*Note: As the standard form of my history, I first cover my personal adventures and history for this period and then in a second section I cover selected technical details of the major computers that I encountered in this period.*

*Also, for my convenience I have mostly not provided HTML links or references to various topics/references that are easy to find. Only unique or rare references are provided.*

# IBM Rochester 1971–1974

**I transferred to IBM Rochester, and for approximately two years I was the second-line manager of the team developing the IBM System/32 operating system. I personally implemented parts of the OS in System/32 "microcode".**

Sometime in early September 1971, Peggy and I finished our honeymoon in California and arrived in Rochester MN to start our new life.[1] Amazingly, our two cats were still with us. Rochester was a small town of about 60,000 in the south of Minnesota. Employment in the area was dominated by the Mayo Clinic, and to a lesser extent by the IBM facility of about a few thousand people. For IBM plants, Rochester was an anomaly since the town was small and isolated, but there is an interesting story about why IBM had a major facility in Rochester. [2]

For a while, we lived in a motel room on the south part of town near Rochester's iconic water tower, shaped and painted to look like an ear of corn. Peggy said that a corn totem gave her a bad feeling about life in Rochester. Since living in a motel room with two cats was not much fun, we really needed to get a more permanent residence. With special help from the IBM Credit Union (who effectively loaned us both the down payment as well as the principle), we were quickly able to able to buy our first house closer to the IBM facility in north Rochester.



There was some important background that I didn't tell in the last chapter. When I was given the "can't refuse" offer to move from Boca Raton to Rochester, I mentioned that I was engaged to be married and my fiancée was an IBM programmer. The answer was "no problem, we'll also get her a good

[1] At the end of the last chapter, I said that I would not be including any personal stuff about my family. I have changed my optinion on this topic and talked with my wife and we decided to add some family stuff in this chapter.

[2] A good history of the IBM Rochester facility including the System/3 and subsequent products is found at www.ibmsystem3.nl/publications/IBM_Rochester_History.pdf. Some of my Rochester work is mentioned in the history.

Copyright © 2026 G. Glenn Henry

job in Rochester." Now we had arrived and Peggy's new job was… nobody seemed to know. We should have pinned down her job before we left Boca because it seemed that nobody communicated the "good job in Rochester" concept to IBM Rochester. Finally, they placed her in a "bad" job: software maintenance of the RPG compiler. Peggy knew nothing about RPG, nothing about compiler technology, and software maintenance is generally an undesirable job: you are trying to debug and fix another person's code, usually with no documentation. Maintenance was particularly difficult in those days since all the code was all in assembly language, and the RPG compiler was a particularly bad example of tangled and poorly documented assembly language. This job was another bad sign to Peggy about Rochester.

And soon the worst sign arrived: the snow and cold of winter. I was from the Bay Area in California and Peggy was from Tennessee, Kentucky, and, most recently, Florida. While we conceptually knew about Rochester weather, we were not prepared for the many days of bitter cold—often there were weeks where the temperature never rose above 0°F—and the hassles of getting to and from work on icy roads. A symptom of the problem was that IBM had a service truck roaming the parking lots at work to help start the frozen engines after a day of below zero weather. We had to use this assist several times.

So, terrible weather and a bad job for Peggy—not a great start! I, like Peggy, didn't like the weather, but unlike Peggy, I had a good and interesting assignment that was clearly going to lead to even better jobs. Thus, in spite of our joint dislike of Rochester, it would be almost nine *long* years before we would leave. The reason for this delay was two projects I worked on, the second of which got me a large corporate award and was instrumental to my being later appointed an IBM Fellow.

My new job was to be the second line manager of the group developing the operating system—the *System Control Program (SCP)*, in IBM terminology—for a new small computer system, ultimately called the IBM System/32 when it was announced in 1975. My new manager was Bob Webster, who managed all System/3 software development. He worked for Harry Tashjian, the "system manager" of all System/3 product development. In the early 1960's, Bob had been a programmer working on an important IBM computer called Stretch (aka, the IBM 7030), and he had managed the System/3 software development from its inception.

The new IBM System/32 was a single-user business-oriented computer that was considerably less expensive than the System/3 Model 6 (some models leased for less than $1,000 per month). The target applications were the usual business written primarily in RPG (billing, inventory control, accounts receivable, sales analysis, payroll, and so forth). However, the other System/3 models were batch oriented and the System/32 was oriented toward more interactive usage with customers having little computer experience.

The system comprised a processor, fixed-disk storage, a data diskette, a printer, and an operator console with a small visual display screen. The System/32's tiny display contained six lines of 40 characters each that were primarily used to give prompts to the operator who was often just entering data into the application. Main storage was available in sizes from 16K to 32K bytes. A diskette contained around 200 KB, and the fixed disk came in sizes from approximately 3 to 13 MB. There were optional bi-sync and serial communication adapters and two types of printers. One type of printer was a very high-performance line-printer that fit into the same area as the more typical serial console printer.

Copyright © 2026 G. Glenn Henry

From an application development viewpoint, the System/32 appeared to have the same instruction set as the System/3 models and the SCP functions were similar to that of the older System/3's. The primary application language was the System/3 job control language (OCL) and the RPG programming language. However, significant changes were required to the SCP to support the new I/O devices and the operator-interaction features.

Here is a picture of the System/32 along with the management team of the entire System/32 software development: the SCP, and languages and utilities. I am the seventh from the left (with a beard, unusual for IBM managers in those days). Several of these managers worked for me. The main one of these is fifth from the left. Embarrassingly, I forget his name but I vividly remember watching the Superbowl that year (1972) at his house where my recent "home team" completed a perfect season.




Note how small the System/32 was compared to the original System/3 systems! Also note its tiny display to the left of the printer.

My main official job was management of the many aspects of the SCP development, which included several managers under me and about 20-30 people total. I remember only a few details of work during this period. Most of the SCP code comprised (1) changes to the batch-oriented System/3 SCP to be interactive with the operator, and (2) support for the new System/32 I/O devices. In addition to this SCP code (written in System/3 assembler, of course), we also worked on the many things needed to produce an IBM product: testing, documenting, developing the service strategy and materials, and so forth. All IBM products—hardware and software—were supported by service personnel called Customer Engineers (CEs) and other related titles. A new product required lots of new documentation describing how it worked, what things could go wrong, how to fix those problems, and so forth. Thus, a representative of the service organization was assigned to my team to help develop that material and prepare whatever the service group needed (he's the third person from the left above). We often complained about the amount of documentation a product required, but in retrospect I think this focus was beneficial to the product development. In particular, the remaining IBM manuals for the System/32 are extensive, detailed, and very complete.

Even though this assignment was challenging and I learned a lot more about business applications and project and people management, my main interest was technical. Accordingly, I studied the hardware and software architecture to see where we could improve things from a technical aspect.

Even though the System/32 appeared to software to execute the System/3 instruction set, in order to reduce cost, the processor did not actually execute System/3 instructions. Also, to reduce cost the I/O devices were generally pretty "dumb", and a significant amount of compute was needed to manage I/O. Thus, the single processor (called the Control Storage Processor, or CSP) was optimized for low-level control of the I/O devices as well as the ability to emulate the System/3 ISA. The code executing in the CSP (called microcode) was contained in a *writable* control storage. The microcode included an emulator of the System/3 instruction architecture such that the apparent instruction set (to the SCP and other software) was that of the System/3 hardware processors, but the underlying

Copyright © 2026 G. Glenn Henry

hardware instruction set was very different. The fact that the control storage was writable was quite rare at that time and facilitated many later improvements such as adding (emulated) floating-point instructions.

As a side note, having a fully writable microcode memory and no ROM memory (the System/3 had no ROM) has a lot of benefit but adds complications, especially in this modern day where (almost) all processors boot from ROM code—where does the initial boot code come from? I'll leave that for the interested reader to discover. Also, off-topic but I'll note that having no ROM boot code is now required in certain high security processors.

Note that this concept of having an apparent instruction set quite different from the real hardware was a key factor in the success of the main IBM computer family—the System/360, which had many different hardware models all emulating the same System/360 instruction set but having different interals. Of course, microcode for the System/32 was much simpler than the typical "horizontal" microcode as implemented on most System/360 processors. The System/32 processor had what today we would call a RISC architecture. Instructions were 16-bits wide and addresses were 16-bits wide and instructions were very basic. This interesting "first RISC processor" is covered in more detail in the second section of this chapter.

Of course, the emulated System/3 ISA ran a lot slower than the inherent speed of the hardware. I remember having a long-running argument with Dr. John Rood about how slow the System/32 was. He compared the timing of an *emulated* System/3 decimal-add instruction to the System/3's *hardware* timing for that instruction; the ratio of System/32 timing vs. System/3 timing was something like 20-to-one. In defense of the System/32, I programmed a much faster decimal-add routine in the real processor instruction set (I'll call it microcode) and claimed that this was the proper measure of performance of the processor. I argued that the System/32 was actually a fast processor, but it was crippled by having to emulate the weird System/3 instruction set.

This argument motivated me to propose a key design innovation of the System/32 that I ended up personally coding: namely, implementing some components of the operating system in CSP microcode. The idea was to execute certain highly used operating system routines at the CSP execution speed rather than the much slower emulated System/3 instruction speed.

While this approach of moving "software" functions into the internal hardware instructions seems obvious now, I believe this was the first time this was done in IBM. At that time, most engineers were reluctant to allow programmers access to the internals of the processor. I think the reason for this was the historical separation of engineering and programming disciplines in IBM and the fact that most engineers had little or no experience with programming and programmers at that time generally knew little about hardware.[3] The new System/32 engineering manager, however, was Keith Slack, who had also transferred from Boca Raton when I did. With our background of working together on the design of the proposed GSD architecture, he donated a portion of the control store for my use. I believe the size was a few hundred words, but I do not remember exactly.

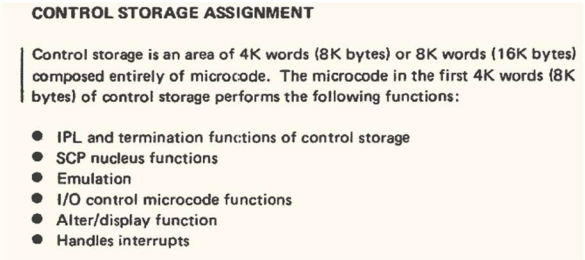
Accordingly, I rewrote some of the SCP's disk management routines into CSP microcode, writing the microcode myself. I vividly remember watching the 1972 Olympics at home nights while also

[3] Perhaps this comes from their separate education background: engineers took courses in the School of Engineering and programmers–in those days–primarily came from a "computer science" background. As a mathematician, both disciplines were equally foreign to me so it was easy for me to move between them.

Copyright © 2026 G. Glenn Henry

translating the System/3 code into System/32 microcode. This approach significantly improved the relative performance of the System/32.

IBM didn't advertise this approach of putting OS functions into "microcode", but it didn't hide it either. Here is an excerpt from manual SY21-0532-4_System32_System_Data_Areas_and_Diagnostic_Aids that shows that "SCP nucleus functions" are part of the microcode.



**CONTROL STORAGE ASSIGNMENT**

Control storage is an area of 4K words (8K bytes) or 8K words (16K bytes) composed entirely of microcode. The microcode in the first 4K words (8K bytes) of control storage performs the following functions:

- IPL and termination functions of control storage
- SCP nucleus functions
- Emulation
- I/O control microcode functions
- Alter/display function
- Handles interrupts

For the rest of my career, I constantly pushed for tighter hardware-software integration. In particular, on my next Rochester project, hundreds of thousands of lines of SCP function was implemented as "microcode".

There was no simulator of the hardware; the only way to debug code was on the hardware. Machine time was hard to get since there were few models available and the engineers were busy working on them. Thus, the time I that I got to use the hardware was mainly at night. I remember long nights at work sitting in front of a very early hardware model and debugging my code using the control panel (the only debug mechanism available). Due to the developing nature of the hardware, I also ran into a few bugs in the early hardware.

In addition to my management and personal coding duties, I was often given other assignments. One memorable example was related to a project Don Estridge was doing back in Boca Raton. He had what IBM called an "ad tech" group: a design group with no product responsibilities but with the focus of developing new technology in some area. At this time, Don's group was proposing a new type of operating system for the System/3 that would be object-oriented. Their approach had a name, which I cannot remember, but for purposes here, I will call it SOS.

Our system manager, Harry Tasjian, asked me to review the SOS proposal and make a report recommending whether the System/3 group should use this new concept. I read the SOS documentation and met or talked with Don's group several times. I then prepared a massive (maybe 100 foils) presentation that analyzed SOS in detail and concluded that it was a terrible idea for the System/3 product group.

My criticism had two general thrusts. First, the performance of SOS would be terrible on any foreseeable System/3 hardware. An object-oriented interface has large size requirements and large execution overhead, and the System/3 processors—including those in development at that time—were very limited in performance and memory capacity. Second, SOS was trying to solve a problem that did not exist in the System/3 world where applications were written in RPG and the operating system needs were simple. My recommendation to ignore SOS was accepted and the concept was forgotten about in the System/3 group (but, I think, this episode was not forgotten by Don as our relationship seemed strained after that [4]).

The irony here will be seen in the next chapter where I propose, "sell" and manage a high-level object-oriented operating system for a new IBM system intended to replace the entire System/3 family. The important differences, however, are that the hardware of the new system was *much* more

[4] But, later, after Don became a high-level corporate executive, he became my second-level manager and he was one of the key approvers of my appointment to IBM Fellow.

Copyright © 2026 G. Glenn Henry

powerful than the System/3 and the operating system requirements were *much* more extensive (for example, virtual memory, relational database, and security).

On the personal front, Peggy had become involved with the newly formed Rochester chapter of the National Organization for Women (NOW). Most of our friends were associated with NOW and Peggy later became president of the Rochester chapter. The most exciting news, however, was that by the middle of 1972 Peggy was pregnant.

The birth of our son, Kevin, was memorable in many ways. By April of 1973, Peggy had planned to quit working three weeks before the estimated due date. The Friday night of her last day at work, I got a call from my father saying that my mother had died. This was not unexpected as she had been ill for some time.

That night I was furiously getting ready to fly back to California the next morning for a funeral on Monday. The next morning, however, Peggy had a cramp in bed that caused her water to break. We rushed to the hospital to await Kevin's birth, which happened a seemingly endless (especially for Peggy!) 12 hours later. In the meantime, the weather turned bad and a massive snowstorm arrived cancelling all flights out of Rochester for the weekend! Sunday morning, I hastily assembled a crib that we had bought, then, while Peggy stayed at the hospital, I took a bus through the snow to Minneapolis to catch the last overnight flight to California. Monday, after attending my mother's funeral, I immediately flew back overnight to Rochester. Fortunately, Peggy was fine and Kevin was born healthy.

At this time, IBM's maternity leave was an unenlightened six weeks. Peggy wrote a "Speak-Up" to IBM management [5] saying that this was inadequate and should be changed to six months. She also requested that IBM provide day care at work. Naturally, IBM did not then change their policy, but they did give her three more weeks of leave. As an interesting twist, Peggy later became the Equal Opportunity manager for the entire Rochester facility (over 2,000 employess).

I also continued interacting with Atlanta GSD headquarters on various architectural issues. One memorable night in early 1974, I remember Dr. Rood visiting at our house and we watched Hank Aaron break the home run record.

Sometime around mid-1974, I was done with my efforts on the System/32 and was asked to take on a new assignment. The IBM System/32 was later formally announced and shipped in January 1975. Due to its small business orientation and low cost, the System/32 was *very* successful and it was shipped for another ten years, a long time given the rapid change in technologies. . By May 1978, the System/32 had surpassed the IBM System/3 as the most installed IBM computer and ultimately its total volumes were second only to the IBM PC volumes.

In early 1975, I received a $5,000 Outstanding Contribution Award for my work on the System/32 and got the first of my several invitations to the annual IBM Corporate Technical Recognition Event. The CTRE was a wonderful event IBM held every year to recognize special technical achievements and contributions. About 200-300 employees were invited and IBM paid for their spouses to go. The events were held at famous places and great hotels. The 1975 event was at the Waldorf-Astoria hotel in New York. In the mornings of two days, there were "business" lectures with the spouses invited. These actually had nothing to do with real business, but were motivational talks or entertainment.

[5] IBM had a policy called "Speak Up" encouraging employees to raise concerns about company behaviors without fear of reprisal.

Copyright © 2026 G. Glenn Henry

For example, over my several trips to the CTRE, I remember Dr. Ballard speaking about the discovery of the Titanic, the US men's gymnastics team that won the gold medal at the 1984 Olympics, some professor explaining plate tectonics, Dennis Conner, skipper of the winner of the America's Cup race, and so forth. The afternoons were free, but IBM arranged optional events such as tours, tickets to plays, and so forth.

While thankful for this recognition of my work on the System/32, by mid-1974 I had already moved on and was already consumed in the startup of what was to become my longest, most difficult, and most important IBM project.

Copyright © 2026 G. Glenn Henry

# The Computers 1971-1974

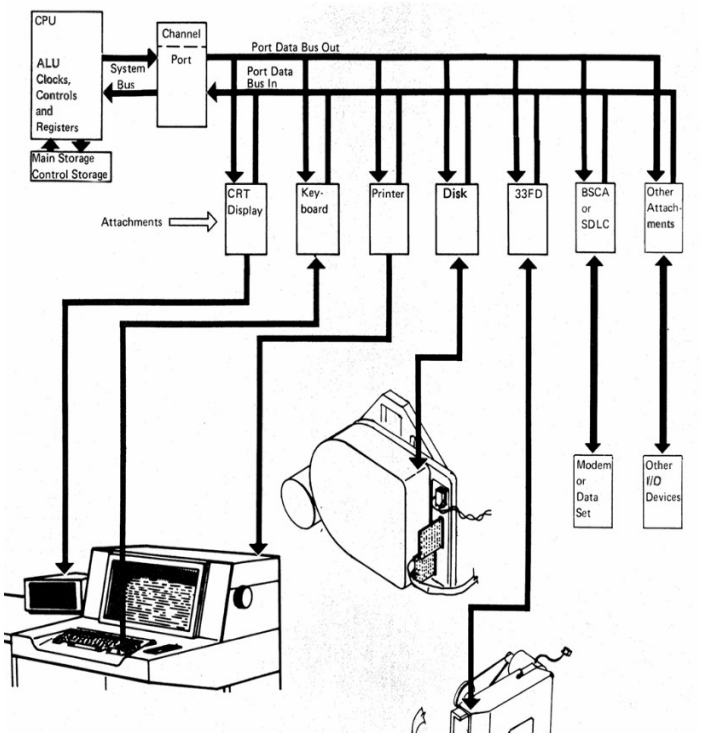
## References: A Real IBM System/32 & Bitsavers

This chapter of my work covers material from over fifty years ago. It's a wonder that anything survives from that period (including me) and a bigger wonder is that I have an instance of what I actually worked on! But I do: I have a complete IBM System/32! I wanted to take some pictures of the hardware specifically for this section but the system is hard to access and is currently completely covered up with other stuff. So, I'll refer to the online pictures from my collection of stuff which I optimistically call a "museum". The System/32 page and pictures are found here: https://www.glennsmuseum.com/items/s32/. As needed, I'll copy the relevant pictures to this document.

Even though I remember what projects I did during this time-frame, I certainly don't remember the technical details of the System/32 processor. But, we developed very good documentation that ended up in IBM manuals and the critical ones are saved by bitsavers, so those manuals are what I'll use to refresh myself. I won't credit specific ones unless the reference is not obvious.

## IBM System/32 Structure

My focus and interest are on the CPU and its associated memory. However, as with all IBM systems, the I/O logic is significant and pretty complicated, especially the fixed-disk control. In a picture included later, you can see that the I/O logic is at least as large as the CPU and memory area.




The "33FD" device (cut-off in this picture) is a diskette drive. The picture also misses a critical I/O device: there is a high-speed line printer (over 4 lines per second!) as well as the expected slower "ball" printer. They actually occupied the same space in front of the keyboard.

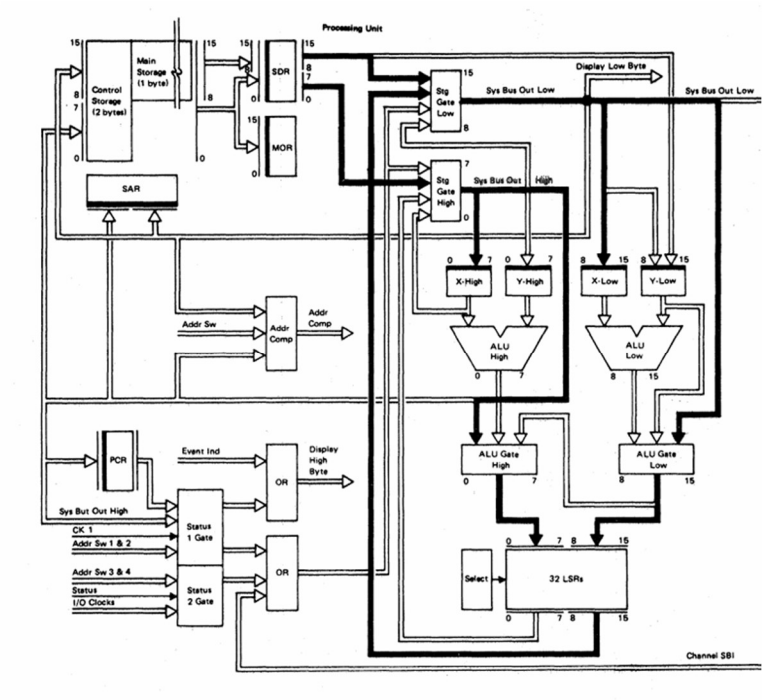
I will skip describing the I/O area further except to note that I/O devices could perform DMA, that is, directly reading and writing from both types of memory. That's not obvious from the CPU diagram shown next.

Copyright © 2026 G. Glenn Henry

# IBM System/32 Processor Microarchitecture

Here is a high-level diagram of the System/32's core processor. The instruction/control storage was 4K writable *16-bit words* (an extension of another 4K words was available and was used for instruction extensions such as floating-point.). The first 128 words of control storage have special control values and some are be directly accessed (via DMA) by various I/O components.

The "large" (up to 32KB) data memory was only 8-bits wide but the ALU operations work on 16-bit values as well as 8-bit values.




The tiny set of registers in the CPU is highlighted in this diagram. The SAR register is the address register for addressing both the data and instruction memories. The MOR register is where the read 16-bit instruction data is returned and then decoded to control the data flow shown here, and the SDR register is where the data memory value is returned. These data bytes or data from the Local Storage Registers (LSRs) are steered though select muxes to the two register inputs (X and Y) of each of the two byte-wide ALUs. The two ALUs can also be combined to do certain 16-bit operations. The outputs of the ALUs are placed in one of the 32 sixteen-bit (LSRs), which can feed their data back around into the ALUs. Derivatives of the results also go to the eight-bit PCR register which contains status about the ALU results that can be tested by branch instructions or fed back into ALU operations.

The LSRs are divided into four eight-register "pages" which are associated with interrupt levels. The first eight LSRs are associated with normal execution ("level-0") and machine check handling. The second eight are the stack of current microcode address and return addresses for each of the four levels of execution. The address register for each execution level is called the MAR and the return register is called the MAB. The third group of eight are dedicated to interrupt 1 (the disk drive), and the fourth group is used by interrupt level 2 (other devices).

So, at any time the processor is operating on one of three "execution levels" (0,1,2), such that the registers in use by that level are the eight main LSRs that associated with that level and the particular MAR and MAB for that level. Interrupts cause the levels to switch "instantly" so that execution can just continue with a new instruction address (MAR) and the eight data registers saved from the last execution at that level. This fast interrupt responsiveness is used to help reduce the amount of circuitry needed for I/O devices.

The CPU instructions other than dealing with I/O are all register-to-register or memory-register operations. The instruction functions available are pretty obvious from the dataflow picture, but the

Copyright © 2026 G. Glenn Henry

actual mapping to opcodes has a few strange/special functions in order to pack things into 16-bits. All instructions use a four-bit master opcode field (bits 0-3) with instruction-specific bits in 4:15. Following is a *very* brief description of the master opcodes and associated instructions. Note that I'm using big-endian notation since that's what the System/32 used:

**Branch(0000):** The next instruction to be executed is from the control storage at the 12-bit address contained in bits 4-15 of the instruction.

**Branch-and-Link(0001):** Same as the Branch instruction except that the current next address is stored in the associated memory-address-backup (MAB) location in the associated bank of LSRs.

**Branch-On-Condition(0010):** This instruction does a conditional branch: bits 4-7 define a condition and bits 8-15 replace the current 8-15 bits in the storage address register. That is, the instruction can branch to any of 256 locations within the current "page" of control memory.

**Branch-On-I/O (0011):** This instruction does a conditional branch based on some selected I/O condition. the instruction can branch to any of 256 locations within the current "page" of control memory.

**Storage(0100):** This complicated instruction has two major functions: (1) performs writes or reads of *byte* values (for main storage) or *word* values (for control storage) to-or-from either main or control memory., or (2) depending on an instruction bit, it transfer one byte to or from an I/O device and main storage.

**Test-Mask(0101):** This instruction ANDs a byte value from the instruction into either the low- or high-byte of the selected register in the current register bank. The PCR indicates whether the selected bits from the register are selected all ones, all zeros, or a mixture.

**ALU-1(0110):** This instruction performs ALU type functions on *one byte* (Hi or Lo for each as selected by two instruction bits). Three instruction bits select one-of-eight LSR registers (of the current bank) for operand one and another three bits selects one-of-eight LSR registers for operand two. The result is placed in the operand 1 register. Four instruction bits define the fourteen ALU functions to be performed. The operation performed are the usual add/subtract/logical functions, including add/subtract with carry. PCR bits are set to describe the status of the result (zero, carry, etc).

**ALU-2(0111):** This instruction performs ALU type functions on two bytes of R1 and one or both bytes of R2. Similar functions are performed as for ALU-1 instruction.

**Set-Bits-Off(1000):** This instruction sets selected bits to zero in either the low- or high-byte of the selected register in the current register bank. The bits to be modified are defined by a byte mask in the instruction. The PCR is not changed.

**Set-Bits-On(1001):** This instruction performs like the set-bits-off instruction except that the selected bits are set on.

**Load-Immediate(1010):** This instruction moves a byte value from the instruction into either the low- or high-byte of the selected register in the current register bank.

**I/O-Immediate(1011):** This instruction sends a a bunch of modifier bits from the instruction to the I/O channel and either sends a selected LSR register byte along or returns a value to the selected register from the I/O. That is, it can cause an I/O channel device to do something, or it can return data from the selected device. As mentioned earlier, the I/O world of the System/32 is complex and not of interest here.

Copyright © 2026 G. Glenn Henry

**Compare-Immediate(1100):** This instruction compares a byte value from the instruction with either the low- or high-byte of the selected register in the current register bank.

**Subtract-Immediate(1101):** This instruction subtracts a byte value from the instruction with either the low- or high-byte of the selected register in the current register bank.

**Move-LSR(1110):** This instruction moves a two-byte LSR value from one register to another. One address field is 5-bits in size and the other is 4-bits in size. The register addresses in these fields are relative to the interrupt level in which the processor is executing. That is, registers 8-15 (the MAR/MAB values for each interrupt level) can be moved to/from the eight registers associated with the current interrupt level.

**Hex-Branch/Move(1111):** This instruction either (depending on an instruction bit) performs a 16-way unconditional branch or performs a complex (to describe but simple to perform) operation on the "zone" digits of the two registers specified. The first case perfoms the 16-way branch based on the 4-bit value in either the high ("zone") or low ("digit") 4-bit portions of the selected LSR register. The branch is relative to the location of the hex-branch instruction itself. The second case is too special to describe here. This instruction reflects that fact that the System/32 applications required a lot of "zoned decimal" arithmetic operations.

# IBM System/32 Processor Programming

Programming the CPU was done, of course, in assembler, which was a modification of the main System/3 ISA assembler so things were very familiar. Here is the only code example that I could find (easily). It comes from the printer control microcode.

For example, the instruction at 0743, (62B6) decodes as ALU-1, reg-1 = 2, reg-2=6, and a function of low-byte ADD (result goes into reg-1). This makes sense per the comment.[6]

And, the next instruction (6CC2) tests this result by subtracting reg-8 from the result in reg-2 of the previous add. Then comes a branch if that subtraction was positive

```
                    3151 ********************************************************************************
                    3152 *    ENTRY POINT IF R BYTE OF 1 SHOULD BE FORCED                               *
                    3153 ********************************************************************************
0738                3154 PTCARRS1 EQU   *
0738 A601           3155          LI    WR6(L),1            FORCE R BYTE TO 1
                    3156 ********************************************************************************
                    3157 *    MAIN ENTRY POINT, INCLUDING INITIALIZATION OF PARAMETERS AND              *
                    3158 *    DETERMINATION OF THE REQUESTED OPERATION.                                 *
                    3159 ********************************************************************************
0739                3160 PTCARR00 EQU   *
0739 A748           3161          LI    WR7(L),X'48'        ASSUME DISABLE EOF SWITCH
073A 9820           3162          SBN   WR0(H),PTCGACTV     SET CARRIAGE BUSY AND INTRPT REQ
073B C600           3163          CI    WR6(L),0            CARRIAGE MOVEMENT SPECIFIED?
073C 2659           3164          JZ    PTCARR04            NO
073D E23F           3165          L     PTFLCL,WR2          FETCH LINE POSITION & FORMS LENGTH
073E E43F           3166          L     PTFLCL,WR4          LOAD UP WORKING COPY
073F 5102           3167          TM    WR1(L),PTIOBFSK     SKIP OPERATION?
0740 2448           3168          JO    PTCARR00            YES-PROCESS THE SKIP
                    3170 ********************************************************************************
                    3171 *    SPACE OPERATION PROCESSING                                                *
                    3172 ********************************************************************************
0741 C603           3173          CI    WR6(L),3            IS THE # OF SPACES SPECIFIED > 3?
0742 2458           3174          JP    PTCARR04            YES-NO SPACING
0743 62B6           3175          AR    WR2(L),WR6(L)       ADD # OF SPACES TO CURRENT LINE
0744 6CC2           3176          SR    WR4(H),WR2(L)       DID SPACE GO PAST FORMS LENGTH?
0745 2454           3177          JP    PTCARR03            NO
0746 62CA           3178          SR    WR2(L),WR2(H)       YES-SUBTRACT FORMS LENGTH
0747 0752           3179          B     PTCARR02            ENTERING NEW PAGE,ENABLE EOF
                    3181 ********************************************************************************
                    3182 *    SKIP OPERATION PROCESSING                                                 *
                    3183 ********************************************************************************
0748 6CC6           3184 PTCARR00 SR    WR4(H),WR6(L)       LINE # > FORMS LENGTH?
0749 2558           3185          JN    PTCARR04            YES-NO SKIPPING
                    3186 *         MVHR  WR2(L),WR6(L)       MOVE NEW LINE NUMBER
074A A200           3187          LI    WR2(L),0                CLEAR DESTINATION REGISTER
074B 62B6           3188          AR    WR2(L),WR6(L)           MOVE SOURCE TO DESTINATION
                    3189 *** END OF EXPANSION **
074C 66C4           3190          SR    WR6(L),WR4(L)       CALC NUMBER OF SPACES TO NEW LINE
074D 274F           3191          JFLG  **2 ----------+     SIO ENTRIES DO NOT SKIP IF RESULT=0
074E 0750           3192          B     **2 ----------|--+  FUNC. KEY ENTRY WILL SKIP F.L. IF=0
074F 2658           3193          JZ    PTCARR04 <---+  |   NO SPACE REQUIRED
0750 2454           3194          JP    PTCARR03 <------+   LINE IS ON THIS PAGE-PROCEED
0751 668A           3195          AR    WR6(L),WR2(H)       ADD FORMS LENGTH TO THE DISTANCE
                    3197 ********************************************************************************
                    3198 *    ROUTINE EXIT POINTS                                                       *
                    3199 ********************************************************************************
0752 D7F8           3200 PTCARR02 AI    WR7(L),X'08'        NEW PAGE,ENABLE EOF
0753 9880           3201          SBN   WR0(H),PTNEWPAG     INH18IT ANY PRT OPS
0754 D6FF           3202 PTCARR03 AI    WR6(L),1            HARDWARE REQUIRES DISTANCE+1
0755 8B86           3203          IOCL  WR6(L),X'B'         SEND # OF LINES TO PRINTER
0756                3204 PTCARR0N EQU   *                   ENTRY POINT FOR UPDATING PARAMETERS
```

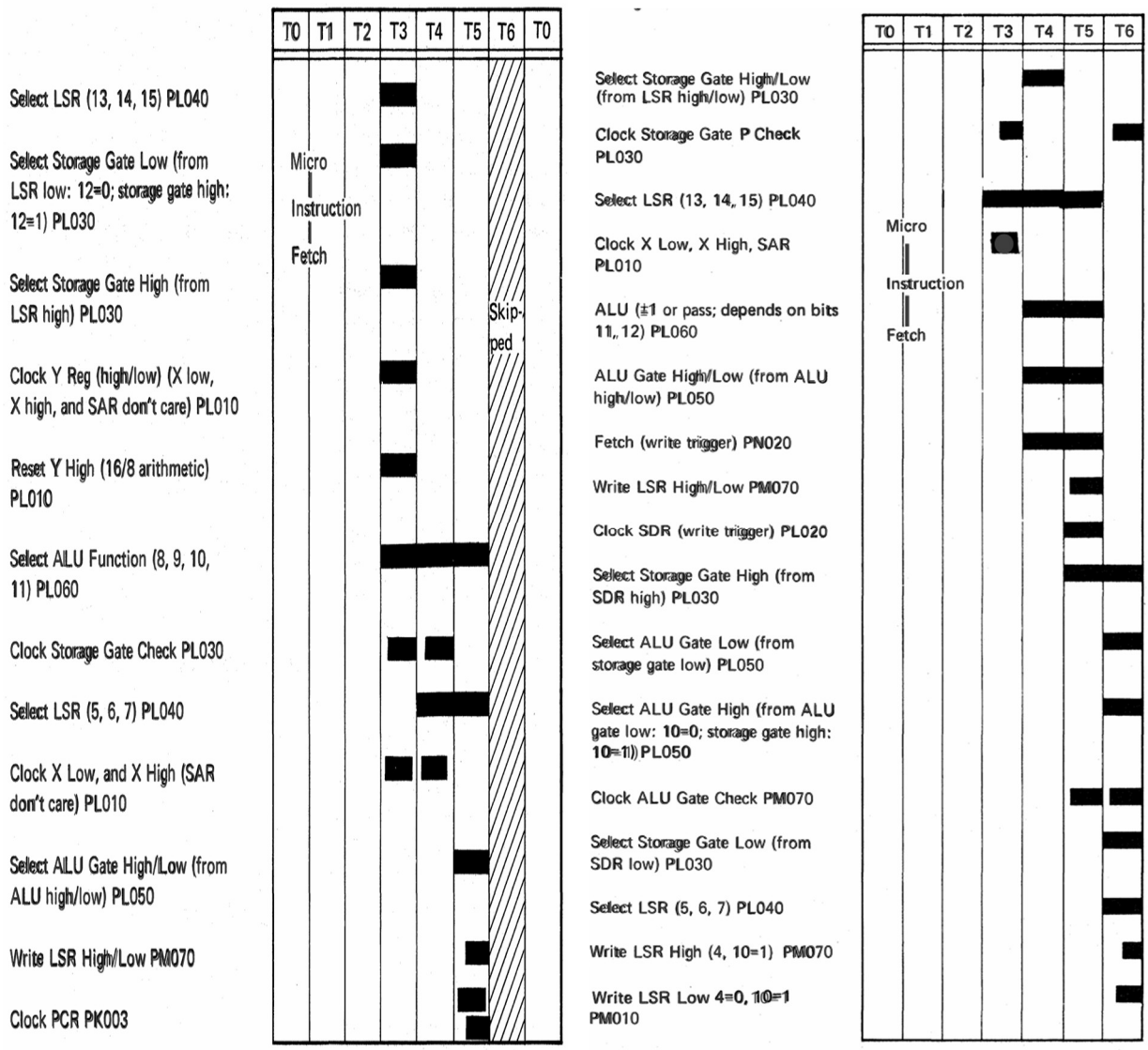
# IBM System/32 Processor Timing

The System/32 did no pipelining of instruction execution: an instruction was fetched, executed completely, then the next instruction was fetched, etc. As discussed for the processors in previous chapters, there was not a single synchronous clock that all registers used. Timing was driven by a master 200ns oscillator that was used to produce overlapping timing pulses, or gates, each several

[6] Note that when I joined IBM the rule was that *every* line must be commented! The protests of "stupidest rule ever" can still be heard echoing throughout the universe.

Copyright © 2026 G. Glenn Henry

200ns long in duration. Each instruction or operation used the timing pulses whenever it needed to gate the flow of data through the logic. For example, the next picture shows the timing of a ALU execution instruction (left) and a load instruction (right). *However, all instructions took five, six, or to seven pulses to execute, for an average of about 1.5µs per instruction.*

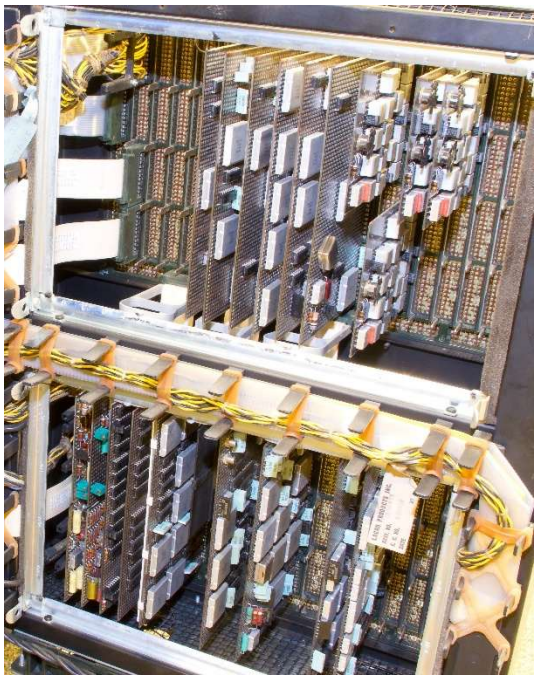



This instruction execution speed is about the same time as the 1.5us processing of a single byte by the System/3 hardware when executing System/3 instructions. This meant that emulation of System/3 was very slow compared to the hardware execution of the other System/3 processors. For example, consider a System/3 decimal-add instruction. The CSP emulator program had to read the System/3 instruction from memory, decode the instruction, and set up the loop for processing the number of digits specified in the instruction. For each byte processed, the emulator had to read two discrete bytes from main memory, perform the decimal add operation, which took multiple instructions (since the System/32 hardware had no decimal instructions), and write the result back to main memory, decrement the string count, etc.

Copyright © 2026 G. Glenn Henry

# IBM System/32 Hardware Technology

Here is a picture (cropped from my museum site) of the entire logic of my personal System/32. The bottom panel contains the logic for the various I/O devices, The CPU and memory are on the top. The six-card CPU logic starts on the first card from the left. The next two half-cards after the CPU itself provide the 4K of the 16-bit control storage (accessed 16 bits per access). Note that there is room for two more half cards giving a potential maximum of 8Kx16b of control storage.



At this point I would. Normally describe the particular IBM circuit technology used. And, in the System/32 case, the technology is new versus our last processor, the System/3. But, I'm spending too much time here and have got to move on.

# Is the System/32 ISA "RISC"?

Personally, I think the question of whether an ISA is "RISC" is meaningless and that term shouldn't be used the way it is in CS teachings. However… For this discussion I asked AI to give me the major bullet points from Hennessy & Patterson for what are the major characteristics of a RISC architecture. Following is a quick summary of the criteria it produced and my take on how the System/32 ISA matches these. I think my rating is pretty obvious from even the brief material I've presented.

**Load/Store Architecture.** Yes.

**Fixed-Length Instructions.** Yes.

**Single-Cycle Execution.** Yes, even loads from memory. Note that most ISAs that are called RISC don't come close to meeting this criteria!

**Simple Instructions & Addressing Modes.** Yes.

**Large General Purpose Register Set.** Depends: there are 32 registers available but a normal instruction can only reference a "page" of eight. Some special instructions can access all registers.

**Hardwired Control.** Yes.

And, a category that I personally add… Is the ISA easily usable by a non-engineer programmer? The System/32 answer in yes in that case, I proved it!

Copyright © 2026 G. Glenn Henry