparently, all installations chose to use the interrupt feature [M. Rubinstein, personal communication].

IBM 7090 (1958) [Be71,Bl83]. The optional "data-channel trap" feature of the 709 architecture was included as standard equipment. An interrupt vector with a pair of saved-PC and new-PC locations for each channel was used to resolve I/O completion traps. The later version channels (7909) could themselves be interrupted by external events and were capable of dealing with I/O retries without CPU intervention.

v. *I/O processor*

UNIVAC LARC (started 1954, delivered 1960) [Cu64,Ec56,Ec59]. High-level request packets (e.g., record number or key) were sent to an I/O processor, which also performed services such as device queueing. The requesting processor was interrupted when its request was complete.

2. *conditional instruction to start transfer*

a. *synchronization by polling*

(example unknown)

b. *synchronization by interrupt*

IBM S/360 (1964) [Am64,Be71,Bl64,Bl83]. The Start I/O instruction set the condition code according to the success of initiating the transfer (e.g., the path might have been busy and thus CPU had to queue the request, or an error might have exist). Channel I/O was the method of transfer, but a less complex channel instruction set was provided than that for 7090 channels.

3. *mailbox deposit to start transfer (i.e., single entry)*

a. *synchronization by polling*

CDC 6600 (1965) - virtual PPUs [Be71,En74,Sa80,Th64,Th70,Wi76]. In the typical CDC OS structure, PPUs were assigned to devices and polled reserved main memory locations (input mailboxes) to look for I/O requests for those devices. After starting a device, a PPU would poll the device until completion and would then place a completion notice in its output mailbox. Programs running on the CPU could poll the output mailbox; otherwise, they could be suspended until the PPU running the OS sees the completion notice and resumes the program by an exchange jump. Before an output transfer, the PPU must move the data from the shared main memory to its local memory; likewise, after an input transfer, the PPU must move the data from its local memory to the shared main memory. The execution of ten virtual PPUs was accomplished by time-sharing a single execution unit.

b. *synchronization by interrupt*

(example unknown)

4. *queue insert to start transfer (i.e., multiple entries)*

a. *synchronization by polling*

Burroughs B7700 (1972) [Do79,En74,Sa80]. Reserved locations exist in main memory that define head and tail pointers to I/O device request queues and I/O completion block queues. Queue manipulations by the CPU and I/O modules were atomic actions. Any IOM could handle any device, but a start I/O instruction issued by the CPU begins IOM processing on a specified device queue. IOM processing continues until an error, interrupt, or empty queue. The CPU polls the completion block queue, or, optionally, interrupts could be generated on completion of each request.

(see also IBM S/370 XA where path busy queueing was handled by the channel subsystem)

b. *synchronization by queueing*

Honeywell Series 60 Level 64 (1974) [At74]. In this French-designed minicomputer, microcoded semaphore operations were used in I/O processing. On I/O completion, the controller inserted a completion message into a queue and signaled the corresponding general semaphore.

ELXSI System 6400 (1987) [Ol85]. This system used message passing as a synchronization mechanism between OS processes and I/O controllers. An I/O processor notified the controller that a message was pending, but it was the responsibility of the controller to handle queues, including out-of-order processing and error handling.

c. *synchronization by interrupt*

(see also IBM S/370 XA where path busy queueing was handled by the channel subsystem)

5. *asynchronous instruction to start transfer*

a. *synchronization by polling*

(example unknown)

b. *synchronization by interrupt*

IBM S/370 (1970) [Ca78,Sa80]. SIOF (start I/O fast release) was used to release the CPU after a channel had fetched its CAW but before the channel had determined if the I/O operation could be successfully initiated or not. An interrupt occurred later if the device or path was busy. The designers assumed these conditions would be infrequent, but on later systems the interrupt overhead canceled out any performance gain from the fast release of CPU.

II. MULTIPROCESSOR I/O

A. *Asymmetric initiation*

1. *synchronization by polling*

(example unknown)

2. *synchronization by asymmetric interrupt*

Burroughs B5500 (1964) [Be71,Bl83,En74]. This system provided up to two CPUs, but only the master CPU could initiate I/O. An ITI instruction was provided to test for pending interrupts at end of interrupt handling; this prevented unnecessary context switching. The I/O channels and CPUs were crossbarred with memory modules; also, the I/O channels were crossbarred with all the peripherals.

IBM S/370 MP (1974) [Ca78,En74,Sa80]. Channels and devices were dedicated to a particular CPU.

3. *synchronization by symmetric interrupt*

UNIVAC 1100 Model 80 (1976) [Bo78,Bo79,Sa80]. This system provided up to four CPUs, but I/O had to be initiated by either of the two CPUs that connect to the storage interface unit that controls memory access for the I/O device. I/O is directed to the cache in the SIU rather than directly to main memory. I/O interrupts were made available to the two CPUs in alternation and for a limited amount of time each; if one CPU doesn't respond to the interrupt within the available period, the interrupt was passed on to the next CPU in sequence.

B. *Symmetric initiation*

1. *synchronization by polling*

Plessey System 250 (1972) - memory-mapped I/O with capability protection [Co72,En72a,En72b,Ha72,Le84]. A design philosophy of reliability and security led the designers to reject I/O channels in favor of additional CPUs, and to reject interprocessor and device-to-processor interrupts. This approach simplified the problem of component sparing, reduced the disruptions from unplanned external events, and eased the problem of hardware isolation in the case of component failure. Device drivers obtained I/O requests from memory queues and polled device registers until transfers were complete. An interrupt-like system was also available in which each processor periodically (100 microsec.) examines a common status word for interrupt-like requests; however, various papers differ on its use in I/O.

(see also B-7700)

(see also IBM S/370 XA option of masking off subclasses and using Test Subchannel)

2. *synchronization by queueing*

Intel 432 (1981) - a layered, intelligent peripheral subsystem and object-oriented design [In81]. GDPs (i.e., CPUs) would request an I/O operation by sending a message object to a device request port object. An I/O process on an interface processor (IP) would own this port and would respond by sending a message to an attached processor (AP) (i.e., placing the necessary information in the local memory of the AP and interrupting the AP). The IP had responsibility to protect the 432 core system logically using capabilities and physically using separate busses, while the AP was a more conventional microprocessor (e.g., 8086) that might use polling, byte-at-a-time interrupts, or DMA controllers for the actual I/O transfer. The device driver on the AP formatted a reply message from the I/O buffers in its local memory and sent it to the I/O process on the IP, which then sent a message object to the corresponding device reply port object.

(see also ELXSI System 6400)

3. *synchronization by asymmetric interrupt*

   Ramo-Wooldridge RW-400 (1960) [Be71,Cu64,En74,Po60]. Multiple computers (CPUs with local memories) could connect over a crossbar exchange to specialized processors called buffer memories, and from there to any one of multiple I/O controllers. Interrupts were available, but the papers are unclear as to whether the connection between a buffer memory and I/O controller must be maintained for an interrupt to be sent. Curtin states that connection requests could only be made by the computers or buffer memories, but that a computer could request that a buffer memory start I/O operations and then later transfer data from the buffer memory into its local memory [Cu64]. Enslow indicates that a complete system was never built [En74].

   Univac 1108-MP (1967) [En74,Sa80,St67]. Any CPU could initiate I/O operations, but interrupts were directed to a single prespecified processor.

   GE-655 (1969, later renamed Honeywell 6000) [En74,Sa80]. Any CPU could initiate I/O operations, but interrupts were directed to a single control processor (which was determined by manually set switches).

4. *synchronization by symmetric interrupt*

   Burroughs D-825 (1960) [An62,Be71,En74,Th63]. All interrupts were transmitted to each processor; an OS-controlled mask register in each processor determines if it would respond to a given interrupt.

   IBM S/360 Model 67 (1966) [Gi66]. I/O handling on this dual processor system was similar to D-825.

   IBM S/370 XA (1983) - subchannel per device [Co83,Do79]. Any CPU could start I/O on any device, and any CPU could accept an interrupt. Optionally, interrupt requests from subchannels could be assigned to one of eight maskable interruption subclasses, and priority schemes could be programmed so that certain high priority programs could be interrupted by only a small number of subclasses. If all CPUs mask off a certain subclass, the interruption status was held pending in the channel system and could be accepted later by use of the test subchannel instruction. A test pending interruption instruction was also available and was used to avoid an immediate context switch after a LPSW was executed by the interrupt handler. Path busy queueing was handled by the channel subsystem.

   Data General MV/20000 (1985). Any processor could start I/O on any channel. Channels either send interrupts to a processor identified by an OS-set register in the channel, or according to device directed interrupt mode, which uses an OS-controlled table in main memory to map device numbers to processor numbers.

   Sequent Balance (1986) - intelligent interrupt bus [Th88]. The SLIC bus interrupted the processor currently running the program with least priority.

Additional Considerations

An additional categorization under synchronization by interrupt might be the determination of the interrupt handler location. For example, the PDP-8 had one interrupt location, while the PDP-11 provides a single, large interrupt vector table yielding the possibility of hardware identification of a separate interrupt handler for each of several I/O devices. Alternatively, the RCA 601 required that the user specify the address of the appropriate interrupt handler as part of the Start I/O instruction [Li60].

Three other issues might also deserve explicit categorization:

- the recognition of unsolicited input (e.g., PDP-8) versus locking each input unit until a read is issued (e.g., IBM S/360)
- in systems with cache memory, I/O to cache (UNIVAC 1100/80) versus I/O to memory (S/370)
- in systems with virtual memory, I/O controllers with virtual address mapping (Honeywell DPS-8) versus I/O controllers requiring premapped physical addresses (S/370).

Acknowledgements

I wish to thank Randolph Bentson, Hank Dietz, Dan Kern, Philip Koch, Jim Haynes, John Levine, Barry Margolin, Robert Olson, Dan Pierson, Marv Rubinstein, Dan Siewiorek, Jan Stubbs, Chris Thomson, and Steve Wilson for their corrections and suggestions during the preparation of the original paper.

References

[Am64] G.M. Amdahl, G.A. Blaauw, and F.P. Brooks, Jr., "Architecture of the IBM System/360," IBM J. Research and Development 8, 2 (April 1964) 87-101.

[An62] J.P. Anderson, S.A. Hoffman, J. Shifman, and R.J. Williams, "D-825 - A Multiple Computer System for Command and Control," in Proc. AFIPS FJCC, 1962, pp. 89-96.

[As57] M.M. Astrahan, B. Housman, J.F. Jacobs, R.P. Mayer, and W.H. Thomas, "Logical Design of the Digital Computer for the SAGE System," IBM J. Research Development 1, 1, (January 1957) 76-83.

[At74] T. Atkinson, "Architecture of Series 60/Level 64," Honeywell Comp. J. 8, 2 (1974) 94-106.

[Ba81] C.J. Bashe, W. Buchholz, G.V. Hawkins, J.J. Ingram, and N. Rochester, "The Architecture of IBM's Early Computers," IBM J. Research Development 25, 5 (September 1981) 363-375.

[Ba86] C.J. Bashe, L.R. Johnson, J.H. Palmer, and E.W. Pugh, IBM's Early Computers. Cambridge, MA: The MIT Press, 1986.

[Be78] C.G. Bell, J.C. Mudge, and J.E. McNamara, Computer Engineering: A DEC View of Hardware Systems Design. Bedford, MA: Digital Press, 1978.

[Be71] C.G. Bell and A. Newell, Computer Structures: Readings and Examples. New York: McGraw-Hill, 1971.

[Bl64] G.A. Blaauw and F.P. Brooks, Jr., "The Structure of System/360, Part I - Outline of the Logical Structure," IBM Systems J. 3, 2 (1964) 119-135.

[Bl83] G.A. Blaauw and F.P. Brooks, Jr., Computer Architecture, 1983 manuscript.

[Bo78] B.R. Borgerson, M.L. Hanson, and P.A. Hartley, "The Evolution of the Sperry Univac 1100 Series: A History, Analysis, and Projection," Comm. ACM 21, 1 (January 1978) 25-43.

[Bo79] B.R. Borgerson, M.D. Godfrey, P.E. Hagerty, and T.R. Rykken, "The Architecture of the Sperry Univac 1100 Series Systems," in Proc. Intl. Symp. Comp. Architecture, April 1979, pp. 137-146.

[Bo69] J. Bouvard, "Operating System for the 800/1800," Datamation 10, 5 (May 1964) 29-34.

[Bu53] W. Buchholz, "The System Design of the IBM Type 701 Computer," Proc. TRE 41, 10 (October 1953) 1262-1275.

[Bu62] W. Buchholz (ed.), Planning a Computer System. New York: McGraw-Hill, 1962.

[Ca78] R.P. Case and A. Padegs, "Architecture of the IBM System/370," Comm. ACM 21, 1 (January 1978) 73-96.

[Co62] E.F. Codd, "Multiprogramming," pp. 77-153, in F.L. Alt and M. Rubinoff (eds.), Advances in Computers, Vol. 3. New York: Academic Press, 1962.

[Co83] R.L. Cormier, R.J. Dugan, and R.R. Guyette, "System/370 Extended Architecture: The Channel Subsystem," IBM J. Res. Dev. 27, 3 (May 1983) 206-218.

[Co72] D.C. Cosserat, "A Capability Oriented Multi-Processor System for Real-Time Applications," in Proc. Intl. Conf. Computer Communications, Washington, DC, 1972, pp. 282-289.

[Cu64] W.A. Curtin, "Multiple Computer Systems," pp. 245-303, in F.L. Alt (ed.), Advances in Computers, Vol. 4. New York: Academic Press, 1964.

[Do79] R.W. Doran, Computer Architecture: A Structured Approach. New York: Academic Press, 1979.

[Du83] R.J. Dugan, "System/370 Extended Architecture: A Program View of the Channel Subsystem," in Proc. Intl. Symp. Comp. Architecture, 1983, pp. 270-276.

[Du56] S.W. Dunwell, "Design Objectives for the IBM STRETCH Computer," in Proc. EJCC, 1956, pp. 20-22.

[Ec52] J.P. Eckert, J.R. Weiner, H.F. Welsh, and H.F. Mitchell, "The UNIVAC system," in Proc. Joint AIEE-IRE Conf., Philadelphia, February 1952, pp. 6-14.

[Ec56] J.P. Eckert, "UNIVAC Larc, The Next Step in Computer Design," in Proc. EJCC, 1956, pp. 16-20.

[Ec59] J.P. Eckert, J.C. Chu, A.B. Tonik, and W.F. Schmitt, "Design of Univac-LARC System: I," in Proc. EJCC, 1959, pp. 59-65.

[En72a] D.M. England, "Architectural Features of System 250," in Operating Systems: Intl. Computer State of the Art Report, Vol. 14. Maindenhead, England: Infotech, 1972, pp. 395-427.

[En72b] D.M. England, "Operating System of System 250," in Proc. Intl. Switching Symp., Cambridge, MA, June 1972, pp. 525-529.

[En74] P.H. Enslow, Jr. (ed.), Multiprocessors and Parallel Processing. New York: John Wiley and Sons, 1974.

[Ev51] R.R. Everett, "The Whirlwind I Computer," in Proc. Joint AIEE-IRE Computer Conf., Philadelphia, December 1951, pp. 70-74.

[Fo57] J.W. Forgie, "The Lincoln TX-2 Input-Output System," in Proc. WJCC, 1957, pp. 156-160.

[Gi66] C.T. Gibson, "Time-Sharing in the IBM System 360: Model 67," in Proc. AFIPS SJCC, 1966, pp. 61-78.

[Gr57] J.L. Greenstadt, "The IBM 709 Computer," in New Computers, Report from the Manufacturers ACM Conf., 1957, pp. 92-98.

[Ha72] D. Halton, "Hardware of the System 250 for Communication Control," in Proc. Intl. Switching Symp., Cambridge, MA, June 1972, pp. 530-536.

[Ha60] S.D. Harper, "Automatic Parallel Processing," in Proc. Conf. Computing and Data Processing Society of Canada, 1960, pp. 321-331.

[Ha68] T.F. Hatch and J.B. Geyer, "Hardware/Software Interaction on the Honeywell Model 8200," in Proc. AFIPS FJCC, 1968, pp. 891-901.

[In81] Intel, iAPX 432 Interface Processor Architecture Reference Manual. Santa Clara, CA: Intel, 1981.

[Ja86] J.F. Jacobs, The SAGE Air Defense System - A Personal History. Bedford, MA: MITRE Corp., 1986.

[Le54a] A.L. Leiner, "System Specifications for the DYSEAC," JACM 1, 2 (April 1954) 57-81.

[Le54b] A.L. Leiner and S.N. Alexander, "System Organization of DYSEAC," IRE Trans. on Elect. Computers EC-3, 1 (March 1954) 1-10.

[Le84] H.M. Levy, "The Plessey System 250," chapter 4 of Capability-Based Computer Systems. Bedford, MA: Digital Press, 1984.

[Li60] A.T. Ling and K. Kozarsky, "The RCA 601 System Design," in Proc. EJCC, New York, December 1960, pp. 173-177.

[Lo59] N. Lourie, H. Schrimpf, R. Reach, and W. Kahn, "Arithmetic and Control Techniques in a Multiprogram Computer," in Proc. EJCC, 1959, pp. 75-81.

[Ma64] B.A. Maynard (ed.), "Honeywell 800 System," in Manual of Computer Systems. London: Gee and Company, 1964.

[Ol85] R.A. Olson, "Parallel Processing in a Message-Based Operating System," IEEE Software 2, 4 (July 1985) pp. 39-49.

[Po60] R.E. Porter, "The RW-400 - A New Polymorphic Data System," Datamation 6, 1, (Jan./Feb. 1960) 8-14.

[Sa80] M. Satyanarayanan, Multiprocessors: A Comparative Study. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[Se62] R. Serrell, M.M. Astrahan, G.W. Patterson, and I.B. Pyne, "The Evolution of Computing Machines and Systems," Proc. IRE 50, 5 (May 1962) 1039-1058.

[St67] D.C. Stanga, "Univac 1108 Multiprocessor System," in Proc. AFIPS SJCC, vol. 30, Atlantic City, NJ, April 1967, pp. 67-74.

[St52] L.D. Stevens, "Engineering Organization of Input and Output for the IBM 701 Electronic Data-Processing Machine," in Proc. Joint AIEE-IRE-ACM Computer Conf., New York, December 1952, pp. 81-85.

[Sv59] J. Svigals, "IBM 7070 Data Processing System," in Proc. WJCC, San Francisco, 1959, pp. 222-231.

[Th88] S. Thakkar, P. Gifford, and G. Fielland, "The Balance Multiprocessor System," IEEE MICRO, (February 1988) 57-69.

[Th63] R.N. Thompson and J.A. Wilkinson, "The D825 Automatic Operating and Scheduling Program," in Proc. AFIPS SJCC, 1963, pp. 41-49.

[Th64] J.E. Thornton, "Parallel Operation in the Control Data 6600," in Proc. AFIPS SJCC, 1964, pp. 33-44.

[Th70] J.E. Thornton, Design of a Computer: The Control Data 6600. Glenview, IL: Scott, Foresman and Co., 1970.

[We52] H.F. Welsh and H. Lukoff, "The Uniservo - Tape Reader and Recorder," in Proc. Joint AIEE-IRE-ACM Computer Conf., New York, December 1952, pp. 47-53.

[Wi76] R. Wilson, "CDC SCOPE 3.2," in R.M. McKeag, R. Wilson, and D. Huxtable, Studies in Operating Systems. New York: Academic Press, 1976.

[History page] [Mark's homepage] [CPSC homepage] [Clemson Univ. homepage]

mark@cs.clemson.edu