IBM

International Technical Support Centers

IBM RT PC SYSTEM ARCHITECTURE

GG24-3024-00

IBM RT PC System Architecture

Document Number GG24-1024-00

April 1986

International Technical Support Center
Austin

First Edition (April 1986)
This edition applies to the first release of the
IBM RT PC system.

ABSTRACT

For IBM RT PC the System Architecture is described by giving a reasonably technical presentation of both the hardware and software structure. In addition several other issues are covered such as an introduction to RISC technology and the UNIX enhancements added to the IBM RT PC operating system environment.

This bulletin is addressed to IBM systems engineers who wish to acquire a strong foundation on the system architecture of the IBM RT PC. The background needed to understand it is being familiar with interactive operating systems and knowing some basic concepts of computer hardware. Familiarity with UNIX is a plus, but it is not a prerequisite.

PREFACE

This bulletin represents a compact presentation of both the hardware and software aspects of the system architecture of the IBM RT PC system.

This product combines:

- A very fast Reduced Instruction Set 32-bit processor for efficient execution of programs compiled from a high-level language.
- A resource manager that provides virtual machine, storage and I/O functions in order to assure data integrity and processing continuity.
- A multitasking, multiuser operating system based on the AT & T UNIX System V, but with a lot of IBM extensions allowing it to serve different user requirements.
- An 80286 Coprocessor feature that allows users to run programs written for the IBM Personal Computer without interfering with the normal native operation of the IBM RT PC.
- A wide variety of displays, printers, communications adapters and processing features are included in a system.
- All that can fit on or under a desk.

The structure of this bulletin is as follows:

- The first chapter: "Reduced Instruction Set Computers", covers the RISC topic, by showing the evolution of CPU technology and explaining what RISC technology is and why it is state-of-the-art.
- The second chapter: "IBM RT PC System Architecture Overview", covers the overall architecture of the IBM RT PC by giving equal emphasis to the hardware as well as to the software structure. It introduces the specific terminology and it allows the reader to better understand and combine the more in-depth treatment that follows.
- The third and fourth chapters: "The ROMP/MMU Processor Complex" and "Virtual Resource Manager (VRM) and Virtual Machine Interface (VMI)", cover in more detail, respectively, the hardware basis and the layered software architecture of the IBM RT PC.
- The appendices cover other topics, such as the Floating Point Accelerator option, the Coprocessor card, the system memory boards, the I/O adapters available and the UNIX enhancements implemented with the operating system of the IBM RT PC.
- A glossary with IBM and non-IBM references is also provided at the end of this bulletin.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Introduction to RISC

Before addressing why RISC technology is used on the IBM RT PC we shall first explain what RISC is. Since it is a relatively new concept most readers have probably never heard about it.

The abbreviation stands for Reduced Instruction Set Computers, and, as the term itself implies, we are talking about processors with only a basic set of instructions. This means that a RISC processor is not equipped with powerful instructions like the ones found in machines such as the IBM System/370 or even those at the microprocessor level.

Thus far we have only discussed the definition of RISC and it is imperative now to explain why this new concept exists in state-of-the-art computers like the IBM RT PC.

As VLSI technology advances, the new generation processor chips can implement the architecture of powerful minicomputers on a single chip. Such an implementation has to cope with certain purely technological constraints which engineers will not know how to overcome for several years. On one hand, the perspective of single-chip designs is rather limited by the resources one disposes of in terms of silicon real-estate, or more practically speaking, of the transistors and power dissipation the chip can handle. Typical designs today contain several tens (and at times even hundreds) of thousands of transistors.

On the other hand, the clear trend of current MOS technology is scaling down successful designs to narrower line widths. When a scaling takes place the devices on the chip become smaller and smaller. The distances electrons now have to travel are shrinking, and the devices become faster. Faster transistors, on a processor chip for example, immadiately means an increase in the processor's throughput.

Since transistors become smaller and smaller the communications between different parts of the same chip become crucial. The signal delays introduced by long wiring connections are not at all negligible and it actually becomes one of the most difficult problems to resolve.

This shows that intra-chip communications have to be carefully addressed and random logic as well as long-distance communications have to be kept to an absolute minimum.

One can now understand that it is not only a matter of generating a proportionally scaled layout (in order to successfully map on a chip the architecture of a mini or a mainframe). The constraints we have discussed have to be well studied, understood, evaluated and dealt with accordingly. The system partitioning in different chips as well as the allocation of certain chip areas has to be done prudently and judiciously.

In this context, IBM researchers have found that a reasonable restriction to a smaller set of instructions, combined with an architecture customized for fast execution of all the instructions in the set, can result in a machine with a surprisingly high throughput.

Such a Reduced Instruction Set Computer can be designed with a relatively small control section and a short machine cycle. The ROMP processor used in the IBM RT PC for example is characterized by a 170 nsec cycle. The design of such a processor is then much more straight-forward than traditional designs in the sense that both design time and the influence on the final result of the eventual architectural flaws are drastically reduced.

IBM has been directly involved in RISC research since 1975 when a team from the Yorktown Heights Research Center, led by George Radin, started experimenting with these ideas. The project was called "Project 801" from the site building number they were working in. An operational result was functional in 1979 based on standard off-the-shelf Establishment Communications Link (ECL) components, but the results were not made public for obvious reasons until 1982. Another project pursued by IBM engineers resulted in the ROMP processor, a very powerful 32-bit RISC microprocessor, which is the base of the IBM RT PC.

RISCs are not something only IBM has been working on. The University of California at Berkeley has developed two distinct designs; the RISC I in 1981 and the more sophisticated RISC II in 1983. A team from Stanford University has also recently introduced with the MIPS (Microprocessor without Interlocked Pipelined Stages) processor. Industry is also moving fast in this direction.

RISC Overview

Computer architecture started existing as a concept in 1964 when IBM introduced the System/360. It was the first time that a differentiation was made between the computer architecture - the abstract structure of a computer that a machine-language programmer needs to know in order to write programs - and the actual hardware implementation of that structure.

Before that, the criteria to rate a computer was the cost of an implementation. However, researchers now tried to come up with new measures of performance. The decreasing hardware cost gave birth to arguments for richer instruction sets. Rich instruction sets were said to simplify compilers, compensate for the rising cost of software development and even improve the architecture quality. The new means of evaluating architectures was program size and the design mentality that prevailed in the 1970's was that large programs are invariably slow programs.

This philosophy is responsible for the exotic instruction formats one can find in machines like the IBM 370/168.

The use of microcode in that generation of machines was very intensive. The reasons are very simple to understand. Memory costs were steadily decreasing and large microcoded modules would add almost nothing to the overall cost. Microinstructions were faster than normal instructions. Registers made the writing of compilers very difficult and this was why stacks or memory-to-memory architectures were adopted.

In the meantime the cache memory, or high speed buffer, was invented, yielding substantial improvement in the implementation speed of the architecture, while compilers started finding it difficult to generate those complex new functions. A special breed of compilers, the optimizing compilers, were removing so many of the unknown elements at compile-time that they almost never made use of the powerful instruction set at run-time.

When computers made the transition from physical to virtual memory, the microcode had to ensure that any routine could start over if any memory operand caused a page fault. The performance gained by microcode was lost by a tremendous overhead which incurred during swapping in a multiprocess environment. When each program has its own microcode, a multiprocessing operating system has to reload the Writable Control Storage (WCS) with the corresponding microcode. This reloading time ranges between 1,000 and 25,000 memory accesses depending on the machine.

This latter point led researchers to decide that future computers should have virtual control storage, which meant that page faults could occur even during microcode execution. The distinction between programming and microprogramming was becoming less and less obvious.

It was clear that the attempt to bridge the "semantic gap" by using writable control storage had led to a "performance gap". The motives were still valid: programmers should write operations that mapped directly to microinstructions, and instructions should be no faster than microinstructions. However, the caches allowed main memory accesses at the same speed microcode was accessed in control storage. Microcode no longer enjoyed a ten-to-one speed advantage.

It was this whole context that gave rise to a new computer design philosophy. Optimizing compilers could be used for the compilation of high-level programming languages down to the level of simple instructions of comparable efficiency to microinstructions, and to make the instruction cycle as fast as technology would allow.

These machines are characterized by fewer instructions - hence the name RISC - which in general execute in one cycle.

RISCs have set up a new set of architectural design principles:

- All functions should be kept as simple as possible unless there is an extraordinary reason not to do so.

- Microinstructions should not be faster than simple instructions because caches are built from the same technology as WCS.

- Software is not better if microcode is used, it only becomes more difficult to change. All the hardware primitives available to microcode have to be accessible to assembly language programming.

- Simple decoding techniques and pipelined execution are much more important than program size. This leads to simple instruction formats which do not cross word boundaries.

- Compiler technology should be used to simplify instructions rather than to generate complex instructions.

The last point is of crucial importance and it deserves explanation. RISC compilers will try at compile-time to remove as much work as possible so that simple instructions can be used. This means, for example, that such a compiler will try to keep operands in registers, so that simple register-to-register instructions can be used. A traditional compiler will try to discover the ideal addressing mode and the shortest instruction format to add the operands in memory. In general, RISC compilers favor register-to-register operations because operands kept in registers are easily reused without new memory accesses and address calculations. RISC compilers use only LOADs and STOREs to access memory so that operands are not implicitly discarded after being fetched, as happens in the memory-to-memory model of execution.

Several unique features of RISC processors such as delayed branches and loads are described in depth in this document (see the Index).

Introduction

The IBM RT PC is characterized by a totally new architecture and it is useful to talk a little about the goals identified right from the start of the development work.

The next generation of workstation systems had to be implemented using state-of-the-art technology both in hardware and software. This implied that the heart of the system should be a processor offering unprecedented capabilities in speed and functionality. Advanced memory management functions should be provided for and the system's physical size should be proportional to the potential of current integrated circuits technology. It also implied that an ingenious software structure had to be devised which would render the software applications practically independent of hardware configuration changes.

IBM RT PC designers wanted to assure a compatibility with the IBM Personal Computer architecture in order to provide the IBM RT PC user with the capability of using the vast amount of IBM and non-IBM applications which run on the PC.

The Research Oriented Mini Processor (ROMP) is a 32-bit IBM proprietary processor capable of about 4 MIPS (Million Instructions Processed per Second) execution speed and reduced instruction set. IBM researchers realized that instead of having a processor with very powerful instructions taking the system a lot of time to decode and being idle when branches and loads happened, they would rather develop a new processor, which would need less silicon real estate and which would only be armed by a basic instruction set. This would result in a system taking less time to run an application. The major development effort needs to be in coming up with really powerful compilers, which will be in position to fully exploit the capabilities offered by the processor. The IBM RT PC is the result of a joint effort development based on RISC system architecture work from Austin, the Yorktown Heights state-of-the-art compiler technology and the 2 micron silicon-gate NMOS process from Burlington, Vt.

Efficient compilers are possible now, where a lot of different optimization techniques are used in order to take advantage of the processor capabilities. For example the PL.8 compiler used internally for the IBM RT PC development generates code about 10% less efficient than good hand-written assembly.

Another concern was the addressability. Sixteen-bit computers are limited to addressing only 64K bytes or words. The only way to overcome this limitation is by using special hardware provisions or segment registers. However, it turns out that it is not that simple to handle objects larger than 64K even when using segment registers.

Accordingly, the following goals are set up:

- The processor should be an all-32 bit machine, where registers, addresses and data are all 32-bits long.
- Virtual memory capabilities are a must in such a powerful workstation environment.
- The addressable space should be linear and practically not limited.
- The architecture should ensure the compatibility with existing IBM PC and PC/AT applications.

Reducing the cost of a system means using state-of-the-art VLSI design and packaging. For silicon area reasons the 2 major subsystems, i.e. processor and memory management unit, have been implemented in 2 different chips (actually the MMU is slightly larger than ROMP).

The IBM RT PC development team had in mind that the system should be characterized by a remarkably high performance without using expensive memory. The 801 minicomputer designed by a team at the IBM Thomas J. Watson Research Center was one of the first to investigate the effectiveness of RISCs. By using directly off-the-shelf ECL MSI components it gave tremendously encouraging results and inspired other teams to go on and mplement different appoaches on the RISC issue. The 801 was using 2 caches, which could deliver an instruction word and a data word at each cycle. However such caches are extremely expensive for small to medium systems and hence the IBM RT PC designers opted for pipeline-techniques. More details are given later on in this bulletin at the discussion of the system boards ("Appendix C. System Memory Boards" on page 51) and in the chapter discussing ROMP and MMU more deeply ("The ROMP/MMU Processor Complex" on page 17).

The IBM RT PC is offering a 40-bit virtually addressable space and it supports real memory size up to 16Mbytes. There are some concepts from the System/38 that are used and the system architecture provides a means of controlling access to virtual memory sections lying within a page. This proves especially useful in database locking schemes.

The system architecture was meant to assure the effective integration of such a virtual memory processor with existing as well as with new 8-bit and 16-bit I/O adapters. The attachment of coprocessors for compatibility and performance enhancement was also considered a must and the open architecture should allow the user to install practically everything he would think would enhance the performance of the machine.

System Hardware Architecture Summary

The IBM RT PC system architecture is the same for both the desktop and the floor-standing model. The floor-standing model offers the maximum extensions because of more free slots, however most options are available to all models.

The IBM RT PC systems have a wide range of standard and optional hardware features. A system planar board is used with all basic circuits and I/O slots. Figure 1 on page 9 gives an illustration of the system architecture and its structuring. The I/O slots are designed in such a way that many existing IBM PC and PC/AT cards can also be plugged into the system. The operator keys-in the input from a 101-key keyboard or an optional 2-button mouse or even from a tablet-pointing device. Besides all the I/O slots there are specific slots in the system for each of the 32-bit system components. The ROMP processor and its Memory Management Unit lie on the processor card which comes with every model. On the same card lies also the ROM with the microcode used by the machine at IPL time. The Storage Control is also on the processor card, as well as the Error Correcting Code logic used by MMU when accessing the main memory. Data storage is provided on 5-1/4 inch hard disks and diskettes. One of the optional cards is the FPA (Floating Point Accelerator) card and it plugs into a separate 32-bit slot. Two other dedicated slots are used for system memory cards. Because of technological limitations, for the time being the only memory cards that are available are of 1 Megabyte and 2 Megabytes. This gives at this time a real storage capacity of up to 4 Megabytes. The hardware architecture however allows an expansion up to 16 Megabytes real memory.

The optional 80286 coprocessor card plugs in one unique slot and provides the ability to execute both IBM PC and PC/AT programs concurrently with native ROMP code. The I/O bus slots offer the possibility of using other coprocessor-options such as PC/AT memory cards and a mathematical coprocessor chip.

Several monochrome and color APA displays are supported by the system in addition to existing IBM PC displays and adapters. For computer-aided design applications a serial link adapter is provided allowing to attach an IBM 5085 graphics workstation in a host-based network.

Hardware Architecture

The IBM RT PC system contains a 32-bit microprocessor and memory management unit combined with a 16-bit I/O bus. In order to provide high performance 32-bit processing capabilities as well as compatibility with standard 16-bit I/O adapters a system partitioning had to be done. The interface between the 2 buses is being taken care of by the IOCC (Input Output Channel Converter). The microprocessor is implemented using RISC architecture with only 118 instructions and a full 32-bit data flow for both data and addresses. The performance is rated at about 1.5 to 2.1 MIPS and the majority of register-to-register operations execute in one 170 nanosecond cycle.

The translation mechanism from a 32-bit system address to a 40-bit virtual address is provided by a "single level store" architecture implemented by the MMU. The MMU also contains internal translation buffers which take care of the conversion between the 40-bit virtual address and the 24-bit real address. The MMU hardware also automatically reloads from the main memory space the translation buffers as needed. The ECC logic for system memory is contained in the MMU as well as some of the memory control logic and the IPL and power-on self test ROM. More details about the ROMP and the MMU chips can be found in "The ROMP/MMU Processor Complex" on page 17.

Figure 1 on page 9 shows the two distinct buses on the processor board. From that it is seen that on the processor card, except for the processor itself and its MMU, there are logic circuits accounting for adapting the high-speed 32-bit packet-switching processor bus to an asynchronous 32-bit bus processor to which directly connect the floating point accelerator card and the System Board IOCC (Input Output Channel Converter). The MMU interfaces with the system memory cards through a dedicated memory bus.

An interesting feature is that I/O channel timing does not affect the processor timing because the clock generation for the processor, its MMU and system memory is provided on the processor card. This will eventually allow the use of higher performance processor and memory cards as technology advances, without disrupting the I/O channel timing.

The optional FPA card can be attached to the 32-bit processor bus and provides much higher performance for floating point applications. The FPA card is based on the National Semiconductor NS32081 coprocessor and can operate totally independent from ROMP. The actual form of the FPA is relatively poor for such a machine and is evaluated to about 200,000 Whetstone Instructions per Second (200 KWIPS).

As already mentioned the system memory attaches to the processor card through two dedicated slots. The memory bus consists of a 40-bit data bus (32 bits of data plus 8 bits of (ECC) error correcting code) and a 24-bit address bus. The 24-bit addressing allows up to 16 Megabytes of real system memory. The ECC allows automatic detection and correction of all single bit system memory errors, as well as detection of all double bit errors.

The technology used is 256K dynamic RAM and on each memory card the system memory is two-way interleaved. This means that the available memory is always logically partitioned into two distinct banks. One bank contains only even addresses while the other contains only odd ones. This simple technique provides a very high system memory bandwidth of 23.5 megabytes per second (4 bytes per 170 nanoseconds) and renders cache architectures totally useless. This memory bandwidth is by far higher than the ones implemented in every other competitive machine of this class.

In order to adapt the 32-bit processor bus to a PC/AT-like I/O bus special conversion logic is needed. This logic is implemented on the system board. All the I/O channel support functions, such as the real time clock and timer, DMA controller and interrupt controller are also provided on the system board. Considerable effort was given to the task of keeping the IBM RT PC I/O channel as compatible as possible with the PC/AT I/O

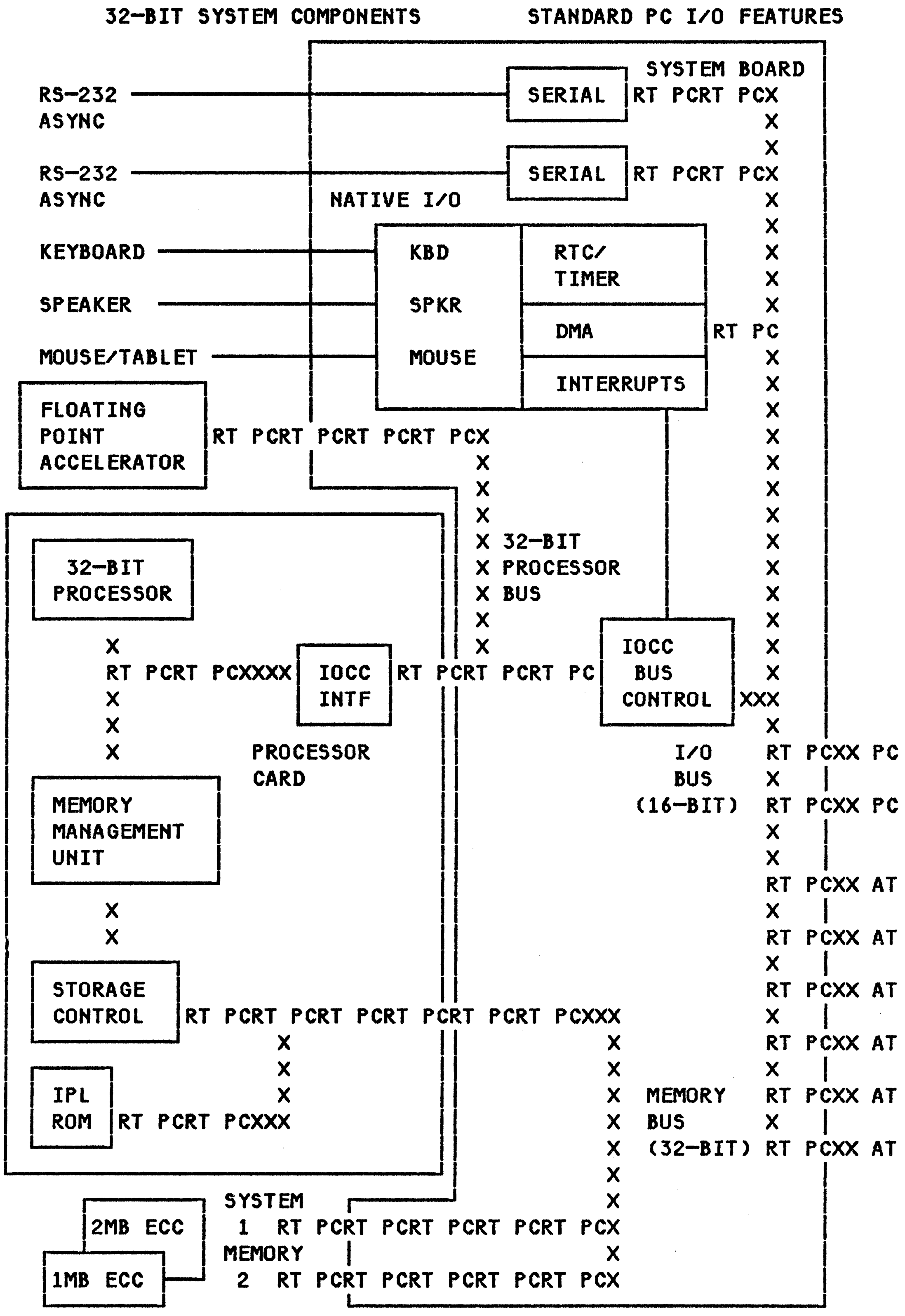

Figure 1. Overview of the hardware architecture

bus. Besides timings and address, DMA and interrupt assignments that had to be kept as close to those of the PC/AT as possible, there have been new provisions such as burst and buffered DMA and shareable interrupts in order to enhance the channel performance. The IBM RT PC system supports eight channels of Direct Memory Access (DMA) by using two Intel 8237 DMA controllers. A DMA Adapter is either a DMA device or an alternate controller. The terms are explained in the glossary at the end of this bulletin. DMA channels 0-3 support 8-bit devices or 16-bit alternate controllers. DMA channels 5-7 support 16-bit devices or 16-bit alternate controllers. DMA channel 8 supports only alternate controllers. This DMA channel is only available on the coprocessor slot. The coprocessor has the lowest DMA priority on the I/O channel. However when it is executing out of system memory or I/O channel, attached RAM, it can hold on to the channel for multiple cycles to improve performance. More details can be found in the "IBM RT PC Hardware Technical Reference Manual" (see "Appendix G. References" on page 63).

Timing and performance of the I/O channel is the same for both models, however the 6150 provides two 8-bit PC slots and six 16-bit PC/AT slots in distinction to the 6151 which provides one 8-bit PC slot and five 16-bit PC/AT slots.

Adapters on the I/O channel can easily access system memory through hardware facilities responsible for programmable translation control on the system board.

The system board contains a separate microprocessor which is handling the keyboard, the mouse, the tablet interface and the speaker. The 6150 includes two built-in RS-232 serial ports with DMA capabilities for the attachment of terminals, printers or other I/O devices.

The optional coprocessor card is based on the 80286 and the optional extension 80287 as a math coprocessor. This card plugs in to an I/O channel slot and provides compatibility with IBM PC and PC/AT programs. On this card there is a considerable amount of control logic that protects the system against improperly written PC code, supports the sharing of system resources between the coprocessor and ROMP and is used by ROMP software to emulate current PC/AT adapters while using new adapters.

PC application programs for the coprocessor are stored in either system memory, or in dedicated I/O channel attached memory. Typically the coprocessor performance is that of a PC when executing programs in system memory, and about 80% that of a PC/AT when using I/O channel attached memory.

Concerning the hard disk possibilities of the system, the 5-1/4 inch industry standard disk units are available with 40 and 70 Megabytes with the possibility of having uo to three hard disks per system. The adapters are the existing IBM PC/AT Fixed-Disk and Diskette Drive adapters, something that shows how flexible the system is.

An external streaming tape drive and separate adapter card that attaches to the I/O channel is also available as an option. Using a standard 1/4 inch tape cartridge the streaming tape unit provides a backup capacity of 55 Megabytes.

Since this system is supposed to work at a multi-user multi-tasking environment different displays are supported offering the user the choice of the display that best suits his or her needs.

In the first place the system provides for the attachment of several existing IBM PC adapters and displays, such as the Monochrome Display and the Enhanced Color Display with their respective adapters.

The new displays and adapters for the IBM RT PC provide direct processor access to a 1024 x 512 bit map with a display viewing area of 720 x 512 pixels. Special hardware assist provides for text and graphics alignment down to the pixel level.

At a functionally higher level another display subsystem provides a considerably larger viewing area of 1024 x 768 pixels as well as extensive hardware assist for very high speed vector-to-raster conversion from a vector list buffer.

A full 1024 x 1024 color display with existing advanced computer-aided design applications is provided by the unique ability to attach an IBM 5085 Graphics Workstation to the IBM RT PC system in a host-based network.

The conclusion is that the IBM RT PC system is a very powerful system designed by IBM in order to bridge the gap between the rapidly expanding Personal Computer market and the more demanding market of 32-bit supermini level workstations with extensive virtual memory management facilities. This new generation of workstations has already become the basis of powerful computing systems that have extensive storage, display and communications requirements as they are supposed to offer THE solution to rapidly growing and evolving applications.

Specifically the IBM RT PC:

- Introduces an IBM developed high performance 32-bit RISC architecture with virtual memory.

- Combines the 32-bit features with a standard PC I/O channel.

- Provides an optional PC coprocessor for compatibility with already existing PC application programs.

- Allows future performance and feature upgrades by replacement of the processor, memory, and floating point cards as technology improves.

The usefulness of the open architecture of the IBM RT PC is a phenomenon already seen from the IBM PC family. It allows both IBM and vendors to very easily come up with different products which will enhance the functionality and the usability of the system. The architecture supports numerous possibilities of performance enhancement, such as increased memory capacity, larger hard-disk capacity, higher performance displays, different local-area networks, higher speed host attachments and different kinds of coprocessors.

Software Architecture

Besides a new hardware architecture the IBM RT PC is implementing a totally new software architecture offering the user a powerful and easily reconfigurable environment.

In order to render this environment as functionally complete as possible, a three-level scheme is used. First, the built-in functions are powerful enough to satisfy most applications. Second, for the cases where the built-in functions are not complete controlled access is provided to the hardware, and third, the operating system is open-ended enough to allow for extensions to cover things such as new types of devices.

Another issue is that most of the IBM PC applications can be executed on the IBM RT PC, and there are vendors who would like to give those applications the maximum benefit from accessing the ROMP and MMU.

The layered structure (as shown in Figure 2 on page 13) of the VRM (Virtual Resource Manager) provides a lot of flexibility allowing the user to easily meet his needs.

The VRM is a software package which provides a high level operating system environment. The VRM is not an operating system itself. Its goal is to directly control the real devices and provide a standardized interface to applications. This interface is called the VMI (Virtual Machine Interface) and gives the applications the capabilities of a virtual machine and of virtual devices.

The VRM is designed to work on hardware consisting of a RISC processor and a PC/AT compatible I/O bus, but it should be kept in mind that this does not at all limit VRM to only such an environment. An example that VRM is able to support different I/O hardware is the VRM's support of the IBM 5080 graphics hardware, which is designed to an IBM 370 architecture channel interface.

We have already seen that the idea behind the RISC concept is to minimize hardware functions providing only a limited set of primitives. Since the processor is designed with a minimum logic, the native instructions enjoy a corresponding increase in execution speed. In such an environment, functions traditionally provided by hardware such as integer multiply and divide, or instructions like character string manipulations have to be replaced by software functions. That is exactly what VRM builds on the hardware. Namely VRM:

- Provides a high-level machine interface, simplifying the development of guest operating systems and their applications.

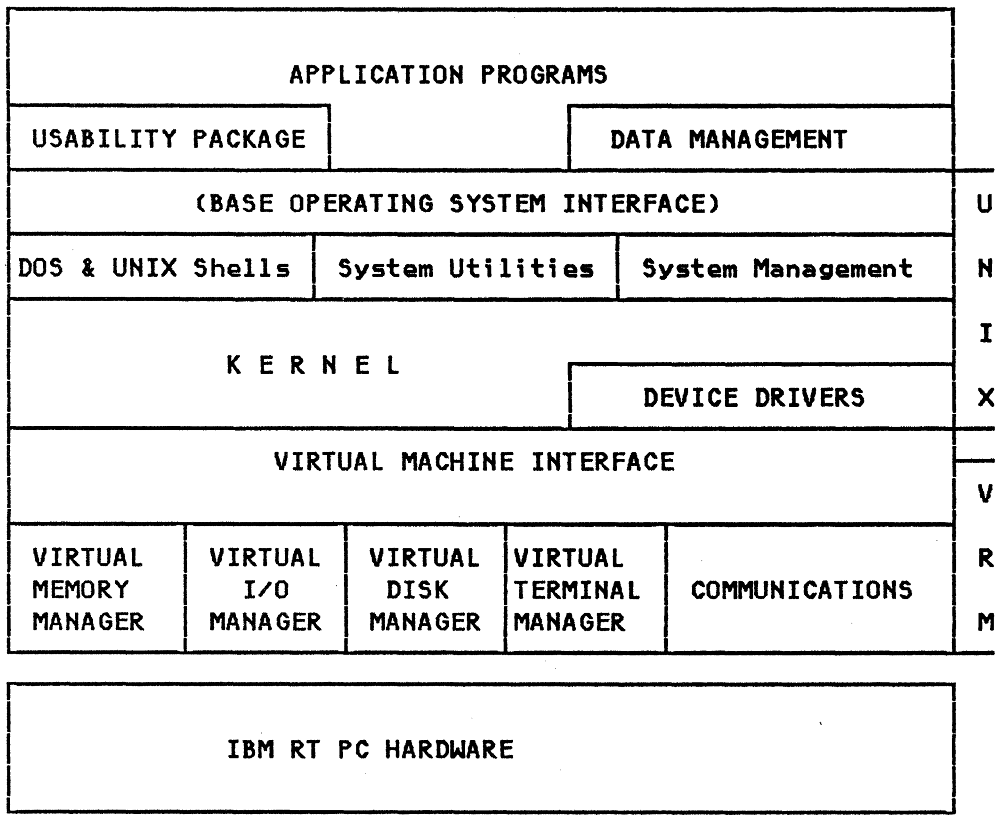

Figure 2. IBM RT PC software design

- Maximizes performance for real-time applications, although the virtual memory capabilities of the machine can be of harm to a real-time environment as we will see later on.

- Allows users to easily reconfigure and customize their system according to their needs, by providing a very flexible and extendable interface.

- Provides compatibility with IBM-PC applications thanks to the 80286 coprocessor.

The VMI implements these points with a set of functions which facilitate the use of a variety of concurrent operating systems. Except for the problem isolation instruction set, the VMI assures the isolation of simultaneously running operating systems and applications from the actual hardware.

The concept of a virtual machine is not something new on IBM machines. One of the major IBM operating systems, VM/370 implemented on IBM 370 architecture mainframes provides a virtual machine environment. There are however many important differences between the way VM implements it and the way VRM does it on the IBM RT PC. VM/370 provides a complete functional simulation of the actual S/370 hardware. This means that an operating system designed and written to run on S/370 hardware, such as MVS for example, will also run under VM in a virtual machine. The VMI support provided by the VRM provides much more functional support than the IBM RT PC hardware can. This means that if an operating system is written at the VMI level it will run as a guest under VRM but it will never run on the actual hardware.

Judging from the VM experience, we know that a virtual machine suffers in performance due to the overhead associated with the simulation of the different hardware functions. The VRM really is an exception to this rule because the vast majority of the guest operating system instructions execute directly on the hardware. The slight overhead VRM introduces is actually due to the VRM interference when it is invoked mainly to handle I/O operations at a relatively high functional level. This overhead was wanted, because the IBM RT PC designers opted for a more flexible and much more functionally rich programming environment than what a rigid native operating system could give.

The IBM RT PC operating system is the AIX, which was derived from AT & T's UNIX System V. Several enhancements to the original version have been included and the kernel has totally been rewritten by IBM. This assures a better integration of all additional and eventual functions as well as a differentiation from the UNIX-me-too vendors. For more details the reader is referred to "Appendix E. UNIX Enhancements" on page 55 and to "Appendix G. References" on page 63.

A key concept to understanding the support of multiple simultaneous interactive applications is the idea of the virtual terminal supported by AIX. A virtual terminal is the virtual counterpart of an IBM RT PC real device, such as the mouse, a display, or the keyboard. Each application works on a single virtual terminal which can either be a simulated ASCII terminal or a high-function terminal equivalent in power with the real device.

One of the responsibilities of the VRM is to take care of controlling all ROMP/MMU virtual memory functions. This relieves the operating system from having to handle page faults and management of real memory and paging space.

The operating system is also provided by the VRM with a queued interface to the I/O devices, insulating in this way the virtual machines from the burden of management of shared devices. AIX can dynamically add code to support and activate specialized devices while the VRM is running; that is without IPL.

The 80286 coprocessor is seen by the VRM as another virtual machine. When the coprocessor is active, all user input is presented to the coprocessor as if it was produced by the corresponding PC/AT devices. The VRM ensures that there will not be any interference between the 80286 coprocessor's and ROMP's distinct processing.

Two types of programs can be installed into the VRM: Device Drivers and Device Managers. Device Drivers are a set of subroutines which support a specific type of hardware device. Every time the VRM needs to cope with device-specific functions like handling interrupts and timeout conditions, and processing I/O commands from the virtual machines, it will call the corresponding Device Driver. The VRM Device Driver supports relatively simple devices, such as printers, diskette drivers, and tape drives.

On the other hand the Device Managers provide an additional level of support for more sophisticated devices, such as virtual terminals or communications subsystems. These subsystems usually have different requirements to handle multiple asynchronous events as well as to manage different kinds of shareable or non-shareable resources. The Device Managers and virtual machines are implemented by the VRM like "processes", which are served by the processor according to the control of a prioritized round-robin scheduling algorithm.

VRM support for processes includes the following features:

- Inter-process communications and queueing of messages and events are supported by queues.

- Semaphores are used for synchronization and serialization.

- Time of day and timer capabilities are equally available.

- Resources like DMA channels and interrupt levels are controlled and allocated by special functions.

A final note on the impact of virtual memory to the real-time performance of the system: Although real-time applications can in principle run on the ROMP, the current version of the operating system supports things like that rather weakly. The reason for that is that whenever there is a page fault, the interrupts from the real-time applications cannot be served before the missing pages are brought into system memory. This can be very harmful for delicate real-time applications when one cannot afford losing extra time. Although the development team is already implementing ways to bypass this shortcoming of virtual-memory systems, one should keep in mind that it is the 80286 which for the time being assumes the real-time responsibilities on the I/O channel.

Introduction

The Central Processing Unit (CPU) and the Memory Management Unit (MMU) of the IBM RT PC system are provided on a separate processor card, which plugs into a dedicated 200-pin slot on the system board. The Research Office Product Division MicroProcessor (ROMP) implements a full 32-bit RISC architecture. Current VLSI technology did not allow combining the ROMP CPU and the Memory Management Unit (MMU) on a single chip. The solution was to use two chips, one for the processor and one for the MMU. They are connected via a high performance channel, called RSC (ROMP Storage Channel). The RSC is a packet-switched 32-bit bus with a bandwidth of 23.5 Mbytes per second. This kind of performance is required to support the pipelined RISC architecture of the ROMP processor, which is able to execute an instruction almost every CPU cycle (170 ns).

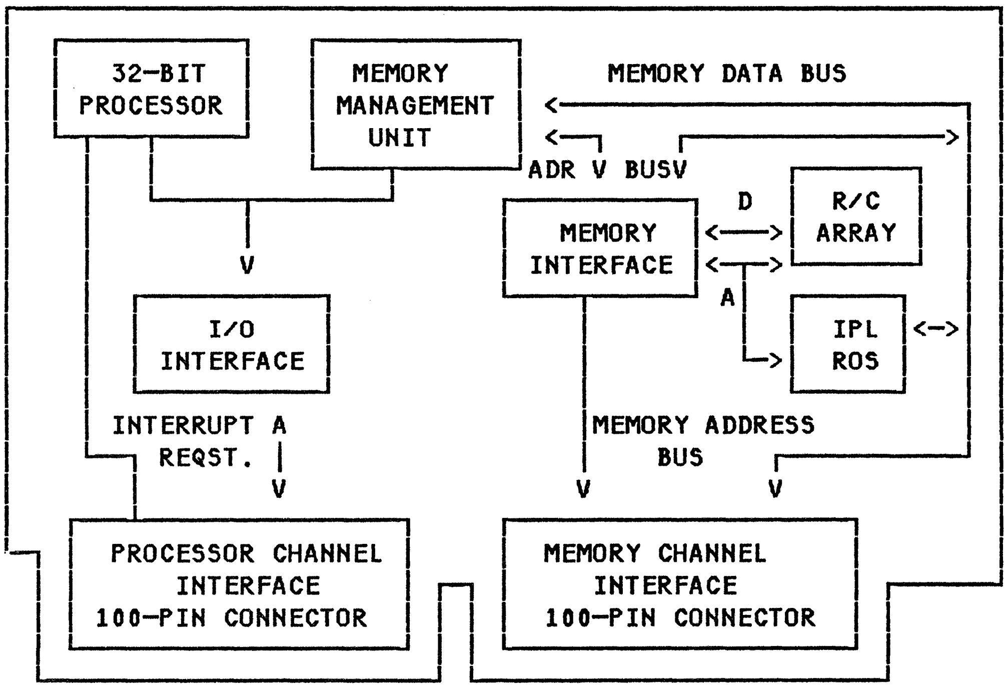

Figure 3. Processor Board Data Flow and Board Interfaces

The following list summarizes some basic features of the processor card. In depth coverage will be given in later chapters.

ROMP processor

- VLSI chip, housed in a 175-pin package
- Fetches and executes instructions
- Pipelined 32-bit RISC architecture
- Implements priority based interrupt scheme
- Supports user and system states

MMU

- VLSI chip, housed in a 175-pin package
- Supports 40-bit virtual address space (1024 GigaBytes or 1 TeraByte)
- Supports up to 16 Mbyte real memory
- High resolution memory protection scheme

RSC bus

- Packet-switched 32-bit ROMP Storage Channel
- Bandwidth: 23.5 Mbyte/sec (4 bytes every 170 ns)
- Ability to support overlapped memory access

IPL ROM

- 32 Kbyte Read Only Memory for Initial Program Load

PROCESSOR CHANNEL Connects the processor board to

- the (optional) Floating Point Accelerator, which plugs into a dedicated slot on the system board
- the I/O subsystem, located on the system board, providing access to the I/O channel. The I/O channel consists of 8-bit IBM PC and 16-bit IBM PC/AT slots for attachment of
  - standard I/O adapters
  - IBM PC AT coprocessor

- IBM PC At memory boards

MEMORY CHANNEL

- Connects the processor board to the system memory
- Two dedicated slots on the system board
- 32-bit data bus plus 8-bit Error Correction Code (ECC)
- Bandwidth : 23 .5 Mbyte/sec (4 bytes every 170 ns)

It is important to distinguish between 32-bit ROMP system memory on the memory channel and 16-bit IBM PC/AT memory on the I/O channel.

ROMP Processor

The register set of the ROMP processor consists of sixteen 32-bit General Purpose Registers (GPR's) and sixteen 32-bit System Control Registers (SCR's). All data and address manipulations are handled by any of the GPR's, which are grouped into eight pairs. These register pairs are implicitly used in certain instructions such as non-destructive shifts. The contents of a GPR can be treated as either a double word (4 bytes), a half word (2 bytes) or a character (1 byte) quantity.
The sixteen System Control registers (SCR's) are shown in Figure 5.

Some SCR's and special SCR fields are reserved. The others are assigned to system facilities such as

counter source	system timer facility to provide real time functions
counter	system timer facility
TS	Timer Status (system timer facility)
MQ	Multiplier Quotient: The MQ is an extension of the MULTIPLY STEP and DIVIDE STEP instructions
MCS	Machine Check Status
PCS	Program Check Status
IRB	Interrupt Request Buffer
ICS	Interrupt Control Status
IAR	Instruction Address Register (instruction pointer)

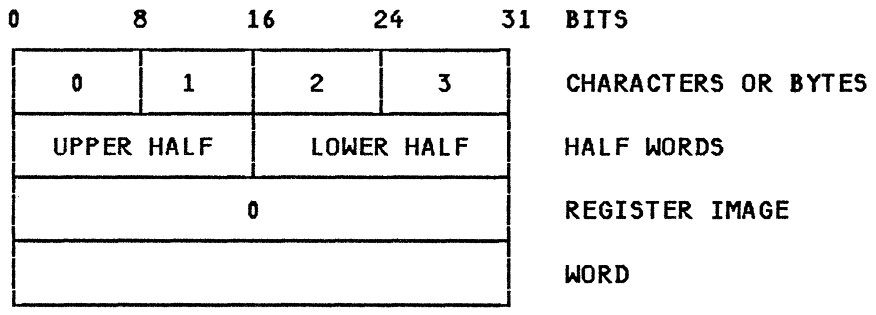

Figure 4. Data Units in System Memory

CS	Condition Status (flags)

The ability of the ROMP processor to execute an instruction almost every CPU cycle (170 ns) requires an on-chip instruction queue. The ROMP has four 32-bit Instruction Prefetch Buffers (IPB's) and they are usually at least partially filled, due to the high bandwidth of the ROMP Storage Channel (RSC), which is by far greater than necessary for program execution. The RSC has still enough bandwidth left to handle data references from the ROMP to system memory and to keep up with DMA traffic from the I/O channel to system memory.

Looking at the pipelined architecture of the ROMP it is obvious that instruction execution is relatively independent of memory requests. Instructions are prefetched into the IPB's, and having usually two or so instructions in its IPB's keeps the processor busy at almost any time. Several CPU cycles are necessary to complete one instruction. Execution of a single instruction consists of the following steps:

InFt	Instruction Fetch
InDc	Instruction Decode
ExOp	Execution of Operands in ALU
HOLD	hold-offs (wait cycles)
RgWr	Register Write to save results from ExOp step

The instruction fetch time can be assumed to be zero because of the on-chip prefetch buffers (IPB's). During every cycle the RSC receive area of the ROMP captures whatever is on the RSC bus. The tag lines of the RSC indicate if the incoming data is an instruction, to be stored in the appropriate IPB, or data to be stored into one of the registers.

At least three cycles are needed to complete a single instruction. The first cycle is used to decode the instruction (InDc), the second manipulates the operands in the ALU (ExOp) and the third stores the results of

RESERVED			SCR 0
RESERVED			SCR 1
RESERVED			SCR 2
RESERVED			SCR 3
RESERVED			SCR 4
RESERVED			SCR 5
COUNTER SOURCE			SCR 6
COUNTER			SCR 7
RESERVED	TS		SCR 8
RESERVED			SCR 9
MULTIPLIER QUOTIENT			SCR 10
RESERVED	MCS	PCS	SCR 11
RESERVED	IRB		SCR 12
IAR			SCR 13
RESERVED	ICS		SCR 14
RESERVED	CS		SCR 15

Figure 5. SCR Organization

the ExOp step into the specified register(s) on the ROMP. Only a few instructions need more than one cycle for the execution of the operands in the ALU (ExOp step).

In certain cases an instruction may be idle for a couple of cycles. So called hold-offs (HOLD cycles) occur if the instruction is waiting for data from the RSC. Since only LOAD/STORE and successful BRANCH instructions reference memory (via RSC) there are no HOLD cycles during the execution of a "normal" instruction. A highly overlapped processing scheme allows the execution of an instruction (InDc+ExOp+RgSt) almost every CPU cycle:

```
                                           cycles
                    1    2    3    4    5    6    7    8    9    10
1. instruction    |InDc|ExOp|RgWr|    |    |    |    |    |    |    |
2. instruction    |    |InDc|ExOp|RgWr|    |    |    |    |    |    |
3. instruction    |    |    |InDc|ExOp|RgWr|    |    |    |    |    |
```

The one-cycle execution rate is prevented when an instruction needs more than one cycle to manipulate the operands in the ALU (very few instructions need that), because the decoding of the next instruction will be controlled by the LAST ExOp cycle of the preceding instruction:

```
                                           cycles
                    1    2    3    4    5    6    7    8    9    10
1. instruction    |InDc|ExOp|ExOp|ExOp|RgWr|    |    |    |    |    |
2. instruction    |    |    |    |InDc|ExOp|    |    |    |    |    |
3. instruction    |    |    |    |    |InDc|ExOp|RgWr|    |    |    |
```

The first instruction needs three cycles to manipulate the operands in the ALU and therefore full overlapping cannot be achieved. The DIVIDE STEP instruction of the ROMP is an example for that.

References to memory require a two cycle hold-off. This time is required

- to get the data request out on the RSC,
- give the MMU time to access storage
- and get the reply from the RSC into the register.

Later, it will be shown that these hold-offs may be overlapped by the execution of subsequent instructions that do not reference the unloaded register(s). The following example examines a LOAD instruction:

```
LOAD      R1,VarA       #load variable from memory into register 1
```

```
                                      cycles
                  1          2          3          4          5
processor    | InDc      | ExOp      | HOLD      | HOLD      | RgWr      |

RSC          |          |          |requ|     |     |repl|          |
             |          |          |data|     |     |data|          |

Storage                                 |storage    |
                                        |access     |
```

At least five cycles are needed to get the job done. Memory will be referenced during the two hold-offs. ROMP puts the address of the variable (VarA) on the RSC at the beginning of the first hold-off cycle (data request) and the corresponding data reply from the MMU will be on the RSC at the end of the following cycle. Additional hold-offs may occur. The 2-cycle memory access includes address translation, address and data buffering and ECC error detection but not error correction. If an error is detected the reply from the MMU to ROMP is cancelled and retransmitted on a subsequent cycle if the error is correctable.

ROMP needs about 60% to 70% of the RSC bandwidth to support its inherent performance. The remainder is available for DMA I/O traffic without causing processor performance degradation. The ROMP can handle four outstanding instruction fetches and two outstanding data requests before requiring a reply. This decoupled nature of ROMP and memory is implemented by using tag lines on the RSC allowing ROMP and MMU to handle data and instructions on the RSC correctly.

As mentioned earlier, it is possible to keep the ROMP busy during the hold-off cycles of an instruction referencing memory. Due to the fact that the instruction queue on the ROMP (IPB's) is at least partially filled, subsequent instructions can be executed if they do not reference the unloaded register of the preceding instruction. Consider the following piece of code:

```
LOAD    R1,VarA       # load variable into register 1
INC     R2            # increment register 2
ADD     R8,R12        # add values of register 8 and 12
                        and store result in register 8
```

```
                                                cycles
                       1    2    3    4    5    6    7    8    9    10
1. LOAD R1,VarA     |InDc|ExOp|HOLD|HOLD|RgWr|    |    |    |    |    |
2. INC  R2          |    |InDc|ExOp|RgWr|    |    |    |    |    |    |
3. ADD  R8,R12      |    |    |InDc|ExOp|RgWr|    |    |    |    |    |
```

The example shows the high overlapping of instruction execution during hold-off cycles. The point is that useful work can be done during memory access if the compiler (or programmer) is able to place the loading of registers and their actual use in subsequent instructions not too close to each other in the instruction stream. In other words, there is a substantial difference in execution time. Consider the following example:

```
program 1:              program 2:

LOAD    R1,VarA         LOAD    R1,VarA
ADD     R1,+5           INC     R2
INC     R2              ADD     R8,R12
ADD     R8,R12          INC     R7
INC     R7              ADD     R1,+5
```

Without altering the program logic proper arrangement of the instructions results in shorter total execution time (7 cycles instead of ten):

program 1:

```
                                            cycles
                      1    2    3    4    5    6    7    8    9    10
1. LOAD R1,VarA     |InDc|ExOp|HOLD|HOLD|RgWr|    |    |    |    |    |
2. ADD  R1,+5       |    |InDc|HOLD|HOLD|HOLD|ExOp|RgWr|    |    |    |
3. INC  R2          |    |    |    |    |    |InDc|ExOp|RgWr|    |    |
4. ADD  R8,R12      |    |    |    |    |    |    |InDc|ExOp|RgWr|    |
5. INC  R7          |    |    |    |    |    |    |    |InDc|ExOp|RgWr|
```

program 2:

```
                                            cycles
                      1    2    3    4    5    6    7    8    9    10
1. LOAD R1,VarA     |InDc|ExOp|HOLD|HOLD|RgWr|    |    |    |    |    |
2. INC  R2          |    |InDc|ExOp|RgWr|    |    |    |    |    |    |
3. ADD  R8,R12      |    |    |InDc|ExOp|RgWr|    |    |    |    |    |
4. INC  R7          |    |    |    |InDc|ExOp|RgWr|    |    |    |    |
5. ADD  R1,+5       |    |    |    |    |InDc|ExOp|RgWr|    |    |    |
```

In the first case (program 1) the execution of the second instruction has to be suspended until register 1 is loaded (cycle 5) and that causes a delay of three cycles. Full overlapping is possible in case two (program 2) because of the fact that subsequent instructions do not reference register 1 before the completion of the LOAD instruction. Both programs do exactly the same but the performance gain is a significant 30 percent with program 2 (7 cycles instead of ten).

During hold-off cycles subsequent instructions can be executed only if they are already in the Instruction Prefetch Buffers (IPB's) of the ROMP. Execution has to be suspended, due to hold-off cycles, if unconditional or successful branches are taken. The BRANCH instructions alter the program flow and instructions, physically stored next in memory and prefetched by ROMP, are of no use in this case. The prefetch buffers have to be cleared and an instruction fetch, controlled by the BRANCH instruction, has to be issued to memory. A 2-cycle memory access, similar to the LOAD, and an additional cycle for reading the Instruction Prefetch Buffer (IPB) is needed before execution can resume. The hold-offs cannot be overlapped by subsequent instructions, because of an empty instruction prefetch queue at this point of time. The Instruction Prefetch Buffers have still to be refilled. Consider the following piece of code:

```
         DIVS      R1,R7        # divide step operation
         BRANCH    lab          # branch to label
  XXX:   ADD       R7,R9        # add register
         INC       R3           # increment register 3
         ...
         ...
  LAB:   SUB       R6,+8        # subtract register
         ADD       R1,R2        # add register
         ...
```

When the BRANCH is executed subsequent instructions (ADD, INC, ...) are already in the prefetch queue (IPB's) on the ROMP. They have to be discarded because the SUB instruction executes next, after being fetched from memory. Three hold-offs without any overlapping occur before execution can resume:

```
                                                cycles
                       1    2    3    4    5    6    7    8    9    10
1. DIVS    R1,R7     |InDc|ExOp|ExOp|ExOp|RgWr|    |    |    |    |    |
2. BRANCH  lab       |    |    |    |InDc|ExOp|HOLD|HOLD|HOLD|    |    |
3. SUB     R6,+8     |    |    |    |    |    |    |    |    |InDc|ExOp|
4. ADD     R1,R2     |    |    |    |    |    |    |    |    |    |InDc|
5. ...               |    |    |    |    |    |    |    |    |    |    |
```

The DIVIDE STEP operation takes three (ExOp) cycles to execute in the ALU. Full overlapping is not possible because only the last ExOp cycle controls the decoding of the next instruction (BRANCH). Simultaneously, the hardwired instruction prefetch mechanism fills the Instruction Prefetch Buffers (IPB's) by issuing memory requests whenever the RSC is available. This results in having the subsequent instructions (ADD, INC, ..) ready in the prefetch queue on the ROMP before executing the BRANCH. However, they are of no use in this case. The BRANCH alters the program flow by jumping far ahead in the instruction stream. The prefetch queue has to be flushed and reloaded, causing a 3-cycle hold-off. This time is needed to get the next instruction (SUB) from memory and have it ready to execute. The first four instructions execute in twelve cycles (only ten are shown).

The ROMP provides special BRANCH instructions to overcome this 3-cycle HOLD gap. Jumps are redefined so that they do not take place until the next instruction has completed. This is called a DELAYED BRANCH or BRANCH WITH EXECUTE. The simple idea is to overlap the 3-cycle HOLD-gap by using the modified BRANCH and rearranging the code:

```
         ...
         BRANCHX  lab        # branch with EXECUTE
         DIVS     R1,R7      # divide step
  XXX:   ADD      R7,R9      # add register
         INC      R3         # increment register 3
         ...
         ...
  LAB:   SUB      R6,+8      # subtract register
         ADD      R1,R2      # add register
         ...
```

The first two instructions (BRANCHX, DIVS) are highly overlapped. The DIVIDE STEP, which is called the subject instruction, executes during the 3-cycle HOLD of the BRANCHX (branch with execute) instruction. This results in saving three cycles. Note that both programs do exactly the same, because the BRANCH actually takes place after the completion of the DIVIDE STEP instruction:

```
                                          cycles
                      1    2    3    4    5    6    7    8    9    10
1. BRANCHX lab      |InDc|ExOp|HOLD|HOLD|HOLD|    |    |    |    |    |
2. DIVS    R1,R7    |    |InDc|ExOp|ExOp|ExOp|RgWr|    |    |    |    |
3. SUB     R6,+8    |    |    |    |    |    |InDc|ExOp|RgWr|    |    |
4. ADD     R1,R2    |    |    |    |    |    |    |InDc|ExOp|RgWr|    |
5. ...              |    |    |    |    |    |    |    |    |    |    |
```

The ROMP provides for all kind of BRANCH instructions counterparts supporting the BRANCH WITH EXECUTE. The example shows that a significant performance gain can be achieved by using this feature (execution time: 9 instead of 12 cycles). The job of the ROMP assembler programmer gets more complex. Inside knowledge of how the processor works is necessary to code efficiently. Overlapping during LOAD and BRANCH instructions has to be used to maximize ROMP performance. In case of higher level languages (C, FORTRAN, PASCAL, etc.) the compiler has to take care of rearranging the code in a way that overlapping is possible.

It should be clear by now, that overlapping of instruction execution is the key to maximize ROMP performance.

<u>ROMP Instruction Set</u>

The instruction set is clearly targeted at optimizing compilers. Seven instruction formats are supported and they are defined to have the op-code and two register fields always in the same bit positions within each instruction format to minimize instruction decode time. On the other hand, using two and four byte instructions adds unnecessary complexity to the instruction fetch and decode mechanism. The decision to support two byte instructions was based on the advantages of minimizing memory code space and memory bandwidth required for instruction fetches. The decrease in memory code space results in fewer page faults and improved system performance. Code space efficiency and compiler studies led to the definition of special short form (2-byte) versions of several instructions:

- add immediate
- subtract immediate
- compare immediate
- load immediate
- short jump (plus/minus 256 bytes)

ROMP is generally a two-address, register-to-register oriented architecture with only LOAD and STORE instructions accessing memory. No memory-to-memory instructions are provided. Usually the two register fields in the instruction formats specify the source registers with the first one also specifying the destination register, where the result of the operation is to be stored. Extensive studies showed the need for a three register instruction, called Compute Address Short (CAS), where the contents of two registers are added and the sum is placed in a third register so that the contents of both source registers could be preserved. This supports efficient address computation.

The ROMP instruction set includes only simple instructions which can be used efficiently by the compiler. ROMP does not include

- complex instructions,
- complex addressing modes,
- complex loop features,
- repeat, edit features,
- advanced floating point operations.

The goal was to implement an instruction set where almost all instructions can be executed in one (ExOp) cycle. ROMP instructions are grouped into ten classes:

class	Number of instructions
1. memory access	17
2. address computation	8
3. branch and jump	16
4. traps	3
5. moves and inserts	13
6. arithmetic	21
7. logical	16
8. shift	15
9. system control	7
10. input and output	2
	118

The memory access instructions are the only ones to reference memory. All storage addresses are computed as 32-bit quantities (base address plus displacement).

The branch and jump class include the previously discussed delayed branches (BRANCH WITH EXECUTE). Subroutine linkage is provided by the branch and link instructions. Decision making and loop control is supported by the conditional branch and conditional jump instructions.

The move and insert class instructions are concerned with the movement of data between GPR's and between a GPR and the Test Bit of the Condition Status (one of the System Control Register).

The arithmetic operations treat the GPR's as 32-bit quantities in two's complement representation. All of them effect certain flags (bits in the Condition Status register). Supported are standard functions like

- add (2- and 4-byte format)
- compare (2- and 4-byte format)
- subtract (2- and 4-byte format)

ROMP has no complex instructions like multiplication or division of two 32-bit quantities. Those operations have to be emulated in software. ROMP gives basic support in providing the MULTIPLY STEP and DIVIDE STEP instructions. There are no explicit floating point operations. A software floating point emulator provides the floating point arithmetic. An optional Floating Point Accelerator (FPA) board can be added to the system to improve performance. It plugs into a dedicated slot on the system board and is connected to the ROMP via the processor channel. The FPA is based on a 10 MHZ National Semiconductor NS32081 Floating Point Unit (FPU) and supports the IEEE 754 Floating Point Standard. Performance depends heavily on how much overlapped processing between ROMP and FPA can be achieved. However, the FPA option is currently the only way to boost floating point performance of a ROMP based system to over 200 KWips (Kilo Whetstone instructions per second).

Instructions in the system control class are generally privileged instructions that are valid only in supervisor state. They include the manipulation of System Control Register (SCR's) like

set and clear SCR bits

load program status

wait for interrupt

The input/output instructions are used to access the I/O ports of the system, e.g. manipulating the control registers in the MMU or other system elements.

Interrupt Facility

Romp implements a priority-base interrupt scheme. The interrupt sources are

- 7 external lines
- software interrupts

- error conditions

The FPA board is one of the external interrupt sources. The 16 interrupt lines of the I/O channel are controlled by two Programmable Interrupt Controllers (PIC INTEL 8259). Each of the PIC's handles 8 interrupt lines on the I/O channel and is connected to one of the external interrupt lines of the ROMP.

Software interrupts are issued by setting dedicated bits in the Interrupt Request Buffer (IRB) control register on the ROMP.

Machine-check errors (parity, time-outs) and program-check errors (addressing errors, page faults) are handled by two additional error reporting interrupt levels.

ROMP provides a special Load Program Status (LPS) instruction for software to return from an interrupt. This instruction will restore the following System Control Registers (SCR):

IAR	the instruction pointer (Instruction Address Register)
CS	the flags (Condition Status)
ICS	Interrupt Control Status

Saving of the current processor status and loading the new processor status is performed automatically by ROMP hardware. This task switching does not include the saving of any GPR's. Software is responsible for saving any GPR's modified by the interrupt service routine.

A selectable priority level, ranging from 0 to 7, is assigned to the instruction execution on the ROMP. Special bits in the Interrupt Control Status (ICS) register control this instruction priority level. Only interrupts with a higher priority level will be served. The interrupt priority levels range from 0 to 6 and are implicitly specified by using the external lines. Explicit setting is done by using dedicated bits in the Interrupt Request Buffer (IRB).

Interrupts can only occur on instruction boundaries. An exception is the BRANCH WITH EXECUTE instruction. No interrupt is possible during the jump and the execution of the subject instruction overlapping the BRANCH. In order to support virtual memory, precise interrupts were defined for ROMP so that the cause of a page fault can be identified easily. All instructions are restartable. Instructions causing page faults can be re-executed after the fault has been resolved.

Memory Management Unit (MMU)

The MMU is a separate chip on the processor board and communicates with the ROMP via the RSC. It provides virtual address translation and controls up to 16 Mbyte of RAM. The MMU can be functionally divided into three sections:

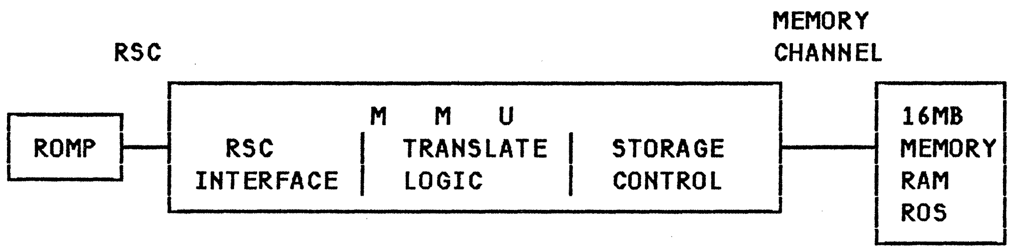

Figure 6. MMU Functional Parts

RSC Interface	handles all the communications to and from the RSC
Translate Logic	provides the translation from a 32-bit effective address, received from ROMP via the RSC, to a real address to access storage
Storage Control	provides storage access, memory refresh and ECC logic

ROMP's ability to execute an instruction almost every CPU cycle requires overlapping operations in the MMU. In fact, the MMU can handle, simultaneously, two memory requests while the result of a third one is being transmitted to ROMP. As mentioned earlier, tag lines are used to manage requests and replies on the RSC.

Virtual Address Translation

The MMU provides several control registers. In supervisor state they are accessed by ROMP via I/O read and I/O write instructions. Sixteen Segment Registers are used to build the 40-bit virtual address:

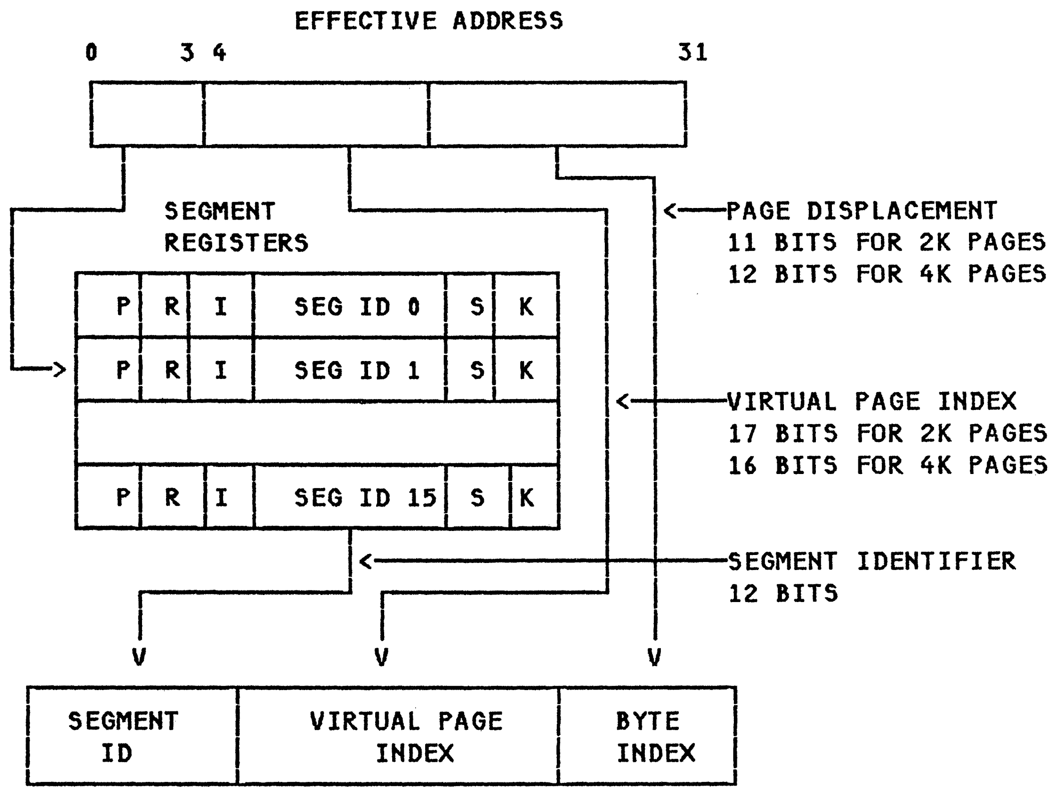

Figure 7. Generation of Virtual Address

Addresses generated by ROMP are 32-bit and called effective addresses. The high-order four bits are used to select one of the sixteen Segment Registers (SR's). The 12-bit Segment Identification (SID) of the selected SR concatenated with the remaining 28-bits of the effective address form the 40-bit virtual address. To the executing program, memory appears to be 4 Gigabytes broken into 16 segments of 256 Mbyte (28-bit) each. The 12-bit SID-entry in a Segment Register allows specification of 4096 different SID values. Therefore 256 sets of 4 Gbyte memory each can be specified, resulting in 1024 Gbytes or 1 Terabyte of virtual memory.

The translation of the 40-bit virtual address to real is done by using

an Inverted Page Table (IPT),

a Hash Anchor Table (HAT),

Translation Lookaside Buffer (TLB).

The IPT and the HAT are combined into one memory resident structure providing an entry for each page of real memory. For a given virtual page the IPT is searched for the corresponding real memory page. The HAT and hashing techniques are used to speed up the IPT search. But it still adds too many cycles to the translation time, because the search for the real memory page in the IPT involves memory accesses. In order to eliminate most of the IPT searches the MMU maintains a cache of recently-used pages in an on-chip Translation Look-aside Buffer (TLB). The TLB has 32 entries and the MMU completes the address translation in one cycle if the required entry is in the TLB. A TLB "miss" initiates the IPT search and adds eight to eleven cycles to the translation time. An IPT "miss" is a page fault and requires further action.

Memory Protection

Special bits in the Segment Register (SR) and bit fields in the IPT entry of a real page control the memory access rights. Memory protection on a page level is implemented by using the SR key bit and a 2-bit Page Key field in the IPT.

If the SR Special Segment bit is set, then a high resolution memory protection scheme applies. Each page is considered to be made up of 16 "lines" of

- 128 bytes (2 Kbyte pages) or
- 256 bytes (4 Kbyte pages).

An IPT entry (one page) contains 16 lock bits plus an 8-bit Transaction Identifier (TID) to control the page "line" access.

ECC and Parity Checking

The MMU supports either 32/40 Error Correction Code (ECC) or byte-parity for RAM. Parity checking provides data integrity by detecting all single bit failures, but does not allow corrections. ECC requires additional check bits, but allows detection and correction of all single bit errors. In addition it will detect all double and most multiple bit failures. The support for ECC on the RAM is justified by the large memory sizes expected to be used with the ROMP/MMU processor complex (up to 16 Mbyte RAM). The impact of ECC on the access time is about 30 nanoseconds for the normal case when no error is detected.

Introduction

The VRM is a collection of programs which control and extend the hardware of the Reduced Instruction Set Computer (RISC). VRM consists of a variety of processes, device drivers and runtime routines which provide virtual devices in order to enable the operating system to support multiple simultaneous interactive applications.

The main purpose of VRM is to support a virtual machine interface (VMI) for the operating system. This defined interface makes it easy to change the underlying hardware without having to adapt the operating system. It's relieving the operating system of the responsibility for page fault handling, paging space management and memory management. Providing a queued interface to the I/O devices, it frees the virtual machines from the shared device management. Code for new devices can be added to the VRM dynamically so that new devices can be added without IPL of the system.

The advantage of virtual machines for the user is that he has no restrictions on program memory (address space for virtual memory is 1000 Gigabytes). For the programmer the VRM means that he does not have to program down to the processor instruction level.

Highlights of VRM

The VRM programs, extending and controlling the hardware, mainly support the Virtual Machine Interface (VMI) which is an interface between the operating system and the VRM plus hardware (see "VMI characteristics" on page 44). VMI contains features allowing operating systems to run concurrently, while insulating them from most details of hardware implementation. Additionally these VMI features allow the installation of extensions to the VRM in order to support different I/O hardware. The basic idea of generating a virtual machine, realized already in the software product VM/370, is significantly different in its implementation in the IBM RT PC. The Virtual Machine Interface supported by the VRM offers more capabilities than the hardware can provide. Consequently an operating system implemented to the VMI will not run on the real hardware. The main characteristics VRM provides along with the VM concept are:

virtual memory

mapped files

virtual devices as minidisks and virtual terminals.

Virtual Memory:

The hardware memory management handles 16 Megabytes of real memory (24-bit address space) and allows a 40- bit space for virtual memory, which allows one to address 1 Terabyte split into 4096 segments of 256 Megabytes each. 16 segment registers are provided by the hardware; one is reserved for addressing I/O devices. Therefore 15 segments can be accessed simultaneously.

Mapped files:

They are related to virtual memory. Logical disk blocks are related to virtual addresses, so that read/write of files can be done with the associated memory addresses and avoid explicit disk read/write (see minidisk manager) which adds to increase performance.

Minidisks:

Minidisks are designed as physically adjacent partitions of a real disk which have their own virtual device addresses. A separate minidisk is allocated for paging space to back up virtual memory on the physical disk. Additionally a VRM minidisk exists containing all its runtime routines and device drivers. The minidisks are partitioned into logical blocks the size of which are determined by the operating system. The minidisk manager not only takes care of functions normally found in simple hardware access methods but it also handles error recovery and bad blocks.

I/O devices:

The virtual resource concept is also applied to I/O devices. VRM provides virtual devices such as virtual terminals whose actions are coordinated by the virtual terminal manager. The latter

handles virtual configurations

opens and closes virtual devices

does routing

coordinates the use of the display screens

makes the virtual device drivers available.

That means VMI includes a high level interface to I/O devices which is consistent for all devices.

I/O support

Device driver

Dynamically installable devices

With the **device driver** concept VRM generates a hardware independent device interface for a physical hardware device. The driver accepts commands from higher level (VRM device manager or AIX device driver) and generates the I/O commands for the real device in order to perform the enqueued request. The device driver is also responsible for handling interrupts from the hardware (see Figure 1 on page 9).

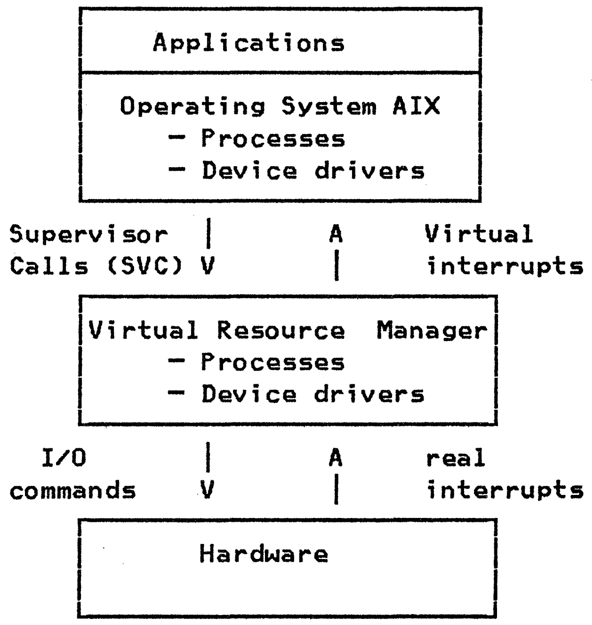

Figure 8. Device Driver: Functional description of device driver concept

The format of the device driver code is mainly the same for all drivers, but it differs in the characteristics of the devices they control. These characteristics are noted in a table-like field, called Define Device Structure (DDS). DDS is sent from the virtual device to the VRM in order to communicate information. It therefore allows the user to define other than IBM-supplied predefined devices. Figure 9 on page 36 shows the basic structure of the DDS. Only the header has a constant length. The other DDS fields vary in length for each device driver and the error log field is not even used by some drivers, so it can be empty. Error logging is supported by the VRM for devices, like disks, where the error log data may occupy several sections because that device driver has to perform error recovery.

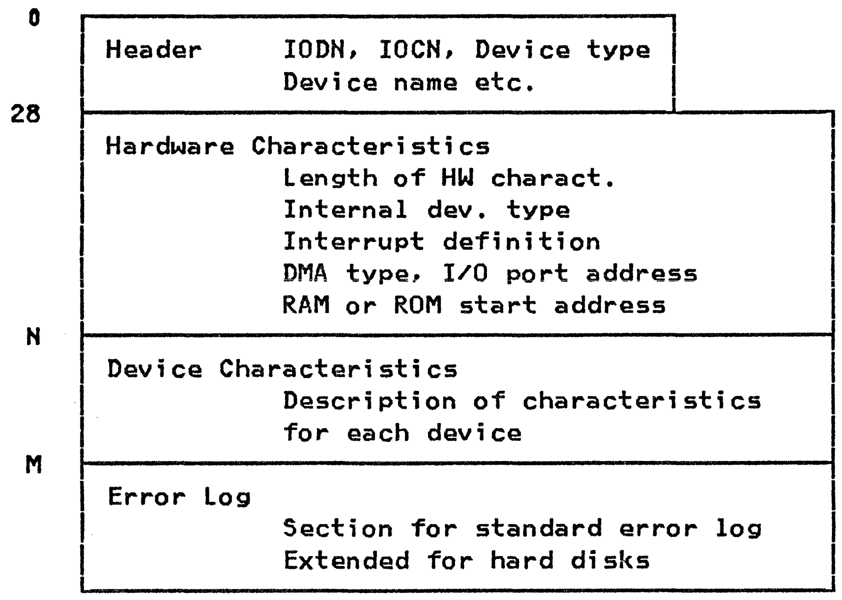

Figure 9. Define Device Structure (DDS): Field of DDS, with DDS header,section for hardware characteristics, section for device characteristics and section for error logging

The information in the DDS field for hardware characteristics includes:

the I/O port address used by the device

which DMA channel is used

which bus interrupt level it uses

whether it has resident RAM or ROM

Moreover the DDS-field contains information about which program module should be called to process functions like:

Device definition (default)

Device initialization

Device termination

I/O initiation

Interrupt handling

Exception and timeout handling.

Using information from the DDS, VRM is able to determine which user installed program to call for handling an interrupt generated by a real device. The VMI has functions to use real devices and to install a logical connection, or path, between the user and the device .

There are two type of modules available in VRM for handling devices:

Device driver (modules)

Device manager (modules)

The **device driver** modules contain subroutines which support specific hardware devices. VRM performs most of the common device driver functions (generating interrupts, allocating space for code and data areas); for device specific functions, however, VRM calls these subroutines (eg. interrupt handling, I/O initiation from the virtual machine, timeout handling etc.). For most device subsystems the VRM device driver support is sufficient (printers, disk and diskette drivers, asynchronous device driver, etc.).

More sophisticated subsystems (resources) which involve virtual devices, such as minidisks, virtual terminals and communication subsystems, however, require **device managers**. The device manager modules handle multiple asynchronous events which are typical for a virtual terminal with its display, keyboard, speaker, etc.

To **summarize** the conceptual ideas of the VRM's device

1. Each physical device is controlled by a device driver.

2. A device driver consists of subroutines that are called by the VRM to handle the following operations for a device:

    definition

    initialization

    termination

    I/O initiation

    interrupt handling

    exception handling

3. Each device has an I/O device number (IODN); VMI has facilities to put the IODN in the device driver or in the process (Define Device Supervisor Call).

4. Each device has a Query Device Structure (QDS) to describe the device and its status. The device driver (or process) is responsible for updating it as well as the error log fields in DDS.

5. The VRM manages the physical resources like:

   adapters

   interrupt levels

   memory

   DMA channels

6. VRM uses the following types of registers to manage its resources:

   segment registers to control virtual memory access

   system control registers to manage virtual machines.

The **dynamically installable devices** make the architecture of the virtual machine extendable. In the above mentioned DDS which is included in the VMI a programmer can describe the attributes of a new device and its related software support to the VRM.

Real time tasking structure (for I/O processes)

Memory management

Timer management

Process management (interrupt/dispatching)

A considerable high accent was given to the real-time performance of the system. Particular emphasis is placed on supporting high speed devices. For example, the VRM device driver is able to handle a disk formatted with a 2:1 interleave factor (see "Appendix D. Other I/O bus options" on page 53). An "off-level" interval handler capability is available which allows a device interrupt handler to process time critical operations without their task being preempted. It transfers less critical processing to a lower priority level which can be preempted by other device interrupts.

For the VRM a process is a distinct entity, which receives time from the processor in order to handle different programs. By its nature, the VRM supports different active processes at the same time. Multiple processes are dispatched according to their priority levels. In the case of equal priorities, a round-robin algorithm is followed.

A process can be in one of the four following states:

- Ready
- Running
- Waiting

- Terminated

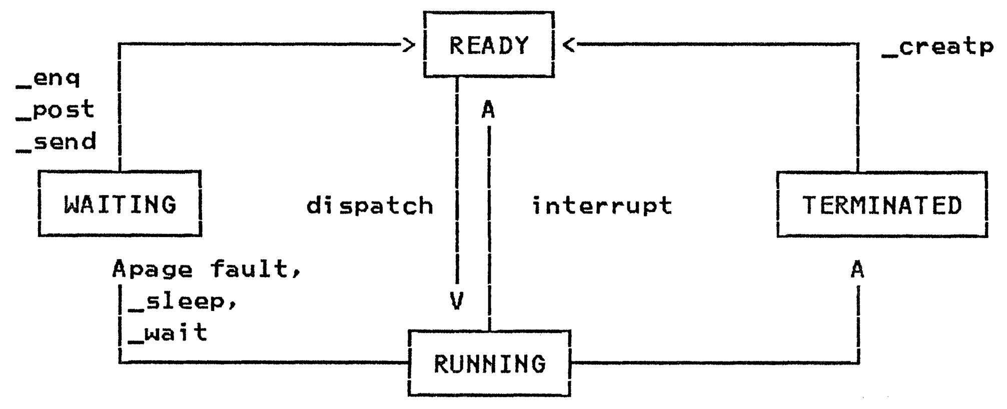

Figure 10. Process management: It describes how a process is managed by VRM

Figure 10 shows the cycle every process has to go through when supervised by the VRM.

When a process is created, the VRM initializes it and places it in the Ready status. The process is then dispatched by the VRM and placed in the Running status. Of course a running process may be interrupted at any moment. In such a case the VRM will put it back in the Ready status. A running process may also be placed in a Waiting status.

There are actually three ways to force a process into the waiting status:

- Queue event (e.g. arrival of an expected element)
- Semaphore (e.g. synchronization between processes)
- Timer
- Page fault (e.g. the VRM has first to resolve it)

For more details about Process Creation and Termination, Exception Handling, etc., the reader is referred to the VMI Technical Reference Manual.

The VRM process management is characterized by a very efficient process switching mechanism. In addition, the queueing and semaphore techniques provide an efficient means of interprocess communications (IPC) by offering the ability to pass messages and synchronize processes, as well as lock shared resources without the need of complex procedures.

In a typical operating system, when an interrupt occurs the state of the interrupted program is saved in a known location, then transferred to a control block associated with the interrupted program if it is necessary to switch control of the processor to a different program. In the VRM, this would require moving around a large amount of data. Therefore, the interrupt handler is set up to save the state of an interrupted program directly into its control block. This contributes to faster context switching.

Logical disk support and virtual storage control

Minidisk

virtual memory

mapped files

VRM supports either serially usable or shared resources. Serially usable are devices like printers on which one application has to be finished before the next one begins. Disks and memory are shared resources like real terminals. They are shareable by splitting them into logical units.

Minidisks: The virtual machine allows a physical disk to be divided into minidisks. VRM takes care of the translation of an element's physical address (sector number) into its logical minidisk block number (and vice versa) using the minidisk manager. It contains commands or I/O routines for handling minidisks as well as fixed disks. It also communicates with the fixed disk device drivers. Additionally the minidisk manager takes care of error recovery and bad block relocation, functions usually not found in hardware access methods.

Virtual memory: VRM provides virtual machines with paged virtual memory. Because the paging mechanism is hidden, virtual machines treat virtual like physical memory with variable access times. Virtual memory is divided into segmunts, that is, linearly addressable spaces of one or more 2K-bytes pages with a maximum size of 256 MB. VRM maintains 16 segment registers and such gives access to 15 segments of virtual memory simultaneously.

Direct Memory Access (DMA) is used for I/O adapters to access either the system processor or the coprocessor. Eight channels may run in shared or in non-shared mode.

Mapped files: Logical disk blocks are related to virtual addresses enabling read and write of files to be done with associated memory addresses access. This avoids explicit disk I/O.

Coprocessor support

The VRM also manages the resources which can be shared with the 80286 coprocessor (DMA channels, disks etc.). For **non shared devices** VRM reserves the device for being used exclusively by the co-processor. Such I/O processes (e.g. printing) take place without further control of the VRM and they perform at the same speed as in an IBM PC/AT.

For **shared devices** (displays, diks etc.) the VRM has to simulate a real dedicated device to the coprocessor. The VRM is able to write data to memory buffer until the requested device is "free". This is the reason why performance is degraded in shared devices to some extent compared to native IBM PC-AT application.

Specifically for memory, another shared resource, VRM can reserve own memory for the coprocessor's use. In order to gain performance a memory card can be plugged into the I/O bus additional to the system memory.

Virtual console support

Virtual Terminal Manager (VTM)

Virtual Terminal Resource Manager (VTRM)

Device drivers

VRM supports multiple virtual terminals. These virtual terminals are handled by a Virtual Terminal Manager (VTM), a component of the VMR. VMI serves as interface between the virtual machine and the VTM. The VTM is a collection of VRM components which:

extends the function of I/O hardware

controls the physical terminals

controls input devices (keyboard, mouse)

controls output devices (displays, speakers)

maps virtual terminals to virtual machines.

The **Virtual Terminal Manager** consists of

1. **Virtual Terminal Resource Manager (VTRM).** It coordinates the actions of all virtual terminals with its two modules: a resource controller and a screen manager.

2. **Virtual Terminal Mode Processor (VTMP).** It controls operation modes, datastream information, etc.

3. **Device drivers.**

The preferred method of using a display from a virtual machine is to take advantage of the VRM and its I/O support functions. The device manager VTMP, however, gives access to two modes: Keyboard Send/Recieve Mode and Monitor mode. The Monitor Mode allows the VRM services to be bypassed, which might be of interest for graphic applications. Performance is gained, but with the disadvantage of less flexibility. It provides controlled access to the real hardware and compatibility with existing applications (PEEK and POKE in BASIC).

Development support

In order to exploit most of the capabilities of the VRM functions, a wide and extensive development support is provided:

VRM debugger: a full screen, real terminal (system console) debugger for error detection mainly in code of device drivers.

C- and assembler interface routines in VRM

VRM Management routines

Process management

Queue management

Memory management

Semaphore management

Timer management

Program management

Device management

Virtual machine control procedures

VMI characteristics

The Virtual Machine Interface (VMI) is a software interface between the operating system(s) and the Virtual Resource Manager (VRM). And as described in the preceding VRM chapters VRM controls and extends the hardware. VMI therefore presents a standardized hardware-like interface to the operating system(s). Changes in hardware configurations normally do not require a change in the operating system. The second important characteristic of the VMI is that it allows concurrent virtual machines (VM), based on one VRM.

Main characteristics:

Allows installation of multiple virtual machines including concurrent execution of operating systems

Virtual machines as defined by VMI have a high level hardware-like interface.

VMI isolates virtual machines from each other and from the VRM.

VMI insulates operating systems from hardware changes.

It thus gives the impression of multiple (see virtual terminals) and enhanced (see virtual memory) resources.

Components:

Problem state instruction set.

A simulated privileged machine structure and a set of privileged machine functions.

A paged virtual memory system.

Device independent interface to displays and input devices.

Functions for multiple virtual machine management, functions which provide for the VM termination, communication between machines, etc.

System integrity and virtual machine architecture.

A virtual machine is, by definition, a simulation of a physical machine and its related devices. Some limitations are imposed on the virtual machine in order to maintain system integrity:

Virtual machines execute with the processor being in an unprivileged state. It runs in the normal execution level 7 until its work is completed, interrupted or preempted.

Hardware devices are accessible only in privileged state with floating point accelerator as exception.

A virtual machine can access only its memory and memory which has no specific owner.

Only programs running in privileged state or programs directly accessing the I/O bus can affect systems integrity.

When a process executes, data manipulation and computation are done in the general purpose registers of the processor. System control registers keep track of facilities like processor, timer, and interrupts. The virtual machine control registers (VMCR) are the equivalents of the system control registers (SCR).

Interrupts are segregated into different levels (levels 0 to 7, plus program cheque and machine communication level). The levels are determined by the source or the cause of the interrupt, work which is done by the interrupt handlers. When an interrupt occurs the interrupt handlers will save the status of the machine at that time and determine the address of the interrupt handling routine for that level. The program status word will be used for the physical machine and the program status block for the virtual machine. The interrupt level points for both machines either to a program status word or block, which directs the machine to the address of the appropriate interrupt handling routine.

APPENDIX A. THE IBM PC/AT COPROCESSOR

The IBM PC/AT Coprocessor option is implemented on a single board which plugs into a unique slot on the I/O channel. The Coprocessor provides full IBM PC/AT processing power. The packaging includes major IBM PC/AT parts like

- 80286 processor
- 80287 numeric coprocessor (optional)
- 80284 clock
- 8259 interrupt controller (two)
- 8254 timer
- 12 mhz crystal clock

There is no onboard memory and two memory options are available to run the Coprocessor

System memory The use of 32-bit ROMP system memory is the most economical way to run the Coprocessor. The long access path from the Coprocessor to system memory results in a performance degradation of about 60 percent (slightly better than an IBM PC/XT). ROMP performance may also suffer due to memory interference.

I/O channel memory The Coprocessor approaches IBM PC/AT performance if 16-bit I/O channel memory is installed. In this case true concurrent processing is provided (ROMP using 32-bit system memory and the Coprocessor using 16-bit I/O channel memory).

ROMP and the Coprocessor use

- keyboard,
- disks,
- displays

on a time shared basis under the control of ROMP.

APPENDIX B. FLOATING POINT ACCELERATOR (FPA)

The FPA is implemented on a single board which plugs into a dedicated slot on the system board. ROMP communicates with the FPA via the processor channel. The FPA is based on National Semiconductor's NS32081 Floating Point Unit (FPU) running at 10 MHZ. The IEEE 754 Floating Point standard is supported but additional software is needed to fully implement the standard.

Overlapped processing between ROMP and FPA is possible. An onboard external register file (32 sets of sixteen 32-bit registers) provides increased read and write Floating Point Register (FPR) performance. Performance depends heavily on how much overlapped processing between ROMP and FPA can be achieved.

Currently the FPA option provides at least 200 KWips floating point performance.

APPENDIX C. SYSTEM MEMORY BOARDS

The architecture of the system memory cards provides full two-way high performance interleaving, allowing a data word access every 170 ns while using inexpensive, industry standard 150 ns Dynamic Random Access Memories (DRAMs). The system memory cards plug into two dedicated slots on the system board.

The memory chips are packaged on 1 MB and 2 MB cards which provide for system memory configurations of 1MB, 2MB, 3MB and 4MB. The maximum throughput rate is 27 Mbytes/sec. The hardware architecture allows up to 8MB per card for a total of 16MB of system memory, which is currently the maximal amount of real memory the MMU can manage.

Each data word consists of 32 data bits and eight Error Correction Code (ECC) bits. The memory interface bandwidth is 23.5 Mbytes/sec.

APPENDIX D. OTHER I/O BUS OPTIONS

The I/O bus system differs in the two IBM RT PC models 6150 and 6151 only in the amount of slots provided for the adapter cards. The 6150 has eight I/O slots, two PC type and six PC-AT type slots; the 6151 has only six I/O slots, one PC type and five PC-AT type slots. In both models one PC-AT slot is unique and is reserved for the coprocessor option.

All PC and PC-AT adapter cards will fit in these I/O slots, but only the following are fully tested and recommended for use:

1. Monochrome display adapter
2. Enhanced graphics display adapter
3. Advanced Monochrome graphics display adapter
4. RS232C asynchronous adapter (4 ports)
5. RS422a adapter(4 ports)
6. Serial/parallel printer adapter (PC-AT)
7. Disk and diskette drive adapter
8. Streaming tape drive adapter

The **Monochrom Display Adapter** used in IBM RT PC is in fact the IBM PC-AT monochrome and parallel printer adapter. It is fully supported by the hardware and the AIX operating system as is the **Enhanced Graphics Display Adapter**.

The **Advanced Monochrome Graphics Display Adapter** provides a 64 K-byte bit map translating 720 pels horizontally by 512 pels vertically to the display. It supports exclusively the **Advanced Monochrome Graphics Display**, a high resolution monochrome display. It has an addressable format of 720 by 512 pels and its display image is interlaced at a refresh rate of 46/92 Hz.

The **RS232C Asynchronous Adapter** provides four serial ports, is fully programmable and supports asynchronous communication only. A programmable baud-rate generator allows operation from 50 bps to 19200 bps. (Note: Two additional RS232C serial ports are provided with the model 6150 on the system board supporting DMA transmit.)

The **Disk And Diskette Drive Adapter** is the actual IBM PC-AT adapter card which is able to support two 40 or 70 MB disks and two diskette drives. For the 6150 model with three disks two adapters are required.

The **Streaming Tape Drive Adapter** supports the streaming tape drive containing a microcontroller and a data buffer of 2Kbytes. The streaming tape drive has a capacity of 55 M bytes and transfers data at a rate of 86.7K bits per second.

For further information refer to literature.

The base of the AIX operating system is the AT and T UNIX System V. However, the kernel is totally rewritten by IBM. The AIX implementation comprises selected industry enhancements from UNIX Version 2, PC-IX (the IBM Personal Computer Interactive Executive) and the Berkeley UNIX Version 4.2.

This appendix gives a tabulated summary of the capabilities of this extended UNIX. For more details, the reader is referred to the "AIX Operating System Reference Manual".

AT & T System V.1 enhancements (over Version 7 and System III)

- File system performance and integrity enhancements
- Command and library enhancements
- Symbolic debug facility.
- Interprocess communication (IPC) with queues, semaphores and shared memory.

AT & T System V.2 enhancements include

- vi Editor
- Improved utilities (ls, ar, pg)
- Enhanced curses and terminfo
- Enhanced shell

Standard PC-IX enhancements

- File system and kernel enhancements
- System management enhancements
- Terminal/port management and control
- Generalized queueing system
- Improved commands and utilities
- Async network applications (innet/inmail/ftp)
- Full screen editor (INed)

Selected Berkeley BSD 4.2 enhancements

- Enhanced signals
- Multiple concurrent group access
- File system enhancements (fsync, ftruncate)
- C-shell.

UNIX extensions for process/program management in virtual memory include the following:

Kernel demand page fault handling

- User process page faults
- Kernel preempts when page fault on user data.

Virtual fork

- Process creation requires duplication
- New program replaces
- Logical copy
- Copy on reference in new process.

Other features of the extended UNIX are

Mapped file support

- Disk files mapped into memory
- "Single level store"

User data files

- Read/write option: Memory operations automatically reflect the disk file shmat (fildes, address, shm_copy)
- Copy on write option: shadow page recovery, shmat (fildes, address, shm_copy), fsync (fildes).

Executable files

- Kernel maps text/initialized data page in place
- linkedit option

(post R1)

- Subroutine level code sharing

IPC message queue enhancements

- Queue element permission structure
- Signal on IPC message queues (post R1).

Floating point support

- IEEE emulation for boxes without FPA
- Management of floating point hardware resources with portable object code mode and direct access for full performance.

For the file system the implemented enhancements include

Virtual disk support

- Mini disk
- Generic DASD device driver.

Support of multiple block sizes

- 512B / 2KB.

Support for data management/data base

- Truncate a file to a specified length - ftrunc
- Release space within a file - fclfar
- Synchronize a file in core state with disk - fsync
- File and record level locking with user and group definition.

Removable media support

For input/output management the extensions include

Device driver enhancements

- Dynamically configured device drivers
- Base for dynamically installed drivers.

Generic device independent device drivers

- TTY device driver
- Printer device driver.

Multiplexed device drivers

- Logical "device" support
- Virtual terminals

- SNA sessions.

Terminal (console) support enhancements include

Multiple virtual terminal support

- "Hot-key" between virtual terminals

UNIX support for VRM console

- TTY
- Monitored mode
- Sound
- Mouse.

Extended curses library and terminal data base

- Upward compatible from Berkeley BSD version
- Enhanced for color, windows, extended graphics,...

The extended curses include a set of window management routines with enhanced performance and major functions, which are

- Window features (create, delete, erase, overlay, overwrite, scroll, subwindows)
- Output functions (add/delete/insert characters/strings, move current position in window)
- Display attribute functions (color, change next x characters to "mode", set up attribute desired initially, start/stop added characters have "mode")
- Windowbox functions (draw a box around the window)
- Input functions (get character/string, get key)
- Control/utility functions (query/change terminal characteristics)
- Full PC ASCII character set.

The Usability shell improvements include

"Usable UNIX" interface

- User friendly interface with UNIX subset of commands and parameters, with po-up menus and pointing device function keys
- Support of a wide range of ASCII terminals

- Features such as windows manager, tools window (to access UNIX system functions), files window (to use the UNIX directory structure), shell windows (UNIX, DOS).

The UNIX extensions include printer services, namely

- 5152/5182
- Quietwriter
- APA page printer
- Pro Printer

  Support for OEM ASCII

- Specific printer conversions

  Multiple character sets / fonts

  Data formatting.

For the IBM Personal Computer compatibility mode, the UNIX extensions include

- DOS 3.0 shell on UNIX with DOS command interface, access to UNIX files and functions and access to DOS files on diskette or the coprocessor minidisk.
- PC DOS file access API. It provides DOS 3.1 file functions for both DOS and UNIX files.
- PC DOS file conversion utilities, allowing transfer between UNIX and DOS files, as well as ASCII conversion.

For basic LAN services there is a "PC Network BIOS" functional interface to PC Network.

From the languages point of view, PC mode is supported in PASCAL and BASIC allowing the treatment of integers, PC I/O functions, and PC floating point.

The coprocessor services assure concurrent execution, shared resources and display mapping.

As an interactive workstation, the following features are supported:

- UNIX "PC Talk"
- VT100 (extended) and IBM RT PC data stream
- File transfer (xmodem protocols)
- Multiple flow control protocols (xon/xoff, prompted, etc.)

The UNIX enhancements allow for the following configuration services:

- Dynamic configuration of I/O devices, hardware features, minidisks, etc.
- Menu-driven user interface that shows current configuration, adds/deletes/changes printers, terminals, minidisks,...
- Application interface to all these features
- System configuration automatically updated, with file systems and queues created and the kernel rebuilt if necessary.
- System dynamically configured at IPL time. Hardware recognition and configuration files are used. Dynamic reconfiguration anytime something is changed.

Finally the RAS (Reliability-Availability-Serviceability) features include

- Error logging services with error device driver, error collection daemon and error log analysis routines.
- Trace services with trace device driver and trace recording daemon.
- Dump facilities
- Software RAS support with patch facility and product update.
- Version/Level information.

The IBM RT PC is a product bridging the gap between the personal computers introduced during the last few years and the emerging advanced 32-bit workstations with extensive virtual memory management facilities. These workstations slowly become the basis of computing systems that have extensive storage, display and communications requirements in order to satisfy different applications as they evolve.

This bulletin presented all the hardware and software capabilities of the IBM RT PC. Namely, this system

- Introduces an IBM developed high performance 32-bit RISC architecture with virtual memory.
- Combines the new 32-bit features with a standard IBM Personal Computer I/O channel.
- Provides an optional PC/AT coprocessor for compatibility with existing PC applications.
- Allows future performance and feature upgrades by replacement of the processor, memory, and floating point cards as technology improves.
- The layered software structure based on the Virtual Resource Manager assures the virtual machine capabilities and provides facilities for an extendable architecture.

1. IBM RT PC Virtual Resource Manager Technical Reference, Doc.No. ATX4-0130.

2. IBM RT PC Virtual Machine Interface Technical Reference, Doc.No. SC23-0811.

3. IBM RT PC AIX Operating System Technical Reference, Doc.No. SC23-0807-0.

4. IBM RT PC Hardware Technical Reference Manual, Doc.No. SC23-0854-0.

5. Radin, G. The 801 minicomputer. IBM J. Res. Dev. 27,3 (May 1983), pp.237-246.

6. Patterson, David A. Reduced Instruction Set Computers, Com. ACM, 28, 1 (January 1985), pp.8-21.

7. IBM RT PC Technology, Doc.No. SA23-1057.

Accumulator: A register in which the result of an operation is formed.

Adapter: An auxiliary device or unit used to extend the operation of another system.

Address: (1) A name, label, or number identifying a location in storage, a device in a network, or any other data source.
(2) A number that identifies the location of data in memory.

Address Bus: One or more conductors used to carry the binary-coded address from the processor throughout the rest of the system.

Addressing: (1) In data communications, the way that the sending or control station selects the station to which it is sending data.
(2) A means of identifying storage locations.

Algorithm: A finite set of well-defined rules for the solution of a problem in a finite number of steps.

All Points Addressable (APA): A mode in which all points of a displayable image can be controlled by the user.

(ASCII)

American National Standard Code for Information Exchange: The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information exchange between data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

Asynchronous Transmission: In data communications, a method of transmission in which the bits included in a character or block of characters occur during a specific time interval. However, the start of each character can occur at any time during this interval. Contrast with synchronous transmission.

Base Address: The beginning address for resolving symbolic references to locations in storage.

Base Register: A general purpose register that the programmer chooses to contain a base address.

Binary Synchronous Communications (BSC): A form of communications line control using transmission control characters to control the transfer of data over a communications line.

BIOS: Basic Input/Output System.

Block: (1) A group of records that is recorded or processed as a unit. Same as physical record.
(2) Ten sectors (2560 bytes) of disk storage.
(3) In data communications, a group of records that is recorded, processed, or sent as a unit.

Block Check Character: The character used in BSC to check that all bits transmitted were received.

Branch: In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met. On the IBM RT PC, all branches occur relative to the Instruction Address Register.

Bus: One or more conductors used for transmitting signals or power.

Channel: A path along which data passes. Also a device connecting the processor to I/O.

Communications Adapter: A hardware feature enabling a computer or device to become a part of a data communications network.

Complementary Metal Oxide Semiconductor (CMOS): A logic circuit family that uses very little power. It works with a wide range of power supply voltages.

Configuration: (1) The arrangement of a computer system or network as defined by the nature, number, and the chief characteristics of its functional units. More specifically, the term configuration may refer to a hardware configuration or a software configuration.
(2) The devices and programs that make up a system, subsystem, or network.

Cyclic Redundancy Check (CRC):
(1) A redundancy check in which the check key is generated by a cyclic algorithm.
(2) A system or error checking performed at both the sending and receiving station after a block-check character has been accumulated.

Device Driver: A program that operates a specific device, such as a printer, disk drive, or display.

Device Manager: Collection of routines that act as an intermediary between device drivers and virtual machines for complex interfaces. For example, supervisor calls from a virtual machine are examined by a device manager and are routed to the appropriate subordinate device drivers.

DMA Direct Memory Access: DMA is a method of transferring large blocks of sequential data between two devices with minimal interference to the system processor. One participant in the transfer is usually memory. The other is an adapter or option which supplies or receives the data.

DMA alternate controller: An alternate controller is an adapter or option WITH the ability to handle its own addressing and I/O channel control. Compare this with a "DMA device" where the system DMA controller is needed. Alternate controllers may also have a device mode. When in device mode, the alternate controller acts like any other I/O device.

DMA controller: A DMA system controller is either the system DMA controller (in the IBM RT PC it is the INTEL 8237 DMA controller) or an alternate controller.

DMA device: A DMA device is a device that has the ability to request a DMA operation. It requests use of the I/O channel when it needs to transfer data with another device (usually memory). Such a device uses the system DMA controller which provides addressing and data transfer control signals for the transfer. Note that the DMA device must be ready to provide or receive the data before it requests the use of the I/O channel.

Effective Address: A real storage address that is computed at execution. The effective address consists of contents of a base register, plus a displacement, plus the contents of an index register if one is present.

Exception Handler: A set of routines used to detect deadlock conditions or to process abnormal condition processing. This allows the normal execution of processes to be interrupted and resumed.

First Level Interrupt Handler (FLIH): A routine that receives control of the system as a result of a hardware interrupt. One FLIH is assigned to each of the six interrupt levels.

General-purpose Register (GPR): A register, usually explicitly addressable within a set of registers, that can be used for different purposes; for example, as an accumulator, or as an index register, or as a special handler of data.

Index: (1) A table containing the key value and location of each record in an indexed file.
(2) A computer storage position or register, whose contents identify a particular element in a set of elements.

Index Register: A register whose contents are added to the operand or absolute address that results when a displacement is added to a base address.

Initial Program Load (IPL): The process of loading the system programs and and preparing the system to run jobs.

Input/Output (I/O): (1) Pertaining to a device or to a channel that may be involved in an input process, and at a different time in an output process.
(2) Pertaining to a device whose parts can be performing an input process and an output process at the same time.
(3) Pertaining to either input, or output, or both.

Input/Output Channel Controller (IOCC): A hardware component that supervises communication between the input/output bus and the processor.

Input-Output Code Number (IOCN): A value supplied by the virtual machine to a VRM component. This number uniquely identifies the code associated with a component and can be considered a module name.

Input-Output Device Number (IODN): A value assigned to a device driver by the virtual machine or to a virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

I/O device: An I/O device is an I/O adapter or option which is able to provide or receive data under the control of the system DMA controller or an alternate controller.

Instruction Address Register (IAR): A system control register containing the address of the next instruction to be executed. The IAR (sometimes called a "program counter") can be accessed via a supervisor call in supervisor state, but cannot be directly addressed in problem state.

Interface: A device that alters or converts actual electrical signals between distinct devices, programs, or systems.

Interleave: To arrange parts of one sequence of things or events so that they alternate with parts of one or more other sequences of the same nature and so that each sequence retains its identity.

Interrupt: (1) To temporarily stop a process.

(2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission.
(3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

Memory Management Unit (MMU): Hardware that manages virtual memory by providing translation from a virtual address to a real address.

Microcode: (1) One or more microinstructions.
(2) A code, representing the instructions of an instruction set, implemented in a part of storage that is not program-addressable.

Microinstruction: (1) An instruction of microcode.
(2) A basic or elementary machine instruction.

Minidisk: A logical subdivision of a real disk that has its own virtual device address.

Module: (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs are assembled separately, then linked to make a complete program.

Multiprogramming: (1) Pertaining to the concurrent execution of two or more computer programs by a computer.
(2) A mode of operation that provides for the interleaved execution of two or more computer programs by a single processor.

Operating System: Software that controls the execution of programs; an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

Page: A fixed-length block of instructions, data, or both, that can be transferred between real storage and external page storage.

Page Fault: A program interruption that occurs when a page of memory not in real storage is referred to by an active page.

Paging: The action of transferring instructions, data, or both between real storage and external page storage.

Parallel: (1) Pertaining to the concurrent or simultaneous operation of two or more devices, or to the concurrent performance of two or more activities.
(2) Pertaining to the concurrent or simultaneous occurrence of two or more related activities in multiple device or channels.
(3) Pertaining to the simultaneous processing of the individual parts of a whole, such as the bits of a character and the characters of a word, using separate facilities for the various parts.
(4) Contrast with serial.

Polling: (1) Interrogation of devices for purposes such as to avoid contention, to determine operational status, or to determine readiness to send or receive data.
(2) The process whereby stations are invited, one at a time, to transmit.

Port: An access point for data entry or exit.

Privileged Instructions: System control instructions that can only run in the processor's privileged state. Privileged instructions generally manipulate virtual machines or the memory manager; they typically are not used by application programmers.

Privileged State: A hardware protection state in which the processor can run privileged instructions. The processor's privileged state supports the virtual machine's VRM state.

Problem State: A state during which the CPU processing unit cannot execute privileged instructions. Most programs written to perform tasks or solve problems run in the problem state.

Process: (1) A sequence of discrete actions required to produce a desired result.
(2) An entity receiving a portion of the processor's time for executing a program.

Program: A document containing a set of instructions, conforming to a particular programming language syntax. Programs perform processes and are represented by process objects when active (i.e., when they are executed).

Program Status Block (PSB): A control block that describes a virtual interrupt condition.

Protocol: In data communications, the rules for transferring data.

Protocol procedure: A process that implements a function for a device manager. For example, a virtual terminal manager may use a protocol procedure to interpret the meaning of keystrokes.

Queue: A line or list formed by items waiting to be processed.

Real Address: A 24-bit address on the internal bus which will be applied to the storage by ROSETTA without modification by the translation mechanism of ROSETTA.

Register: A storage area, in a computer, capable of storing a specified amount of data such as a bit or an address.

Relative Address: (1) A means of addressing instructions and data areas by designating their locations to the Instruction Address Register or to some symbol.
(2) An address specified in relation to the contents of the Instruction Address Register or to a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

Reduced Instruction Set Computers (RISC): Processors with no sophisticated instructions in their instruction sets. They tend to perform almost as well as microcode, by using highly optimizing compilers, that take advantage of their high speed execution and their pipelined architecture.

ROM/BIOS: The ROM resident basic input/output system, which provides the level control of the major I/O devices in the computer system.

Routine: A set of statements in a program causing the system to perform an operation or a series of related operations.

Run-time Environment: A collection of subroutines that provide commonly used functions for system components.

Second Level Interrupt Handler (SLIH): A routine that handles the processing of an interrupt from a specific adapter. An SLIH is called by the first level interrupt handler associated with that interrupt level.

Sector: (1) An area on a disk track or a diskette track reserved to record information.
(2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

Segment: A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

Semaphore: Entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

Serial: (1) Pertaining to the sequential performance of two or more activities in a single device.
(2) Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.
(3) Pertaining to the sequential processing of the individual parts of a whole, such as the bits of a character or the characters of a word, using the same facilities for successive parts.
(4) Contrast with parallel.

Server: A program that handles protocol, queueing, routing, and other tasks necessary for data transfer between devices in a computer system.

Stack: An area in storage that stores temporary register information and returns addresses of subroutines.

Supervisor: The part of IBM RT PC's control program that coordinates the use of resources, and maintains the flow of processing unit operations.

Supervisor Call (SVC): An instruction that interrupts the program being executed and passes control to the supervisor so it can perform a specific service indicated by the instruction.

System Unit: The part of the system that contains the processing unit, the disk drive and the disk, and the diskette drive and diskettes.

Task: A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

Translation Lookaside Buffer (TLB): Hardware that contains the virtual-to-real address mapping.

Trap: An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

Unprivileged State: A hardware protection state in which the processor can only run unprivileged instructions. The processor's unprivileged state supports the virtual machine's operating system state and problem state.

Virtual Address: A 32-bit address on the internal bus intended to be translated by MMU.

Virtual Device: A device that appears to the user as a separate entity but is actually a time-shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

Virtual Machine Interface (VMI): A software interface between IBM RT PC workstations and operating systems. The VMI shields operating system software from hardware changes and low-level interfaces and provides for concurrent execution of multiple virtual machines.

Virtual Resource Manager (VRM): A set of programs that manage the hardware resources (main storage, disk storage, display stations, and printers) of the system so that these resources can be used independently of each other.

Virtual Storage: Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

Word: A contiguous series of 32 bits (four bytes) in storage.

GG24-3024-00 **READER'S COMMENT FORM**

IBM RT PC System Architecture

You may use this form to communicate your comments about this publication, its organization, or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment include: Clarity, Accuracy, Completeness, Organization, Coding, Retrieval, Legibility.

If you would like a reply, please give your name, company, mailing address and date:

____

____

____

____

What is your occupation? ______________________________

Most recent Newsletter associated with this publication: ____________

Thank you for your cooperation.

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM International Technical Support Center
Department 948, Building 808
11400 Burnet Road
Austin, Texas 78758

Fold and tape Please Do Not Staple Fold and tape

IBM®

GG24-3024-00

IBM RT PC System Architecture GG24-3024-00 PRINTED IN THE U.S.A.

IBM

GG24-3024-0