Burroughs

AN INTRODUCTION TO THE BURROUGHS 5500

M. H. J. BAYLIS

Contents

1. Summary

The B5500 reference manual is 160 pages long with few redundant words. This description attempts to cover the same span in a shorter space by jumping over much of the detail.

2. Introduction

The B5500 is a problem language oriented machine. The hardware provided was designed with Algol, Fortran and Cobol in mind. Important features are that it is a 48 bit word machine with 12 bit instructions, and a complex CPU operating in word or character mode. In word mode there is a dynamic stack, the top two words of which are the arithmetic registers. The Master Control Program (MCP) is a general multiprogramming operating system; its interaction with a running program also provides automatic handling procedures to meet most processing conditions - to this extent it provides facilities which all the compilers use. Programming the machine in basic code is strongly discouraged; although the description here is necessarily at this level.

3. System description

3.1 Major units

Central control unit, consisting of store access controller, input/output controller and system controller. The latter includes the master timing clock, real time clock and an interrupt system.

Processors. Maximum of two. If one, it will be operating in normal state or control state (interrupts inhibited, some extra functions and addressing structures active). If two, one will always be in normal state, so all interrupts go to the designated processor.

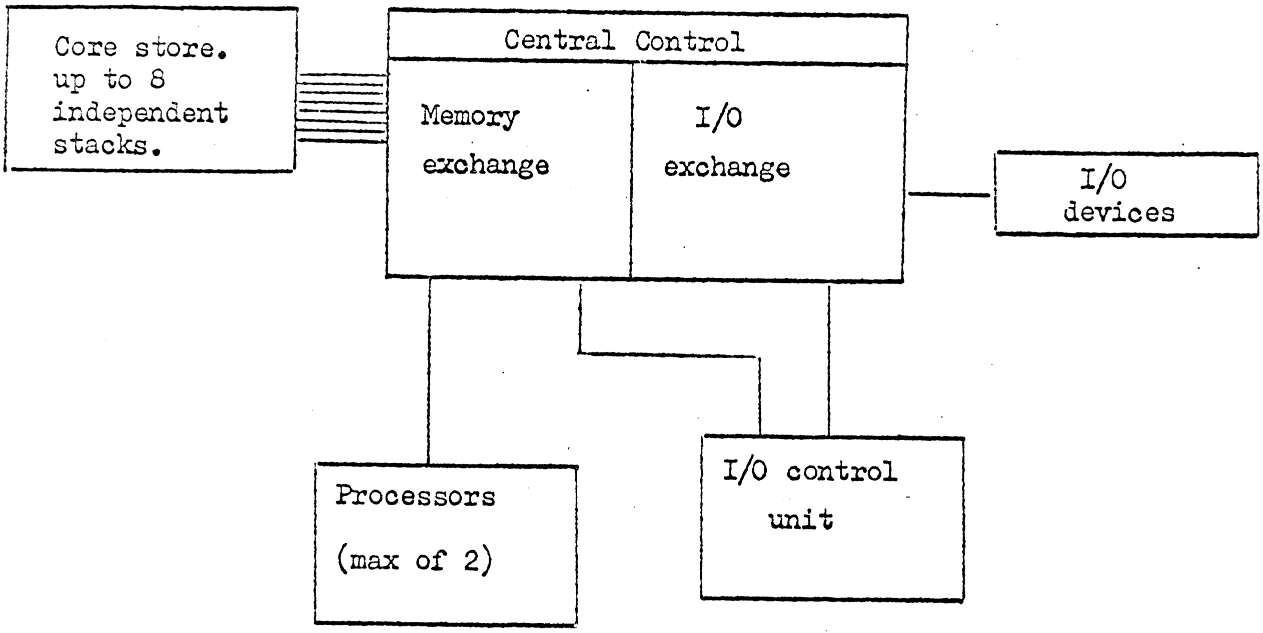
Memory modules. Up to 8 modules of 4K by 48 bits, either 6 μs or 4 μs cycle time.

Display and distribution unit.

3.2 Peripherals

type	max.	capacity/speed.
drum	2	32K
I/O channels	4	
mag tape	16	up to 72 Kc/s
line printers	2	up to 1040 cpm, 132 char/line.
card readers	2	up to 1400 cpm,
card punch	1	300 cpm,
tape readers tape punches	3	
disc file controllers	2	
various discs		

various communications equipment, multiplexors, teletypewriters etc.



4. Central processor outline description

Instructions (syllables) are 12 bits long, 4 to a word. In word mode, operands are 48 bit floating point numbers, on which arithmetic is performed in a parallel unit. In character mode, a word is regarded as eight 6-bit BCD characters. Operators work with single characters or one to six bits of a characters, using a serial 6-bit arithmetic unit.

The machine is clocked at 1Mc/s and there is a degree of instruction overlap both in the store accessing and execution stages. In word mode, average times are add 3 μs, multiply 30 μs. There is a parity bit with every 48 bit word.

There are up to four I/O control units which are independent, each able to drive any channel to a peripheral. Each control unit has a 48 bit buffer, 6-bit characters are parity checked, code translation is automatic and store access is by cycle stealing.

Three areas are associated with the computation part of a program. These are the Program Reference Table (PRT) which contain constants and variables, the stack and the program segment string. In word mode, all arithmetic is done at the top of the stack and it is necessary to think in Polish notation.Relative addressing is used to access all these three areas. Operands in general are numbers or descriptors. The latter describe information; for scalars they simply give an address (and type), for vectors they give address and length etc. They are discussed later. Some of the central registers are explained below; their functions are elaborated on here and later.

Register.	Width.	Function.
P	48	current word in program segment string, containing 4 syllables.
T	12	the syllable from P being executed
C	15	the address of P
L	2	address of T within P
NSCF	1	normal state ff
SALF	1	sub-program level ff:, indicates the program has entered at least one

CWMF	1	character word mode ff. Set by a basic instruction.
I	7	Interrupt register. Switches to control state after execution of current syllable if any bits set.
F	15	Used to record the current stack base, which alters on subroutine entry and exit.

These registers operate differently in word and character mode.

Register	Width	Word Mode	Character mode.
A	48	top word of stack	8 chars. of source string.
B	48	second word of stack A and B are the arithmetic registers. Most operations leave the result in B, with A invalid.	8 chars. of destination string.
S	15	address of top word of the stack area in core store. Ideally the stack runs A,B,(S),(S-1)... but after various operations A or B or both may be invalid; before the next operation A, B or both may be reloaded from the core part of the stack and S adjusted accordingly.	address of the destination string.
AROF	1	A valid/invalid.	
BROF	1	B valid/invalid. These two flip flops control the top of the stack logic.	
X	39	extension of B in some arithmetic operations.	contains a "loop control" word.
M	15	(i) for relative address developments. (ii) during arithmetic, for overflow bits and special counts.	address of source string.
R	9	address of base of PRT (m.s. 9 bits of 15 bit address)	bits 4-9 form the "tally" register.
Q12F	1	Mark stack ff (MSFF) see later	True/False ff. (TFFF) gives result of last conditional test.

These registers are for character and bit manipulation, and are also used in both modes.

Register	Width	Function
Y	6	contains one char. from A
Z	6	contains one char. from B Y and Z are the inputs for the character serial adder.
G	3	address of char. position in A
K	3	address of char. position in B
H	3	bit address of character addressed by G
V	3	bit address of character addressed by K H and V count 0-5
N	4	records octal shifts on B

5. Word mode

Syllables are grouped into four categories, recognised by the two l.s bits. (digits 10,11)

(i) Literal call. bits 0-9 contain an integer in the range 0-1023. Thus the instruction "LITC (literal call syllable) 16" would have the effect of setting A = 16. If A was previously valid then A→B first. If B was also previously valid then S+1→ S and B→ (S), i.e. the stack is pushed down.

(ii) Operator syllables. bits 0-9 define the type of operation. For single length arithmetic, the operands are A and B, the result goes to B and leaves A clear, and AROF is set so B is the new stack top. For double length arithmetic, one operand is A and B, the other is the top two words in the core part of the stack i.e. (S) and(S-1). The result is left in A and B, with S-2→ S.

(iii) Operand call. To place an operand on the stack.

(iv) Descriptor call. To place an address on the stack.

These two use bits 0-9 as indices for relative addressing and to give the base address to be used. Both need rather full descriptions, which are given later.

Relative addressing. In program level (SALF=0) the base addressis always the PRT base, held in the R register. The index is then either the 10 bits in the syllable or the l.s 10 bits in A, depending on the function. The absolute address is formed in M by addressing R and A (preloading A from the syllable where necessary). In sub-program level (SALF=1) the base can be R (for the PRT), F (for within the stack) or C (the current program word address). If R or C, the index can be positive or negative. These bits to decide base and sign are taken from the 10 available, leaving correspondingly fewer for the indexing.

Normal word mode addressing. S is always valid and controls the stack in core store, In general, B is always loaded and stored to and from the address in S. M is used to access any other operand needed, either by it containing a developed relative address or by transferring to it an address from A.

There are three types of word that occur in the PRT and hence in the stack. These are operands, descriptors and control words.

Descriptors describe some data, a program or an operation. They only occur in word mode, and there are four types, data descriptors, program descriptors, I/O descriptors and I/O result descriptors, all of 48 bits. The bits are used as follows (except for I/O results)

bit	use
0	=1 if descriptor or control word
1	=0 if descriptor
2	presence bit. If = 1 the information is present in core store.
8-17	word count field. (=0 if only 1 word)
19	=1 if data is type integer.
20	=1 if more than one descriptor referencing multiple areas (continuity bit)
33-47	absolute address of information.

Data areas are referenced indirectly through descriptors in the PRT. If the area is one word, the descriptor contains its address, if a vector it contains the base address and word count. For higher dimensioned arrays, there is a "Mother" descriptor giving the base address of a table of descriptors (dope vectors). These point on to more descriptors if there are more dimensions and eventually to the array elements, the word count at each level being equal to the size of the following level dimension. Any number of dimensions is allowed.

A subroutine is entered by a program descriptor (placed in the PRT during compilation). This descriptor contains a mode bit (to show whether word or character mode), and argument bit (set if arguments are needed) and the starting address of the subroutine. There is also a presence bit; if not present the MCP will be interrupted to load the relevant information.

Because subroutines also have to use the stack, and may be recursive, various facilities are provided. We have already mentioned SALF, which signifies sub-program level and allows additional relative addressing. Before entering a subroutine, the top of the stack is marked with a Mark Stack Control Word (MSCW). The parameters for the subroutine are then placed on the stack. A return Control Word (RCW) is automatically placed on top of these on entry.

The MSCW causes the A and B registers to be pushed into store, if necessary, and it contains the current values of R, MSFF, SALF and F. F was initially pointing at the base of the stack. Now it is set to the address of the word in which the MSCW is stored ready to be the stack base for the current subroutine. The RCW contains the return address and various registers (H,V,G,K,F,C). Return is automatic (via a return normal or exit operator) and the registers are restored from the RCW and MSCW.

We can now finish the description of the operand call and descriptor call syllables.

Operand call (OPDC). After accessing the operand address, the following actions can occur, depending on the type of object fetched.

(i) if an operand it goes to the stack top.

(ii) if a control word it is treated as an operand.

(iii) if a data descriptor, the word addressed by the descriptor will be placed on the stack.

(iv) if a program descriptor, the program will branch to a subroutine, placing a RCW on the stack.

Descriptor call (DESC).

(i) if an operand, causes a data descriptor to be generated that contains the absolute address and placed on the stack.

(ii) if a data descriptor, it is placed on the stack.

(iii) if a program descriptor, the action is as if the instruction was OPDC.

During these calls, rather a lot of logic is involved, which takes four pages of flow diagrams in the manual's explanation.

Without going into detail, rigorous hardware checks are made at each step on the validity of the information. For example, on accessing a data descriptor with a positive field count, the top of the stack (which is at B or (S) at this stage) is taken to contain the modifier. If this is not in integer form it is made so by forcing the exponent to zero. If this does not cause overflowand the resultant integer is less than the field count then the descriptor is altered so that the base address it contains is modified (and its field count may be set to zero). The resultant descriptor may then be either the end result or the address of the required operand.

The operators in word mode are as follows:-

Arithmetic

Add, subtract, multiply, divide, integer divide and remainder divide.

The first four can be single or double length.

Logical:-

And, or, equivalence (which is unusual I suppose) and logical negate.

Relational:-

These also use A and B as operands, but the result is a single bit in B, set to a one if the relationship is true. The instructions are of the form B ⟨relationship⟩ A, and are:-

$>, \geq, = \leq, <, \neq$

Branch:

These may branch absolutely or relatively. Conditional branches test the bit in B left by the relation operators. Branch unconditionally forward/backward. Branch conditionally forward/backward. Branch return - for sub-program exit. Non zero field branches - there are four instructions of this type which test any specified field of B (1 to 15 bits in length) for zero.

Store:

These orders exist in two forms. Both expect the top stack word to be an address and the next to be the operand. The 'destructive' form removes both from the stack, the other removes only the address. The address may be absolute or relative. Store, Convert to integer and store.

Bit Operators:

Dial A, dial B. These orders select a bit position within a character in A and B. Transfer bits. A field from A (up to the full word) is transferred to B, starting and going to the bit addresses previously dialed. Compare fields equal, low. These two comparisons are again over a specified field starting from the dialed bit addresses in A and B. Set/reset A flag bit. This flag bit is concerned with addressing technique. Set/reset/change A sign bit. Variable field isolate. This extracts a field from A and leaves it as a right justified number on the stack. Transfer fields. There are three orders which transfer fixed fields from A to B.

Subroutine:

Mark stack. Constructs a MSCW on the stack and sets F to its address. Exit, return normal, special return. These are all concerned with sub-program exits.

Enter character mode.

Stack.

Exchange. Swaps A and B - frequently needed to reposition a variable and its address in preparation for storing.

Duplicate. The top word is duplicated.

Delete top of stack.

Miscellaneous

Load operator, index, construct operand call, construct descriptor call, communication operator, program release, store for interrupt operator, conditional halt, set variant, etc. Most of these are too involved for this description.

6. Character mode

Character mode operation is always at sub-program level. The functions of various registers are different and there is a completely different list of operators. The primary areas of memory used are the source string and the destination, both of which can be regarded as continuous strings of characters or character positions. It is possible to skip characters in either string, compare fields, add fields, place results etc.

As listed earlier, the source string is associated with A, the destination with B. All processing between the two strings uses a 6-bit adder and the Y and Z registers. Character mode addressing uses G,H,K and V along with S and M.

Character mode syllables use the l.s. 6 bits determine the operation, and the m.s 6 bits as a repeat field. This repeat field defines the number of characters or bits with which the syllable will act on, up to a maximum of 63.

When a group of syllables is to be repeated more than once, a loop is formed using two syllables - the begin loop and end loop syllables. (BNS and ENS). BNS has the effect of setting up a loop control words (LCW) in X, and ENS uses this to cycle back round the loop. The LCW is stored in X, its previous contents being stacked as loops can be nested to any depth, and contains the current value of control (L and C), the repeat count and the address of the previous control word.

Character operators

To recap first:

M selects the word from the source string to be placed in A.

G points at the character within the word.

H points at the bit within the character.

MGH can be regarded as a single register in so far as overflow from H (which counts 0-5) causes G to be counted up 1, and overflow from G counts M up one.

Similarly for the destination string we have SKV associated with B.

At no time is it possible to have M=S.

Transfer: these transfers may occur from any character position.

Transfer source characters, transfer program characters, transfer zone part of characters, transfer numeric part of characters, transfer blanks for non-numerics. This last operator operates on the destination string and, up to the limit of the repeat field, replaces non-numeric characters by 00's until the first numeric one is encountered.

Transfer words is the final transfer operation, this one starts on word boundaries.

Tests:

Test operators test a character or bit in the source stream against a predetermined character or bit held in the syllable repeat field. The tests are > , ≥ , =, ≤ , < , ≠ and, 'test for alphanumeric' for characters and 'test bit ', which sets the true/false flip flop for bits.

Comparisons:

For comparing fields, which can start at any bit position. Word boundaries are ignored. The address registers are always advanced by the field length although the comparison result may have been determined earlier.

The tests are > , ≥ , =, ≤ , < and ≠

Jumps:

Jump forward/backward, conditionally on the TFFF or unconditionally. Here the C,L registers are modified by the contents of the repeat field. Begin loop, and loop have been explained. Jump out-of-loop conditionally or unconditionally is self explanatory.

Skips:

There are six orders - skip forward/backwards on source/destination strings in terms of characters/bits. The skip amount is given by the syllable repeat field.

Address:

There are ten orders to store and call addresses in and from the stack, and to set addresses from character strings.

Arithmetic:

Field add/subtract. These orders operate on character strings of length given by the repeat field (i.e up to 63 characters in width), placing the result in the destination field.

Conversion:

Input convert converts up to eight decimal characters to a binary integer, output convert does the reverse.

Miscellaneous:

'set/reset' bit' sets up to 63 bits to 1/0 in the destination string starting at the selected bit address, 'set/increase/store tally' manipulates the l.s 6 bits of the R register. 'Exit character mode'.

7. Interrupt handling

Interrupts switch the machine from normal state to control state, in which state all the normal operations are still available, plus a few additional ones. In control state, interrupts are recorded but do not interfere with computing until the current interrupt has been processed.

There are two classes of interrupt, processor independent and processor dependent.

The independent ones are:-

(a) time interval - for logging and checking program running time.

(b) other processor busy - to detect errors in the other processor, if present.

(c) line printer end-of-line signal.

(d) I/O channel busy

(e) I/O channel operation complete

(f) keyboard interrupt - for operators input

(g) disc file check complete

The dependent ones are:-

(a) core store parity

(b) invalid address

(c) request for program to enter control state

(d) flag bit - construct obtained instead of an operand

(e) continuity bit - multiple descriptors

(f) invalid index - i.e. exceeds field count

(g) exponent underflow

(h) exponent overflow

(i) integer overflow, detected on the automatic conversion of floating point to integer form in preparation for modifying.

(j) division by zero

(k) program release - indicates an I/O area of store is available.

(l) stack overflow - the stack is positioned below the PRT and S is checked against R, the PRT base, whenever numbers are added to the stack.

(m) presence bit - program has referred to information legally which happens to not be in core store at the time.

The I and IAR registers

The 7 bit I register is split into two halves. The top 3 bits are set by core store parity, invalid address and stack overflow; these are the three highest priority interrupts and more than one might occur at the same time. The other 4 bits are coded, and the number set indicates one of the other interrupts classes described above - only one of which can occur at a time. A hardware register IAR (interrupt address register) is automatically set to a predetermined value for each detected interrupt, or to the highest priority one if more than one interrupt occurs. When IAR has been accessed by the control routine, it resets itself to the next priority interrupt address and so on until there are none left. These are reserved words in the core where these interrupt addresses are kept. As well as setting IAR, the control logic forces the instruction "store for interrupt"(SFI) to be executed as the next syllable. There is often a set of interrupts within a class, each with its own entry address (e.g. 4 for the 4 I/O channels).

SFI causes all the relevant registers (different in the two modes) to be recorded in control words which are placed on the stack. It then generates an interrupt control word and return word on the stack and initiates the interrupt processing routine. This exits by an instruction which restores the central registers and allows the object program to continue correctly.

I/O processing is not discussed here; the peripherals are driven by the use of control words in a way consistent with the philosophy already described.

8. Finale

The B6500, 7500 and 8500's are elaborations of the 5500, each extending the power and flexibility and keepting the emphasis on data structures. Thus the 8500 removes I/O to separate processors, for example, and has 52 bit words so that each 48 bit word has a 3 bit descriptive tag and parity check. It also has a more generalised addressing system so a large core store can be utilised, and instructions which are multiples of 6-bits in length.

I have not been able to write and run programs for this machine and I would like to inspect the generated code from high level languages. My impressions are of elegance stemming from an aesthetically pleasing machine structure, and I would expect the underlaying philosophy to survive for many years. In my view a range of machines with the same philosophy is a better proposition than a range which is aimed to be bit compatible.

I summarise below some of the interesting features of the system:

(i) a powerful instruction code, particularly for character and bit manipulation.

(ii) high bit utilisation leading to compact well organised programs.

(iii) the addressing techniques and store control system make pure procedures easy to write.

(iv) the segmented program structure gives many desirable multi-access features, for example in allowing shared routines and data.

(v) the protection and verification given by the descriptor logic detects diagnoses and avoids a large class of program errors.

(vi) the store management system could be extended to cover multi-level stores, and paging or some other automatic technique could replace the present MCP software without disrupting user programs.

(vii) because the design is based on high level language requirements, these languages should produce efficient programs.

(viii) the operating system itself has been implemented naturally in a higher level language, and the evidence is that this has been done cheaply, efficiently and quickly.

It is hoped that this introduction provides enough information to give a general impression and to make the reference manual more readable.

M. H. J. Baylis

References:
B5500 Reference Manual.

The Burroughs Scientific Processor (BSP)

DAVID J. KUCK, MEMBER, IEEE, AND RICHARD A. STOKES

***Abstract*—The Burroughs Scientific Processor (BSP), a high-performance computer system, performed the Department of Energy LLL loops at roughly the speed of the CRAY-1. The BSP combined parallelism and pipelining, performing memory-to-memory operations. Seventeen memory units and two crossbar switch data alignment networks provided conflict-free access to most indexed arrays. Fast linear recurrence algorithms provided good performance on constructs that some machines execute serially. A system manager computer ran the operating system and a vectorizing Fortran compiler. An MOS file memory system served as a high bandwidth secondary memory.**

***Index Terms*—Conflict-free array access, high-speed computer, parallel computer, pipeline computer, scientific computing, vectorizing compiler.**

I. Introduction

FROM the beginning of the Burroughs Scientific Processor (BSP) design activity, we attempted to develop a system with high performance and reliability that is practical to manufacture, easy to use, and produces high quality numerical results. It is not possible to substantiate the success or failure in achieving these goals because the product was cancelled before user installations were realized. However, a full prototype system was operational for several months on customer benchmarks and demonstrated the practicality of the architecture. A number of points of technical merit have surfaced and are presented herein.

Because of the market place for which the BSP was intended, we chose Fortran as the main programming language for the machine. This choice leads to a need for array-oriented memory and processor schemes. It also leads to various control mechanisms that are required for Fortran program execution. To design a cost-effective and user-oriented system, more than programming languages must be considered; characteristics of the types of programs to be run on the machine must be carefully considered. We have throughout the design effort paid attention to the syntax of Fortran and also the details of "typical" scientific Fortran programs.

In the past 10 years several high performance systems have been built, including the pipelined CDC STAR (CYBER-205) [12], CRAY-1 [22], MU5 [24], and TI ASC [29], as well as Burroughs Illiac IV (BBKK68) and PEPE [9], the Goodyear Aerospace STARAN [21], and the ICL DAP [10], all parallel machines. While parallelism and pipelining are effective ways of improving system speed for a given technology (circuit and memory family, etc.), they both have shortcomings. Some pipelines perform better, the longer the vectors are that they have to process. The performance of other pipeline systems depends on vector lengths matching high-speed register set sizes. Parallel systems perform best on vectors whose length is a multiple of the number of processors available. Either type of system performs adequately if vector sizes are very large relative to the machine, but as these systems are used in wider application areas, short vector performance becomes more important. Other limitations of most pipeline processors have been that the arithmetic operations to be pipelined can reasonably be broken into only a limited number of segments and that overlapping of several instructions in one pipeline leads to unreasonable control problems.

Another important characteristic of high performance machines is the level of the language they execute. The CDC STAR and TI ASC, for example, have in their machine languages scalar and very high-level vector instructions, while ILLIAC IV, on the other hand, has a traditionally very low level machine language. A high-level machine language that is well matched to source programs can make compilation and control unit design easier and also helps ensure high system performance. A difficulty of array instructions can be that the setup time for instructions effectively stretches the pipeline length out intolerably.

System Overview

In the BSP we have combined parallelism and pipelining, and have provided array instructions that seem well matched to user programs. The machine has a five segment, memory-to-memory data pipeline, plus earlier instruction-setup pipeline segments. The data pipeline executes instructions that represent whole array assignment statements, recurrence system evaluations, etc., in contrast to most machines in which one or two arithmetic operations may be pipelined together. The BSP has 17 parallel memories, 16 parallel processors, and two data alignment networks. Since most instructions can be set up in the control pipeline preceding the data pipeline, instruction setup overhead should be insignificant in most cases. Thus, we have attempted to balance those architectural features that can provide good speedups with various overheads that can degrade or ruin system performance.

Another important factor in most supercomputers is I/O speed. If a very high speed processor is connected to standard disks, system performance may collapse because of I/O bound computations. The BSP has a high performance semiconductor

Manuscript received June 5, 1978; revised December 3, 1981. This work was supported in part by the National Science Foundation under Grant US NSF MCS77-27910 and the Burroughs Corporation, Paoli, PA.

D. J. Kuck is with the Department of Computer Science, University of Illinois, Urbana, IL 61801.

R. A. Stokes is with the Burroughs Corporation, Paoli, PA 19301.

0018-9340/82/0500-0363$00.75 © 1982 IEEE

file memory. This is used as a backup memory and to provide a smooth flow of jobs to and from the BSP.

Certain technological decisions were dictated by a Burroughs parallel development of a standard circuit and packaging design called Burroughs Current Mode Logic (BCML). A relatively long clock period was established in order to reduce the number of pipeline segments (for both data flow and instruction setup), avoid the complexities of high frequency clock distribution, and to facilitate manufacturing and testing the machine. The memory cycle time, the time to align 16 words between memories and processors (in either direction), and the time for many processor operations are all one 160 ns clock period; two clocks are required for floating-point addition and multiplication. In terms of these major events per clock, we attempted to lay out an array instruction set whose performance in the final system could be easily estimated during the design period.

A very important point in predicting system performance, and hence rationally choosing between design alternatives, is the determinacy of the system's behavior. We attempted to remove as much uncertainty as possible by several design choices. First, a parallel memory system was designed that provides conflict-free access to multidimensional arrays for most of the standard access patterns observed in programs. For cost reasons a parallel memory is required to achieve adequate bandwidth, and our design guarantees that for most instructions the effective bandwidth will be exactly at its maximum capacity. Since array elements are accessed in a different order from that in which they are processed, data alignment networks are needed along the path from memory to the processors and from the processors back to memory. These alignment networks also operate in a conflict-free way for most common operations. Finally, to guarantee that the memory-to-memory data pipeline is seldom broken, an array instruction set was designed. For example, a single BSP instruction can handle a whole assignment statement (with up to five right-hand side arguments) nested in one or two loops. The instruction can represent a number of 16 element slices of the operands, as long vector operations are automatically sliced and the slices overlapped in the memory-to-memory pipeline. Furthermore, as one vector assignment statement instruction ends, the next one can be overlapped with it in the pipeline. So for short vector operations there is usually no problem in keeping the pipeline full, since several different Fortran level instructions may be in operation at once.

Thus, for a wide class of instructions it was possible to predict the system performance (up to the clock speed) very early in the design process (1973). Furthermore, it has been possible during the later stages of design to make tradeoffs in these terms. Array instructions have also been very beneficial in allowing logic designers and compiler writers to communicate with each other about their own design efforts and to make tradeoffs in concrete performance terms.

A Fortran compiler has been implemented that vectorizes ANS Fortran programs, thus allowing old programs to be run without expensive reprogramming efforts. Vector extensions to Fortran are also provided to allow users to "improve" certain parts of old programs, if desired, or to write new programs in efficient ways. The vectorizer not only handles array operations, it also substantially speeds up linear recurrences—as found, for example, in processes that reduce vectors to scalars (e.g., inner product, polynomial evaluation, etc.)—and effectively handles many conditional branches within loops.

For some applications, numerical stability is a serious problem. The BSP does high quality (approximate R*) rounding of its 36-bit mantissas and also provides double precision hardware operations. Furthermore, interrupts are generated for standard floating-point faults. Error detection and correction are provided throughout the system and automatic instruction retry is provided to ease the burden on the user in some cases.

The BSP and its file memory form a high-speed computing system that may be viewed as standing inside a computational envelope. This envelope is serviced by a *system manager* that can be a Burroughs B6700, B6800, B7700, or B7800. This front-end general-purpose system provides the following:

- compilation of BSP programs,
- archival storage for the BSP,
- data communication and time-sharing services to a user community,
- other languages and computation facilities.

Thus, a typical user will interactively generate compiled program and data files on the system manager, pass them to the BSP for execution, and have results returned via the system manager with permanent files maintained on the system manager's disks. Most job scheduling and operating system activities for the BSP are carried out on the system manager, so the BSP is dedicated to high-speed execution of user application programs.

II. System Overview

Fig. 1 shows a block diagram of the BSP and the system manager. The BSP itself consists of three major parts: the control processor, the parallel processor, and the file memory. In this section some characteristics of these parts of the BSP will be presented. In subsequent sections we will give more details about how they operate.

A. Control Processor (CP)

The control processor is a high-speed element of the BSP that provides the supervisory interface to the system manager in addition to controlling the parallel processor and the file memory. The CP consists of a scalar processor unit, a parallel processor control unit, a control memory, and a control and maintenance unit.

The CP executes some serial or scalar portions of user programs utilizing an arithmetic element (similar to one of the 16 arithmetic elements in the parallel processor) that contains additional capabilities to perform integer arithmetic and indexing operations. The CP also performs task scheduling, file memory allocation, and I/O management under control of the BSP operating system.

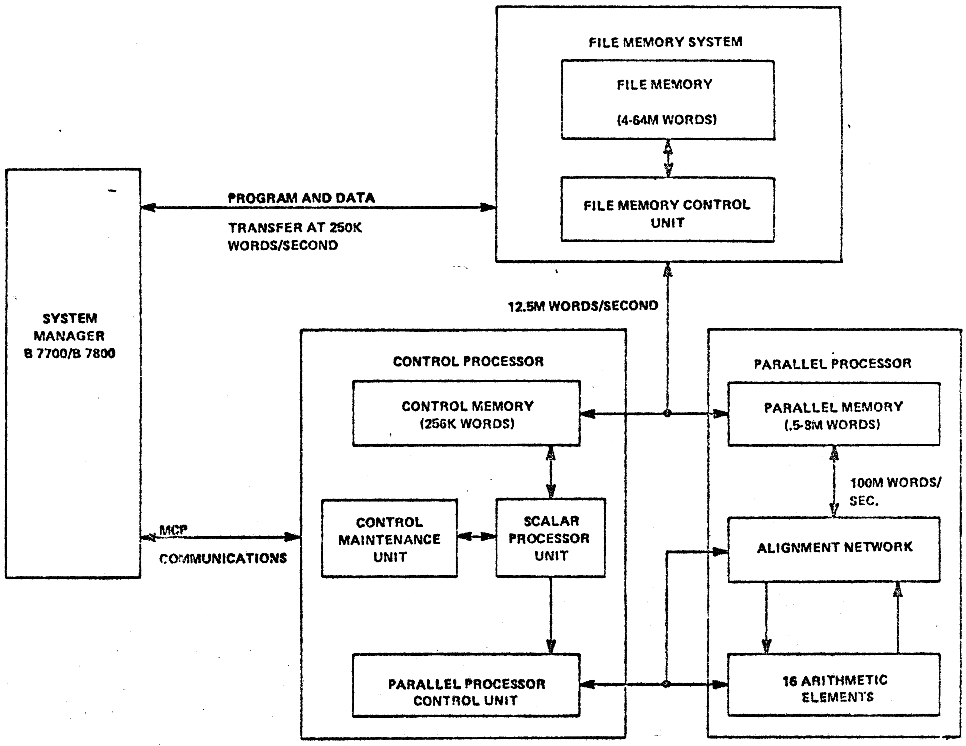


Fig. 1. BSP system diagram.

Scalar Processor Unit (*SPU*)*:* The scalar processor unit processes all operating system and user program instructions that are stored in control memory. It has a clock frequency of 12.5 MHz and is able to perform up to 1.5 million floating-point operations/s. All array instructions and certain scalar operations are passed to the parallel processor control unit, which queues them for execution on the parallel processor.

Parallel Processor Control Unit (*PPCU*)*:* The PPCU receives array instructions from the scalar processor unit. The instructions are validated and transformed into microsequences that control the operation of all 16 arithmetic elements in the parallel processor. Vectors of any length are handled automatically by the PPCU hardware, relieving the programmer and compiler of this burden.

Control Memory (*CM*)*:* The control memory is used to store portions of the operating system and all user programs as they are being executed. It is also used to store data values that are operands for those instructions executed by the scalar processor unit. The control memory is a 4K bit/chip NMOS memory with a 160 ns cycle time. Capacity of the memory is 256K words; each word consists of 48 data bits and 8 bits for error detection and correction. Four words are accessed simultaneously, giving a minimum effective 40 ns access time per 48 bit word.

Control and Maintenance Unit (*CMU*)*:* The control and maintenance unit serves as the direct interface between the system manager and the rest of the control processor for initialization, communication of supervisory commands, and maintenance. It communicates with the input/output processor of the system manager. The CMU has access to critical data paths and registers of the BSP, so that it can perform state analysis and circuit diagnostics under control of maintenance software running on the system manager.

B. Parallel Processor (*PP*)

The parallel processor performs array-oriented computations at high speeds by executing 16 identical operations simultaneously in its 16 arithmetic elements. Data for the array operations are stored in a parallel memory (PM) consisting of 17 memory modules. Parallel memory is accessed by the arithmetic elements through input and output alignment networks. A memory-to-memory data pipeline is formed by the five steps (fetch, align, process, align, store) and overlap in the pipeline provides significant performance benefits.

Parallel Memory (*PM*)*:* The parallel memory is used only to hold data arrays for the parallel processor and consists of 17 memory units, each of which may contain from 32K to 512K words, making a total of from 0.5 to 8 million words. It is a 4K bit/chip NMOS memory with a 160 ns cycle time as in the control processor memory. Each word contains 48 data bits and 8 bits for error detection and correction. The maximum rate of data transfer between the PM and the arithmetic elements is 10^8 words/s. The organization of the PM permits simultaneous access to most commonly referenced components of an indexed array, such as rows, columns, or diagonals. For some operands the compiler must choose between allocating storage in PM or CM, and performance can suffer if this is not done properly.

The Alignment Networks (*AN*)*:* The BSP has two alignment networks: the input alignment network for data fetching and the output alignment network for data stores. Both units contain full crossbar switching networks as well as hardware for broadcasting data to several destinations and for resolving conflicts if several sources seek the same destination. This permits general-purpose interconnectivity between the arithmetic array and the memory storage modules. It is the combined function of the memory storage scheme and the alignment networks that supports the conflict-free capabilities of the parallel memory. The output alignment network is also used for interarithmetic element switching to support special functions such as the data compress and expand operations and the fast Fourier transform algorithm.

Arithmetic Elements (*AE*): At any time all of the arithmetic elements are executing the same instruction on different data values. The arithmetic elements operate at a clock frequency of 6.25 MHz and are able to complete the most common arithmetic operations in two clock periods. Each arithmetic element can perform a floating-point add, subtract, or multiply in 320 ns, so the BSP is capable of executing up to 50 million floating-point operations/s. Each arithmetic element can perform a floating-point divide in 1280 ns and extract a square root in 2080 ns.

C. File Memory (FM)

The file memory is a high-speed secondary storage device that is loaded by the system manager with BSP tasks and task files. These tasks are then queued for execution by the control processor. The FM is also used to store scratch files and output files produced during execution of a BSP program. It is the only peripheral device under the direct control of the BSP; all other peripheral devices are controlled by the system manager.

The FM utilizes high-speed semiconductor memory as its storage medium; it combines a 1 ms access time with a 12.5M word/s transfer rate. Since it is entirely semiconductor, the reliability of the file memory is much greater than that of conventional rotating storage devices.

III. Languages and Their Translation

The BSP can be regarded as a high performance Fortran machine, although many of the ideas in the design are useful for various languages. In this section we present some details of the vector form language seen by the PPCU and discuss how vector forms can be obtained from Fortran programs by a vectorizing compiler. The essential elements of many numerical algorithms are represented by these vector forms and numerical programs in most languages could be reduced to the same set of vector forms. However, the large collection of Fortran programs existing in the numerical computation community has dictated that the primary language of the BSP be Fortran. Nevertheless, vector extensions to Fortran are provided so that users may write new programs in a more convenient language than Fortran, and also to allow faster translation and possibly faster executable BSP code. We conclude this section with a sketch of the BSP vector Fortran extensions.

A. Vector Forms

The parallel processor control unit sequences the five stages of the data pipeline: fetch, align, process, align, store (FAPAS). In this pipeline several instructions corresponding to Fortran statements may be in execution at one time. To clarify this process, several definitions are required; these will lead to an understanding of the parallel processor control unit and compiler.

The BSP has a total of 64 *vector forms* that may be grouped in the following four types:

1) array expression statements,
2) recurrence and reduction statements,
3) expand, compress, random store, and fetch, and
4) parallel memory transmissions to and from control memory and file memory.

Array expression statements include indexing and evaluating right-hand side array expressions ranging from monad to pentad (five right-hand side operands), plus the assignment of the resulting values to parallel memory. A separate vector form exists for each possible parse of each right-hand side expression. The array operations are performed in an element by element fashion and allow scalars and array variables of one or two dimensions to be mixed on the right-hand side. For example,

```
      DO 5 I = 1, 30
         DO 5 J = 7, 25
5           X(I, J) = (A(I, J + 1) * 0.5 + B(I + 1, J))
                      * X(I, J + 1) + C(J)
```

would be compiled as a single vector form. This vector form can be regarded as a six-address instruction that contains the four array arithmetic operation specifications and the assignment operation.

Recurrence vector forms correspond to assignment statements with data dependence loops. For example,

```
      DO 3 I = 1, 25
3        Y(I) = F(I) * Y(I - 1) + G(I)
```

has a right-hand side that uses a result computed on the previous iteration. This recurrence produces an array of results, while others lead to a scala: result and are called *reductions*. For example, a polynomial evaluation by Horner's rule leads to the reduction

```
         P = C(O)
      DO 5 I = 1, 25
5        P = C(I) + Y * P.
```

Both of these are recurrences that can be represented by a linear system of the form $x = Ax + b$, where A is a lower triangular matrix with a single band, one diagonal below the main diagonal, and x is an unknown vector. We will refer to a linear recurrence of dimension n and order m as an $R\langle n, m\rangle$ recurrence, where n is the dimension of the matrix A and $m + 1$ is the bandwidth of matrix A. Thus, the above program leads to an $R\langle 25, 1\rangle$ system. Fast efficient algorithms exist for solving such systems and the $R\langle n, 1\rangle$ solver of [7] and [23] for small n and the $R\langle n, 1\rangle$ solver of [6] for large n have been built into the BSP. For wider bandwidth recurrences the column sweep algorithm [15] is more efficient and it is used in the BSP. However, the user is not concerned with any of these considerations, since the vector forms described have array control unit hardware for their direct execution, as we shall see shortly.

Note that all recurrences would have to be executed serially without these algorithms. With them, $R\langle n, m\rangle$ systems ($1 < m \leq 16$ will obtain speedups proportional to m. For $m = 1$ and n of moderate size (say, 50 to 100), a speedup of 5 to 6 can be obtained, with greater speedups for large n. By speedup we mean time reduction compared to a hypothetical BSP with just one AE.

The third type of vector forms involves various sparse array operations. For example, in the case of a Fortran variable with subscripted subscripts, e.g., $A(B(I))$, no guarantee can be made concerning conflict-free access to the array A. In this case the indexing hardware generates a sequence of addresses that allows access to one operand per clock and these are then processed in parallel in the arithmetic elements. These are called *random store* and *random fetch* vector forms. Sparse arrays may be stored in memory in a compressed form and then expanded to their natural array positions using the input alignment network. After processing, the results may be compressed for storage by the output alignment network. These are called *compressed vector operand* and *compressed vector result* vector forms and they use control bit vectors that are packed, such that one 48 bit word is used for accesses to three 16 element vector slices.

A list illustrating the above three classes of vector forms is found in Table I. The mnemonics and comments should give an idea of what these vector forms do.

The fourth class of vector forms is used for I/O. Scalar and array assignments are made to control memory and parallel memory depending on whether they are to be processed in the scalar processor unit or the parallel processor, respectively; however, it is occasionally necessary to transmit data back and fourth between these memories. Transmissions to file memory are standard I/O types of operations.

These four types of vector forms comprise the entire set of array functions performed by the BSP. At the vector form level, the array processor may be regarded as arbitrarily large; thus, vector code generation is simplified in the compiler because it can transform Fortran programs into objects that map easily into vector forms, as we shall see shortly. However, in most Fortran programs some parameters are not defined at compile time (e.g., loop limits), so some run-time source language processing remains. This is carried out by the scalar processing unit of the control processor, and when it is finished the parallel processor is controlled by a template sequencing mechanism (see Section IV-D).

Vector forms are very high-level instructions with many parameters. For example, an array expression statement vector form corresponds to an assignment statement parse tree and leads to the execution of up to four operations on operands that may be combinations of scalars and one- or two-dimensional arrays. The following is a sketch of how the scalar processing unit (SPU) initiates the execution of a vector form in the parallel processor control unit (PPCU).

Consider a triad vector form

$RBV, Z = (A\ op_1\ B) op_2\ C, OBV$

where Z, A, B, and C are vector descriptors, op_1 and op_2 are operators, and RBV and OBV are optional result and bit-vector descriptors, respectively. Bit vectors may be used to specify the elements of an array to be operated on or stored; they are optional (but not shown) for a number of the entries in Table I. In executing the triad, the SPU issues the following sequence of instructions that describe the vector form to the PPCU:

*V*FORM TRIAD, op_1, op_2
OBV
RBV
*V*OPERAND *A*
*V*OPERAND *B*
*V*OPERAND *C*
*V*RESULT *Z*.

The *V*FORM instruction contains bits that name the first template (see Section IV-D) to be executed, specify actual operator names, indicate the presence of bit-vectors, and specify the program countercontents; it also contains other synchronization and condition bits. The *OBV* and *RBV* descriptors give the bit-vector starting addresses and lengths. The *V*OPERAND and *V*RESULT instructions give the start of the vector relative to an array location, the location of the array, the volume of the array, the skip distance between vector elements to be accessed, and optionally the skip distance between the start of subsequent vectors in a nested pair of loops. The *V*FORM instruction is preceded by a *V*LEN instruction that specifies the level of loop nesting and array dimensions. At this point, all source language parameters have been bound and run-time source language processing ends. The remaining processing done by the PPCU, e.g., array bounds checking, is the same for all operations. We shall return to a discussion of template sequencing in Section IV.

B. Fortran Vectorizer

In ordinary Fortran programs it is possible to detect many array operations that easily can be mapped into BSP vector forms. This is accomplished in the BSP compiler by a program called the Fortran vectorizer. We will not attempt a complete description of the vectorizer here, but we will sketch its organization, emphasizing a few key steps. For more discussion of these ideas, see [15], [17], and [4].

First, consider the generation of a program graph based on data dependences. Each assignment statement is represented by a graph node, and directed arcs are drawn between nodes to indicate that one node is to be executed before another. Algorithms for data dependence graph construction are well known [1], [4] and will not be discussed here. It should be observed that some compilers have used naive algorithms, for example, checking only variable names. The BSP algorithm does a detailed subscript analysis and thus builds a high quality graph with few redundant arcs, thereby leading to more array operations and fewer recurrences.

TABLE I

Vector Forms

MONAD	It accepts one vector set operand, does one monadic operation on it and produces one vector set result.	$Z \leftarrow op\ A$
DYAD	It accepts two vector set operands, does one operation on them and produces one vector set result.	$Z \leftarrow A\ op\ B$
VSDYAD	It is similar to the DYAD except that operand B is a scalar.	$Z \leftarrow A\ op\ B$
EXTENDED DYAD	It accepts two vector set operands, does one operation and produces two vector set results.	$(Z1,Z2) \leftarrow A\ op\ B$
DOUBLE PRECISION DYAD	It accepts four vector set operands (i.e., two double precision operands), performs one operation and produces two vector set results.	$(Z1,Z2) \leftarrow (A1,A2)\ op\ (B1,B2)$
DUAL-DYAD	It accepts four vector set operands, does two operations and produces two vector set results.	$Z \leftarrow A\ op_1\ B$ $Y \leftarrow C\ op_2\ D$
TRIAD	It accepts three vector set operands, does two operations and produces one vector set result.	$Z \leftarrow (A\ op_1\ B)\ op_2\ C$
TETRAD1	It accepts four set operands, does three operations and produces one vector set result.	$Z \leftarrow ((A\ op_1\ B)op_2\ C)\ op_3\ D$
TETRAD2	It is similar to the TETRAD1 except for the order of operations.	$Z \leftarrow (A\ op_1\ B)op_2\ (C\ op_3\ D)$
PENTAD1	It accepts five vector set operands, does four operations and produces one vector set result.	$Z \leftarrow (((A\ op_1\ B)op_2\ C)\ op_3\ D)op_4\ E$
PENTAD2	It is similar to the PENTAD1 except for the order of operations.	$Z \leftarrow ((A\ op_1\ B)op_2\ (C\ op_3\ D))op_4\ E$
PENTAD3	It is similar to the PENTAD1 except for the order of operations.	$Z \leftarrow ((A\ op_1\ B)op_2\ C)\ op_3\ (D\ op_4\ E)$
AMTM	It is similar to the MONAD and is used to transmit from parallel memory to control memory.	$Z \leftarrow op\ A$
TMAM	It accepts 6 vector set operands from control memory to transmit to parallel memory.	Z ← A1(0,0), A2(0,0) A3(0,0), A4(0,0), A5(0,0), A6(0,0)
COMPRESS	It accepts a vector set operand, compresses it under a bit vector operand control and produces a vector set result.	X ← A, BVO
EXPAND	It accepts a vector operand, expands it under a bit vector control and produces a vector set result.	X ← V, BVO
MERGE	It is the same as the EXPAND except that the vector set result elements corresponding to a zero bit in BV are not changed in the parallel memory.	X ← V, BVO
RANDOM FETCH	It performs the following operation Z(j,k) ← U(I(j,k)), where U is a vector and I is an index vector set.	
RANDOM STORE	It performs the following operation X(I(j,k)) ← A(j,k), where X is a vector and I is an index vector set.	
REDUCTION	It accepts one vector set operand and produces one vector result given by X(i) ← A(i,0) op A(i,1) op A(i,2) op A(i,3) ... A(i,L), where op must be a commutative and associative operator.	

TABLE I (CONTINUED)

DOUBLE-PRECISION REDUCTION	It accepts two vector set operands (one double-precision vector set) and produces two vector results (one d.p. vector) given by $(X_1(i),X_2(i)) \leftarrow (A_1(i,0),A_2(i,0))$ op $(A_1(i,1),\ A_2(i,1))$ op ... $(A_1(i,L),\ A_2(i,L))$, where op must be a commutative and associative operator.
GENERALIZED DOT PRODUCT	It accepts two vector set operands and produces one vector result given by $X(i) \leftarrow \{A(i,0)\ op_2\ B(i,0)\}\ op_1\ \{A(i,1)\ op_2\ B(i,1)\}\ op_1 \ldots \{A(i,L)\ op_2\ B(i,L)\}$, where op_1 must be a commutative and associative operator.
RECURRENCE-1L	It accepts two vector set operands and produces one vector result given by $X(i) \leftarrow (\{\ldots\{(B(i,0)\ op_1\ A(i,1))\ op_2\ B(i,1)\}\ op_1 \ldots\}\ op_1\ A(1,L))\ op_2\ B(i,L)$ where op_2 can be ADD or IOR and op_1 can be MULT or AND.
PARTIAL REDUCTION	It accepts one vector set operand and produces one vector set result given by $Z(i,j) \leftarrow Z(i,j-1)$ op $A(i,j)$, where op must be a commutative and associative operator.
RECURRENCE-1A	It accepts two vector set operands and produces one vector set result given by $Z(i,j) \leftarrow \{Z(i,j-1)\ op_1\ A(i,j)\}\ op_2\ B(i,j)$, where op_1 can be MULT or AND and op_2 can be ADD or IOR.

As an example, consider the following program:

```
  DO 5 I = 1, 25
1     A(I) = 3 * B(I)
          DO 3 J = 1, 35
3             X(I, J) = A(I) * X(I, J - 1) + C(J)
5     B(I) = 2 * B(I + 1).
```

A dependence graph for this program is shown in Fig. 2, where nodes are numbered according to the statement label numbers of the program. Node 1 has an arc to node 3 because of the $A(I)$ dependence and node 3 has a self-loop because $X(I, J - 1)$ is used one J iteration after it is generated. The crossed arc from node 1 to node 5 is an antidependence arc [16] indicating that statement 1 must be executed before statement 5 to ensure that $B(I)$ on the right-hand side of statement 1 is an initial value and not one computed by statement 5. Arcs from above denote initial values being supplied to each of the three statements: array B to statements 1 and 5, and array C to statement 3. The square brackets denote the scope of loop control for each of the DO statements.

Given a data dependence graph, loop control can be distributed down to individual assignment statements or collections of statements with internal loops of data dependences. In our example there is one loop (containing just one statement) and two individual assignment statements. After the distribution of loop control, the graph of Fig. 2 may be redrawn as shown in Fig. 3.

The graph of Fig. 3 can easily be mapped into BSP vector forms. Statements 1 and 5 go into array expression statement vector forms directly since they are both dyads. Had they had more than five right-hand side variables, their parse trees would have been broken into two or more array expression statement vector forms and joined by a temporary array assignment. Statement 3 can be split into 25 independent re-

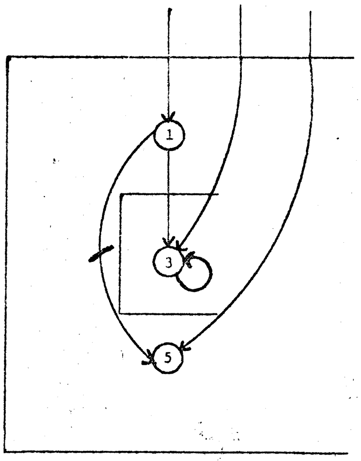


Fig. 2. Data dependence graph.

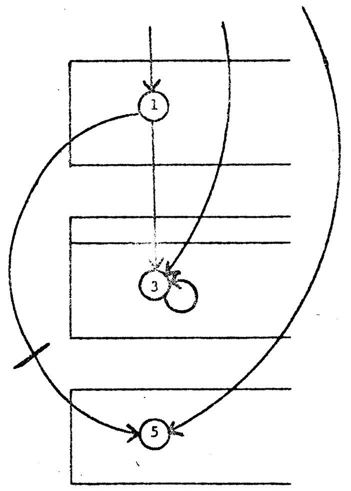


Fig. 3. Graph with distributed loop control.

currence systems, each with a bandwidth of 1 caused by the $J-1$ to J dependence, resulting in 25 independent $R\langle 35, 1\rangle$ systems, each of which maps into a recurrence vector form. Alternatively, it can be computed as a series of 35 array expression statement vector forms (triads). The maximum speed choice is made based on the loop limits.

The above cases are rather simple constructs, but they are typical of those found in existing Fortran programs. Several kinds of more complex cases are possible. If arrays with subscripted subscripts occur, they are compiled using the random fetch or store vector forms mentioned earlier. If recurrences with nonlinear right-hand sides occur, e.g., $X(I) = A(I) * X(I-1) * X(I-2) + B(I)$, then serial code is compiled. To this point we have ignored conditional statements inside loops, another problem that in the past has caused serial code to be compiled.

A number of IF statements can in fact be handled in parallel in the BSP. For a theoretical discussion of various types of IF's, see [14], where examples and measurements of the frequency of various types of IF's are presented. It turns out that many of the commonly found IF's can be handled in the BSP by using standard vector forms which include bit vectors. In this way, certain IF statements can be combined with assignment statements in a single vector form. For example, consider the following program:

```
      DO 1 I = 1, 92, 2
         DO J = 1, 46
1           IF(A(I, J).LT.0) B(I, J) = A(I, J) * 3.5.
```

This loop can be mapped into a single array expression statement vector form with bit-vector control that performs the parallel tests and makes the appropriate assignments to $B(I, J)$. By using loop distribution, many of the IF's found in ordinary Fortran programs can be transformed into such vector operations that allow substantial speedups on the BSP. Of course, there is also a residual set of IF's that must be compiled as serial code.

A traditional objection to array computers was that too many of the statements found in ordinary programs could not be vectorized and would have to be executed in a traditional sequential manner. In the BSP a combination of software and hardware innovations has led to a system that avoids most of these traditional objections. Of primary software importance are the distribution of loop control, fast algorithms to solve linear recurrences, and the vectorization of IF statements. Also of key importance is a good test for data dependence between subscripted variables, the appropriate introduction of additional subscripts to variables of lower dimension than their depth of loop nesting, and the transformation of scalar expressions inside loops as well as their substitution into subscripts. After a source program has been mapped into vector forms it is ready for execution.

C. Fortran Language Extensions

To provide users with language conveniences for writing new programs and to allow rewriting old programs that are difficult to vectorize automatically, several language extensions are being provided in the BSP software. Some of these extensions are also part of the proposed new ANS Fortran. The extensions may be categorized in four cases as array description, array operation, control, and I/O statements; we shall deal with them in that order, providing only a quick sketch of the ideas.

Arrays may be declared with the colon notation, using positive and negative subscripts, e.g., REAL $A(0{:}3, 0{:}3, -3{:}3)$ declares a $4 \times 4 \times 7$ array. Portions of declared arrays may be renamed for easy reference using an ARRAY statement as the following example shows:

```
REAL A(100, 100)
ARRAY ROW 2(J = 1:100) = A(2, J),
      DIAG(I = 1:100) = A(I, I)
```

identifies ROW 2 as a vector consisting of the second row of A and DIAG as the main diagonal of A. No storage is allocated or data are moved by an ARRAY statement, only additional array descriptors are created.

Array operations can be specified in various ways. If A and B are declared arrays, then $A = 0$ sets all elements of A to zero and $B = B + 1$ adds 1 to all elements of B. If A is two-dimensional, then $A(*, 0) = 2$ sets column zero of A to 2. Arithmetic, relational, and logical operators may be applied to pairs of arrays that are congruent, in which case element by element operations are performed. Furthermore, bit-vectors may be used to control array operations by use of the WHERE statement as follows. Assume that A and B are 500 element vectors, then

```
WHERE(A.GE.0) B = B + A
```

is equivalent to

```
   DO 10 I = 1, 500
10    IF (A(I).GE.0) B(I) = B(I) + A(I).
```

This is generalized to a block-structured WHERE DO that contains a sequence of OTHERWISE statements and ends with an END WHERE. PACK and UNPACK statements are provided to allow sparse arrays to be compressed and expanded, respectively; multidimensional arrays may be packed into vectors based on logical tests. There is also an IF-THEN-ELSE construct and an END DO that does not require a label in its DO statement. For scalars or congruent arrays, the exchange statement

$$A = = B$$

exchanges A and B in memory.

A collection of intrinsic functions is also provided, one set generalizes the standard scalar intrinsics to arrays (e.g., transcendental functions of the elements of an array) and the other set provides some standard array operations (e.g., dot product, matrix product, max of an array, etc.).

User programs can control I/O without supervisor intervention and without buffering by using the DIRECT statement which names an array that appears in a following READ statement. Execution of statements after the READ continues simultaneously with the input until the DIRECT named variable is encountered on the right-hand side of an assignment statement, at which point execution is suspended until the READ is completed.

IV. System Operation

In this section we give more details of the overall system operation. First, the parallel memory and its conflict-free structure are discussed (Section IV-A), and this is followed (Section IV-B) by the alignment networks that stand between the parallel memory and the arithmetic elements, which are discussed in Section IV-C. Section IV-D ties these components of the parallel processor together by detailing the template sequencing as carried out by the parallel processor control unit.

This also relates back to Section III-A, where it was pointed out that source programs are first mapped by software into vector forms which in turn are mapped by hardware into a sequence of templates. The template sequencing mechanism is one of the key points in achieving high system utilization through pipelining and overlap of vector forms in the BSP. This section concludes with a discussion of the high performance file memory used for secondary storage in the BSP.

A. Parallel Memory

The BSP parallel memory consists of 17 memory modules, each with a 160 ns cycle time; and since we access 16 words per cycle, this provides a maximum effective 10 ns memory cycle time. This is well balanced with the arithmetic elements which perform floating point addition and multiplication at the rate of (320 ns/16 operations) = 20(ns/operation), since each operation requires 2 arguments and temporary registers are provided in the arithmetic elements. Note that only array accessing (including I/O) uses parallel memory, since programs and scalars are held in control memory. Thus, perfect balance between parallel memory and floating-point arithmetic may be achieved for triad vector forms since three arguments and one result (four memory accesses) are required for two arithmetic operations. For longer vector forms, since temporaries reside in registers, only one operand is required per operation, so there is substantial parallel memory bandwidth remaining for I/O.

The total memory size ranges from 0.5 to 8 million 48 bit words. Eight parity bits provide single error correction and double error detection.

The main innovation in the parallel memory of the BSP is its 17 modules. In past supercomputers it has been common to use a number of parallel memory modules, but such memory systems are vulnerable to serious bandwidth degradation due to conflicts. For example, if 16 memories were used and a 16 × 16 array were stored with rows across the units and one column in each memory unit, then column access would be sequential.

Various storage schemes have been invented to avoid such memory conflicts. For example, arrays may be skewed [16] so that rows and columns can be accessed without conflict, and other related skewing schemes may be found with other useful properties. However, it is easy to show [5] that in general it is impossible to access rows, columns, and diagonals of square arrays without conflict if any power of two number of memory modules is used. Of course, various ad hoc procedures may be contrived, e.g., different skewing schemes for different arrays, but in the long run these are a compiler writer's (or user's) nightmare. A uniform, conflict-free procedure carried out by the hardware would be far superior for users of the system.

Early in the design of the BSP we decided to build the best possible parallel memory in this respect, and thereby avoid as many software implementation and performance problems as possible. For this reason we settled on a 17 memory module system that would provide conflict-free array access to most common array partitions, and yet have little redundant memory bandwidth since only one memory unit is unused per cycle.

With 17 memory modules it is clear that conflict-free access to one-dimensional arrays is possible for any arithmetic sequence index pattern except every 17th element. For two-dimensional arrays with a skewing distance of 4, conflict-free access is possible for rows, columns, diagonals, back-diagonals, and other common partitions, including arithmetic sequence indexing of these partitions [5]. The method extends to higher numbers of dimensions in a straightforward way.

One mundane characteristic of some Fortran programs can cause a problem here, namely, the use of COMMON in subroutine parameter passing. If used in the most general ways, this forces the storage of arrays in a contiguous way across parallel memory. In this case conflict-free access can still be guaranteed to any arithmetic sequence of physical memory addresses, as long as the difference between addresses is not a multiple of 17. This may force some array dimensions to be adjusted slightly for conflict-free access to all of the desired patterns.

To access parallel memory a set of 16 memory addresses must be generated. Assume that addresses are to a linear address space, i.e., multidimensional arrays have been mapped into a one-dimensional array, and that the array is stored across the memory modules beginning with module 0, through module 16, continuing in module 0, and so on. Then to access address α we must generate a *module number* μ and an *index* i in that module. These are defined by[1]

$$\mu = \alpha (\text{mod } 17)$$

$$i = \left\lfloor \frac{\alpha}{16} \right\rfloor$$

since there are 17 memory modules and we access 16 numbers (one for each AE) per memory cycle. Notice that this wastes $\frac{1}{17}$ of the address space, a minor penalty for the conflict-free access it provides. Address generation hardware for the memory system is somewhat complex, but can be done in parallel for a sequence of addresses in one clock using the scheme described in [18]. This hardware also generates indices to set the aligment networks appropriately for each memory access.

B. Alignment Networks

The separation of data alignment functions from processing and memory activities is another departure of the BSP from most previous computers. As discussed earlier, the BSP has an input alignment network (IAN) connecting parallel memory to the arithmetic elements, and an output alignment network (OAN) connecting the arithmetic elements either to themselves or to parallel memory. Alignment of the elements of two arrays is sometimes required by the parallel memory and sometimes required by program or algorithm constraints, as the following examples illustrate.

Suppose we want to add together two rows of a matrix, element by element. The origins of the rows will in general be stored in different memory modules, so one row must be shifted relative to the other to align them for addition. Now if we want to add the odd elements of one row to the elements of another row (half as long), the first row will have to be "squeezed" as well as shifted to align proper pairs of operands. Similar alignment problems arise in row-column, column-diagonal, etc., pairings. Since we must store arrays consistently in parallel memory, the output alignment network is used to satisfy

[1] We use $\lfloor x \rfloor$ to denote the integer part of x.

the storage requirements and indexing patterns of the variable on the left-hand side of each assignment statement.

The above uses of the alignment networks hold for recurrence and reduction vector forms. In these cases, data may be aligned after fetching via the IAN, but now the OAN is useful between processing steps. As a simple example, consider the summation of 32 numbers. This may be done in five steps, each step consisting of an addition followed by an output alignment network mapping of the AE's into the AE's in the form of a tree, which reduces the 32 numbers to one in $\log_2 32$ steps. Similarly, other reduction operations may be carried out using the OAN; these include such operations as finding the maximum or minimum of a set of numbers. Notice that for any such operations the reduction to a set of 16 numbers is carried out using all 16 processors, and after that the number of processors used is halved on each step.

In solving more general linear recurrences, a vector of results is produced, e.g., an $R\langle n, 1\rangle$ system leads to n results. Again, the OAN is used for an AE to AE mapping of intermediate results. Other important algorithms also require data alignment between operations; the FFT [20] and the Batcher merge and sort algorithms [2] are examples. These and other algorithms can be implemented directly in the BSP using microprogrammed AN sequencing patterns. Each alignment operation takes one clock in a FAPAS pipeline sequence. In array expression statement vector forms most are overlapped, while in recurrence vector forms the later alignments must be alternated with arithmetic.

In the course of some of the above algorithms, certain vector positions are vacated during the course of the computation, e.g., reduction operations reduce a vector to a scalar. An effective way of handling this is to have the IAN introduce *null elements* into the computation at appropriate points. In AE operations null elements are handled as follows:

$$\begin{aligned}\text{operand} &\leftarrow \text{null } \theta \text{ operand}\\ \text{operand} &\leftarrow \text{operand } \theta \text{ null}\\ \text{null} &\leftarrow \text{null } \theta \text{ null}\end{aligned}$$

for any operator θ. Memory modules block the storing of a null. Nulls are also used when a vector length is not equal to a multiple of 16, so the last slice of the vector is padded out with nulls by the IAN.

The alignment networks are constructed from multiplexers and are generalizations of crossbar switches. In addition to the permutation functions of crossbars, the alignment networks can also broadcast an input element to any selected set of destinations. Furthermore, the random store and fetch vector forms, as well as the compress and expand operations, use the alignment networks to carry out their mappings. Control of the AN's is closely related to PM control. As was pointed out in Section IV-A, memory addressing information and AN control indices are generated by the same hardware control unit.

The two alignment networks have similar control in that they are both source initiated; conceptually, for the IAN the memories specify which AE to transmit to and for the OAN, the AE's specify which memory they want to store to. In certain cases, e.g., random store and fetch, the AE's actually generate memory addresses as suggested here, whereas in the standard array expression statement or recurrence vector forms all of the addressing is carried out by a special control unit hardware. Conflict resolution hardware is provided to sequence certain alignments in several steps. For more AN details, see [18].

C. Arithmetic Element

The 16 arithmetic elements are microprogrammed, being sequenced by the parallel processor control unit using a wide (128 bit) microcode word. Besides the arithmetic operations expected in a scientific processor, the BSP has a rich set of nonnumeric operations that include field manipulation, editing, and Fortran format conversion operators. Floating-point addition, subtraction, and multiplication require two 160 ns clocks each, floating-point division requires 1280 ns, and the square root operation requires 2080 ns; the latter two use Newton-Raphson iterations that start with ROM values selected in each AE [11].

Single precision floating-point arithmetic is carried out using normalized signed magnitude numbers with 36 bits of mantissa and 10 bits of exponent, providing about 11 decimal digits of precision with a range between approximately 5.56×10^{-309} and 8.99×10^{307}. Four guard digits are retained within the AE and R* rounding is carried out using these four bits. Given that most alignment shifts are small [25], this should provide a good approximation of full R* rounding. Double-precision operations are carried out in the hardware (a double length product is always generated) about four times slower than single precision. The range of double precision numbers is the same as for single precision, and the precision is twice as great. Characters are stored as 8 bit EBCDIC bytes, packed six to a word.

Although they are not seen by users, each AE has a file of 10 registers, in addition to the standard registers used in the course of various operations. These are very convenient for holding intermediate results in the course of evaluating vector forms. The register assignment is done within the vector forms and the number of registers necessary to ensure high system performance was thus decided when the machine was designed. This is in contrast to building a machine first and then studying register allocation as a later compiler design question.

To add to the system reliability each AE contains residue checking hardware to check the arithmetic operations. Two-bit, modulo 3, residue calculations are carried out for each exponent and mantissa. To enhance the diagnosability of the entire system, the AE's (actually the AE-alignment network interfaces) contain Hamming code generators, detectors, and correctors for the data path loop from the AE's through the alignment networks and memory, and back to the AE's. This allows control information to be included in the Hamming code in order to check for failures in the control hardware of the parallel memory and alignment networks as well as the data paths mentioned above. The eight parity bits allow single error correction and double error detection. Most double-bit parity failures and all residue check failures lead to an instruction retry (see Section IV-D).

D. Template Sequencing

The execution of a vector form must be broken into a sequence of elemental array sequences called templates. This is done by the parallel processor control unit which issues a sequence of (one or more) *templates* to execute each vector form. The template provides the control framework in which vector forms are executed. Because it is desirable to overlap the execution of these template in the FAPAS pipeline, appropriate matching must be found in one template to accommodate the next template. For example, the fetching of operands for the next template may occur before the storing of the results from the previous template. In terms of such matching, *template families* have been defined with respect to the interface characteristics of their front and back templates. Each is said to have a front and back family number.

During execution the PPCU chooses templates on the basis of the vector form and the family numbers. For example, in executing a dyad followed by a triad, the PPCU will match the front family of the first triad template with the back family of the last dyad template. Then (assuming that there are more than 16 triad operations to perform) a second triad template will be chosen by matching its front family with the back family of the first triad template. Within three templates this reaches a cycle of one or two templates that repeats until the end of this particular vector form execution.

It is important to realize that the FAPAS overlap between templates as well as the arithmetic element register allocation for each one is done at machine design time. This leads to an *a priori* understanding of the machine's performance over a wide range of computations and also allows more vector operation overlap than might be possible otherwise, since all possible template combinations have been considered at machine design time.

This is in contrast to most previous high performance machines, wherein such overlap attempts are made at compile time or execution time. For example, in the CDC 6600 [27] and its successors, the control unit SCOREBOARD attempts run-time overlap and this is aided by compile-time transformations [26]. In the IBM 360/91 an overlap mechanism was built into the processor [28] in an effort to sequence several functional units at once. Most previous high performance machines seem to have overlap mechanisms that combine compile-time and run-time (control unit and processor) overlap decisions in ways similar to these. In some cases, however, certain common functions can be chained together (e.g., multiply and add for inner product in the CRAY-1), although in the CRAY-1, for example, intricate run-time considerations dictate whether or not chaining is possible [8].

The five-stage FAPAS pipeline may have four templates in process at once, and five more templates may be in various stages of setup in the PPCU pipeline that precedes execution. The amount of such overlap that exists at any moment depends, of course, on the width of individual templates. However, it is clear from an analysis of the templates that for most of the common Fortran constructs a very high percentage of processor and memory utilization can be expected in the BSP. Notice that there is overlap between templates of one vector form as well as overlap between different vector forms.

Due to the fact that the BSP executes a very high-level array language, complex hazard checking can be performed. As was mentioned in the discussion of template families, one template's fetches may begin before a previous template's store has been executed. In some cases, two previous stores may be pending while a third template's fetch is executed. This leads to *data dependence hazards* that must be checked before executing such memory accesses, i.e., if a fetch is for data that are to be stored by a previous template, the fetch must be delayed until after the store. Such hazard checking is carried out for up to two stores before a fetch by a combination of software and hardware in the BSP.

To clarify the above ideas, consider the following example program:

```
      DO 1 I = 1, 40
1     Y(I) = A(I) + B(I)
      DO 2 I = 1, 90
2     Z(I) = (C(I) + D(I)) * E(I).
```

This would be compiled as a dyad vector form followed by a triad; the dyad would be executed using 3 $\left(= \left\lceil \frac{40}{16} \right\rceil\right)$ templates and the triad using 6 $\left(= \left\lceil \frac{90}{16} \right\rceil\right)$ templates. Fig. 4 shows a FAPAS pipeline timing diagram for the execution of these templates.

In Fig. 4 notice that three distinct dyad templates are used:[2] dyad (2, 3), dyad (3, 6), and dyad (6, 7). The last memory cycle on clock 3 is provided for a previous template—some waste can be expected when beginning or ending a computation—but otherwise all memory cycles are occupied until clock 8. Notice also that after processing begins, the first three templates use $\frac{2}{3}$ of the processor clocks until clock 12.

The triad processing begins with three distinct templates: triad (2, 4), triad (4, 8), and triad (8, 8). It then continues in a steady state with triad (8, 8) templates. Except for one last clock at the dyad interface, the triad templates operate with total utilization of both the parallel memory and the parallel processor. Generally speaking, longer templates achieve very high utilization of the system hardware.

The total number of clocks required to execute this sequence of templates is 39, and since the original program contains 220 floating-point operations, the effective speed of this computation is greater than 35 million floating-point operations/s (Mflops). Assuming processing overlap at the beginning and end of this sequence, only 36 clocks are required for processing and on this basis a rate of more than 38 Mflops is achieved.

The use of vector forms and templates allows the easy implementation of a number of desirable features. The basic idea is that templates can be regarded as global microinstructions (the PPCU is microprogrammed) or "macroinstructions" that sequence the entire parallel processor. Consequently, even with

[2] The parenthesized numbers are the front family and back family identification numbers. The back family number of one must match the front family number of the next.

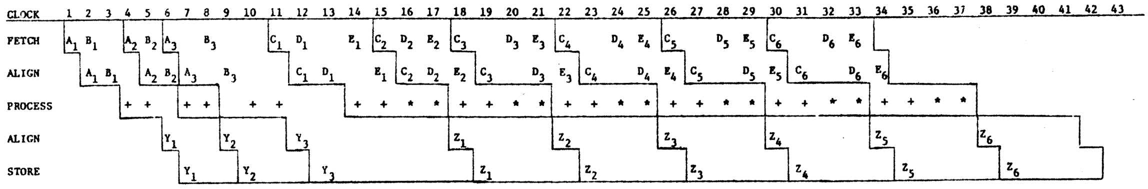


Fig. 4. Triad template sequencing.

a set of overlapped templates in progress, it is easy to delay the entire parallel processor. This is easiest to think of in terms of Fig. 4. For example, if an error occurs, a vertical slice is made through Fig. 4 at the end of some clock period and all further steps may be deferred until the interrupt is handled. Also, longer operations than addition and multiplication (i.e., division or square root) can simply be handled by extending the process step for sufficiently many clocks and deferring all other tracks in the FAPAS diagram, resuming them when the longer operation is complete; double precision is implemented in a similar manner. Furthermore, instruction retry is facilitated by this ability to break the FAPAS sequence at any step and back up to the beginning of a vector form, since memory stores are prevented in case of an error.

The distinction between data pipelining and overlap here and in other machines should be clear at this point. The BSP pipelines each template through the five FAPAS segments, in contrast to traditional pipelining of arithmetic operations; the individual operations being decomposed are array assignment statements, recurrences, etc., in the BSP. A sequence of templates can be likened to a sequence of additions in an arithmetic pipeline. On the other hand, most modern machines are overlapped in the sense that memory and arithmetic can be performed simultaneously, based on run-time lookahead. In the BSP the analogous process is overlap between vector forms, as was illustrated in Fig. 4.

Run-time lookahead schemes are limited by data hazards, jumps, and various other resource conflicts. By complex analysis of source programs, much more simultaneity is possible. For the BSP, many IF and GOTO statements are replaced by bit-vector modifications of vector forms, so even these difficulties of standard run-time lookahead schemes are often avoided.

Globally, the PPCU may be viewed as accepting a single stream of array instructions, namely, vector forms, and producing five streams of array instructions to sequence the FAPAS segments. Thus, in the terminology of [16], the BSP array processor is a SIAMEA (single instruction array/multiple execution array) machine. Of course, the scalar processor unit also may execute instructions from a given Fortran program simultaneously with the parallel processor.

E. File Memory

The BSP file memory consists of two sections, the file storage units and the file memory controller. This is the secondary memory of the BSP, with longer term storage being provided on conventional disks and other I/O devices attached to the system manager (see Fig. 1).

During the BSP design phase several semiconductor technologies showed promise as high-speed secondary memory devices, in competition with conventional head-per-track disks. Charge-coupled device (CCD) and random access memory (RAM) chips were considered because they provide very attractive characteristics, they were well along in development, and their price outlook seemed likely to be competitive with head-per-track disks. As a result, an MOS RAM was chosen to provide the BSP file memory with a very low latency and a high transfer rate, which is expected to perform with good reliability and maintainability at a reasonable cost.

The file storage unit is built with semiconductor high density memory devices that provide a transfer rate of two words every 160 ns with a maximum latency of less than 1 ms. Nonaddressed modules are operated at $\frac{1}{4}$ that clock rate to conserve power, yet provide the refresh needed for the volatile devices. Each file storage unit is organized in 4 M ($M = 2^{20}$) word sections and may contain 4 sections for a total of 16 M words of 48 data bits plus 8 parity bits. Two words are accessed in parallel for a transfer rate of 112 bits/160 ns $= \frac{2 \text{ words}}{160 \text{ ns}} = 12.5 \times 10^6$ word/s. A file memory system may contain up to 4 file memory units for maximum file memory capacity of 64 M (64×10^6) words. Thus, the file memory provides a backup storage of one to two orders of magnitude the size of parallel memory.

Consider the file memory in relation to a typical BSP computation. For example, a block of 32K words may be read at an overall effective rate of about 10^7 words/s. Since the BSP operates at a maximum of 50 Mflops, this means that only 5 floating-point operations need be performed per I/O word to balance the I/O and processing rates. To sustain this, output must be considered as well so the ratio becomes 10 floating-point operations per I/O word, for balance. These ratios seem well within the range of most large scientific calculations (whose ratios often range up to 100 or more).

The file memory stands between the BSP and the system manager. The file memory controller can transmit data to and from the BSP parallel memory or control memory as well as the system manager I/O processor (see Fig. 1), the latter at the 0.25×10^6 words/s maximum channel speed of the system manager. I/O instructions are passed between the BSP control processor and the file memory controller.

The file memory controller (FMC) contains buffer areas to match the file memory speed with that of the system manager. I/O requests from the BSP and system manager are queued in the FMC in a 32 entry queue. Normally, the slower system manager receives highest priority and the BSP requests are handled in a first-in, first-out manner.

The file memory can be addressed to the word level and a block to be transmitted can be any number of words. The FMC converts logical addresses into physical addresses. A logical address contains a file name, a starting address in the file, a block length, and a destination (or source) memory address. File protection is provided by FMC hardware that allows any combination of four access modes: system manager READ or WRITE and BSP user program READ or WRITE. When operating in problem state, BSP I/O instructions can be executed with no supervisor program intervention at all for error-free transmissions. Synchronization bit registers are provided that allow BSP user programs to test for I/O completion without supervisor program intervention. The FMC also contains hardware for I/O instruction retry for all errors not automatically corrected. Thus, a number of situations that might traditionally have required slow, operation system intervention are handled by FMC hardware in the BSP.

Task switching within the BSP takes less than one-half second, but the speed of flow of jobs from the system manager, through file memory, and into the parallel memory and control memory, depends on many details of individual jobs.

V. Conclusion

We have outlined the organization of the BSP, a high performance scientific computer. Its key features include high quality, fast arithmetic, conflict-free access to arrays in parallel memory, separate data alignment networks, a pipelined control unit that sequences a high-level data pipeline and can overlap the execution of its vector form language, and a semiconductor file memory for secondary storage. Many error-checking and correcting features are included throughout the system to enhance its reliability and maintainability. Software was provided on a system manager computer that handles most operating system functions and has a Fortran vectorizing compiler that can also handle vector extensions.

The BSP system design effort began in early 1973, and led to an operational prototype machine in 1978. The maximum system speed is 50 million floating-point operations/s. The system performed as a "class 6" computer by running the Department of Energy LLL loops at speeds in excess of 20 Mflops. In fact, its average speed is almost identical to the average speed of the CRAY-1 on these benchmarks. A major design goal was a system that could achieve a high sustained performance over a wide variety of scientific and engineering calculations using standard Fortran programs. It is estimated that 20-40 Mflops could be achieved for a broad range of Fortran computations.

Acknowledgment

The authors are grateful to the entire BSP design team for years of work in the development of this system.

References

[1] U. Banerjee, "Data dependence in ordinary programs," M.S. thesis, Dep. Comput. Sci., Univ. of Illinois, Urbana-Champaign, Rep. 76-837, Nov. 1976.

[2] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conf.*, vol. 32, 1968, pp. 307-315.

[3] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The ILLIAC IV computer," *IEEE Trans. Comput.*, vol. C-17, pp. 746-757, Aug. 1968.

[4] U. Banerjee, S. C. Chen, D. J. Kuck, and R. A. Towle, "Time and parallel processor bounds for Fortran-like loops," *IEEE Trans. Comput.*, vol. C-28, pp. 660-670, Sept. 1979.

[5] P. Budnik and D. J. Kuck, "The organization and use of parallel memories," *IEEE Trans. Comput.*, vol. C-20, pp. 1566-1569, Dec. 1971.

[6] S. C. Chen, D. J. Kuck, and A. H. Sameh, "Practical parallel band triangular system solvers," *ACM Trans. Math. Software*, vol. 4, pp. 270-277, Sept. 1978.

[7] S. C. Chen and D. J. Kuck, "Time and parallel processor bounds for linear recurrence systems," *IEEE Trans. Comput.*, vol. C-24, pp. 701-717, July 1975.

[8] "The CRAY-1 computer," *Preliminary Reference Manual*, Cray Res. Inc., Chippewa Falls, WI, 1975.

[9] P. H. Enslow, Ed., *Multiprocessors and Parallel Processing*. New York: Wiley-Interscience, 1974.

[10] P. M. Flanders, D. J. Hunt, S. F. Reddaway, and D. Parkinson, "Efficient high speed computing with the distributed array processor," in *High Speed Computer and Algorithm Organization*. New York: Academic, 1977, pp. 113-128.

[11] D. D. Gajski and L. P. Rubinfield, "Design of arithmetic elements for Burroughs scientific processor," in *Proc. 4th Symp. Comput. Arithmetic*, Santa Monica, CA, 1978, pp. 245-256; also in *Proc. 1978 LASL Workshop Vector and Parallel Processors*, Los Alamos, NM, 1978.

[12] R. G. Hintz and D. P. Tate, "Control data STAR-100 processor design," in *Proc. IEEE COMPCON 1972*, Sept. 1972, pp. 1-4.

[13] D. J. Kuck, "ILLIAC IV software and application programming," *IEEE Trans. Comput.*, vol. C-17, pp. 758-770, Aug. 1968.

[14] ——, "Parallel processing of ordinary programs," in *Advances in Computers*, vol. 15, M. Rubinoff and M. C. Yovits, Eds. New York: Academic, 1976, pp. 119-179.

[15] ——, "A survey of parallel machine organization and programming," *ACM Comput. Surveys*, vol. 9, pp. 29-59, Mar. 1977.

[16] ——, *The Structure of Computers and Computations*, Vol. I. New York: Wiley, 1978.

[17] D. J. Kuck, Y. Muraoka, and S. C. Chen, "On the number of operations simultaneously executable in Fortran-like programs and their resulting speed-up," *IEEE Trans. Comput.*, vol. C-21, pp. 1293-1310, Dec. 1972.

[18] D. H. Lawrie and C. R. Vora, "Multidimensional parallel access computer memory system," U.S. Patent No. 4,051,551, Sept. 27, 1977.

[19] ——, "The prime memory system for array access," *IEEE Trans. Comput.*, vol. C-31, this issue, pp. 435-442.

[20] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," *J. Ass. Comput. Mach.*, vol. 15, pp. 252-264, Apr. 1968.

[21] J. A. Rudolph, "A production implementation of an associative array processor—STARAN," in *Proc. 1972 AFIPS Fall Joint Comput. Conf.*, vol. 41, 1972, pp. 229-241.

[22] R. M. Russell, "The CRAY-1 computer system," *Commun. Ass. Comput. Mach.*, vol. 21, pp. 63-72, Jan. 1978.

[23] A. H. Sameh and R. P. Brent, "Solving triangular systems on a parallel computer," *SIAM J. Numer. Anal.*, vol. 14, pp. 1101-1113, Dec. 1977.

[24] F. H. Sumner, "MU5—An assessment of the design," in *Proc. IFIP Congress, Information Processing 1974*. Amsterdam, The Netherlands: North-Holland, 1974, pp. 133-136.

[25] D. W. Sweeney, "An analysis of floating-point addition," *IBM Syst. J.*, vol. 4, no. 1, pp. 31-42, 1965.

[26] J. F. Thorlin, "Code generation for PIE (Parallel Instruction Execution) computers," in *Proc. AFIPS Spring Joint Comput. Conf.*, 1967, pp. 641-643.

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

[28] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Res. Develop.*, vol. 11, pp. 25-33, Jan. 1967.

[29] W. J. Watson, "The TI ASC-A highly modular and flexible super computer architecture," in *Proc. 1972 AFIPS Fall Joint Comput. Conf.*, 1972, pp. 221-228.

David J. Kuck (S'59-M'69) was born in Muskegon, MI, on October 3, 1937. He received the B.S.E.E. degree from the University of Michigan, Ann Arbor, in 1959, and the M.S. and Ph.D. degrees from Northwestern University, Evanston, IL, in 1960 and 1963, respectively.

From 1963 to 1965 he was a Ford Postdoctoral Fellow and Assistant Professor of Electrical Engineering at the Massachusetts Institute of Technology, Cambridge. In 1965 he joined the Department of Computer Science, University of Illinois, Urbana, where he is now a Professor. Currently, his research interests are in the coherent design of hardware and software systems. This includes the development of the PARAFRASE system, a program transformation facility for array and multiprocessor machines. His recent computer systems research has included theoretical studies of upper bounds on computation time, empirical analysis of real programs, and the design of high-performance processing, switching, and memory systems for classes of computation ranging from numerical to nonnumerical. The latter work includes the study of interactive text processing and database systems, from the point of view of both the computer and the user.

Dr. Kuck has served as an Editor for a number of professional journals, including the IEEE TRANSACTIONS ON COMPUTERS, and is presently an area editor of the *Journal of the Association for Computing Machinery*. Among his publications are *The Structure of Computers and Computations*, vol. I. He has consulted with many computer manufacturers and users and is the founder and president of Kuck and Associates, Inc., an architecture and optimizing compiler company.

Richard A. Stokes was born in the Bronx, NY, on June 30, 1931. He received the A.B. degree in mathematics from St. Michael's College, Winooski, VT, and the B.S.E. degree in computer science from George Washington University, Washington, DC.

Following military service he joined the National Security Agency where he was involved in the early application of magnetic cores and transistor to computer design. In 1960 he joined the Martin Company as project leader in the development of CRT graphic displays. In 1964 he joined the Burroughs Corporation, Paoli, PA, as a Staff Engineer working on the design of the B 8501 computer. Later he became Lead Engineer in the development of the Illiac IV computer. In 1976 he was made General Manager of the plant responsible for the design and manufacture of the BSP. He holds several patents in computer design including the B 8501, the Illiac IV, and BSP.

Burroughs

B 5500
B 6500
B 7500

COMPILER ORIENTED HARDWARE

WHAT MAKES A COMPUTER SYSTEM **"COMPILER-ORIENTED"?** The BURROUGHS PUSH-DOWN STACK!

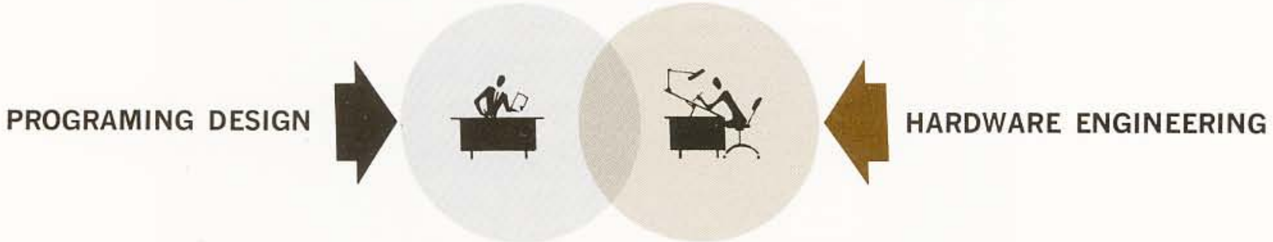
In the late 1950's, when the Burroughs B 5000 and its successor, the B 5500, were being designed, a major industry problem concerned man's ability, or lack of ability, to communicate his needs to the computer in a language that both he and the machine could understand. Compiler languages had been developed. But they produced inefficient translations into machine language. Lengthy compile times were common, and object programs, too, took longer to run. Most programmers summarily rejected the use of compilers because they wasted the resources of the computer.

Today, many competent programmers still reject compilers provided by other vendors because of their continuing inefficiencies.

DESIGNING A MACHINE YOU CAN TALK TO . . .

The Burroughs approach, from the beginning, has been to design computer system hardware that is compiler-oriented. The success of that effort is evident in the success of the Burroughs B 5500 . . . a computer that, for practical purposes, evolved without a need for assembly languages. And today, with the new B 6500 and B 7500 systems, Burroughs Corporation offers the only third generation systems based on the proven architecture of the B 5500 . . . the original compiler-oriented computer system.

The basis for successful compiler-oriented systems lies with Burroughs policy of cross-training its computer design teams. Hardware engineers were given proficiency in software architecture . . . and the programing engineers learned the intricacies of hardware design. And then they were merged into a single design team.



SHORTCUT TO EFFICIENCY—THE PUSH-DOWN STACK . . .

A technique called "Polish notation" was adopted by the team as the basic architecture for both the hardware and software. Polish notation simplifies mathematical expressions by eliminating the conventional rules of arithmetic precedence and "bracket-grouping" of values within an expression. Using Polish notation, the expression: (A+B)/C=E becomes: AB + C / E=

The rule that applies is: Follow two arithmetic values with the operation designated to work with those values. Thus, every mathematical operator automatically works on the most recently obtained pair of operands.

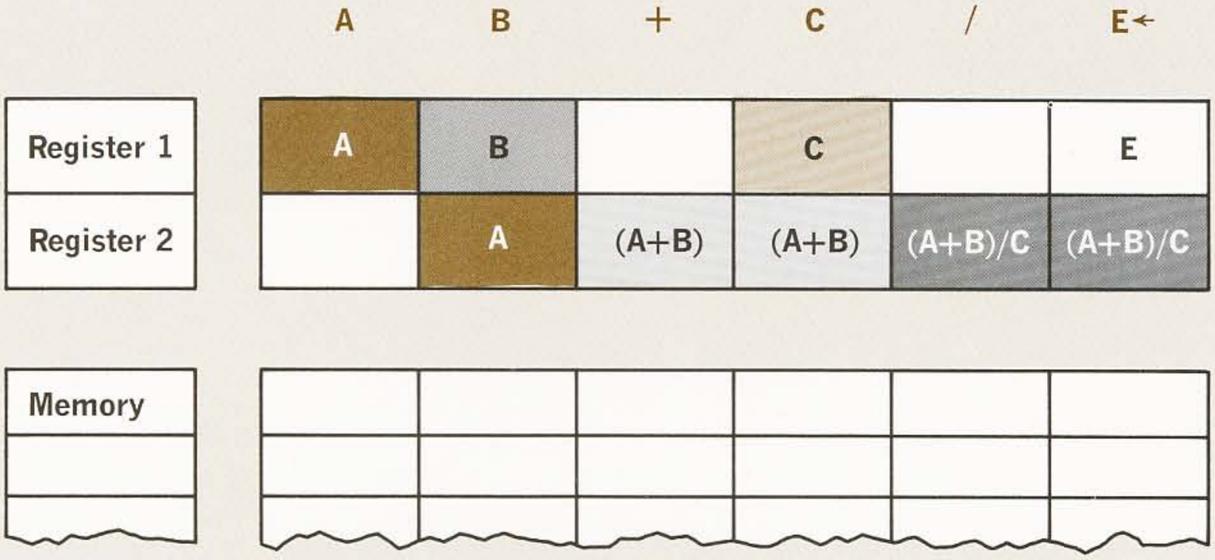
The Polish String Rule:

A B	Get Two Operand Values
+	Get the Appropriate Operator and automatically "Add"

High-level Polish notation is the ''machine language'' of these Burroughs systems. The power of Polish notation was fully applied to the hardware architecture of the B 5500 and is carried forward in the B 6500 and B 7500 systems. The push-down stack structure of these systems allows programs to be considered as strings of elements which correspond to values, literals, and operators in the compiler language.

The push-down stack, in hardware terms, consists of two or three registers (at the top of the stack) and a contiguous area of memory that permits the stack to extend beyond the registers. Working with a Polish string, the stack allows an arithmetic operator to work with whatever happens to be in the registers at a moment in time.

The ALGOL statement: E←(A+B)/C is expressed in Polish Notation as:



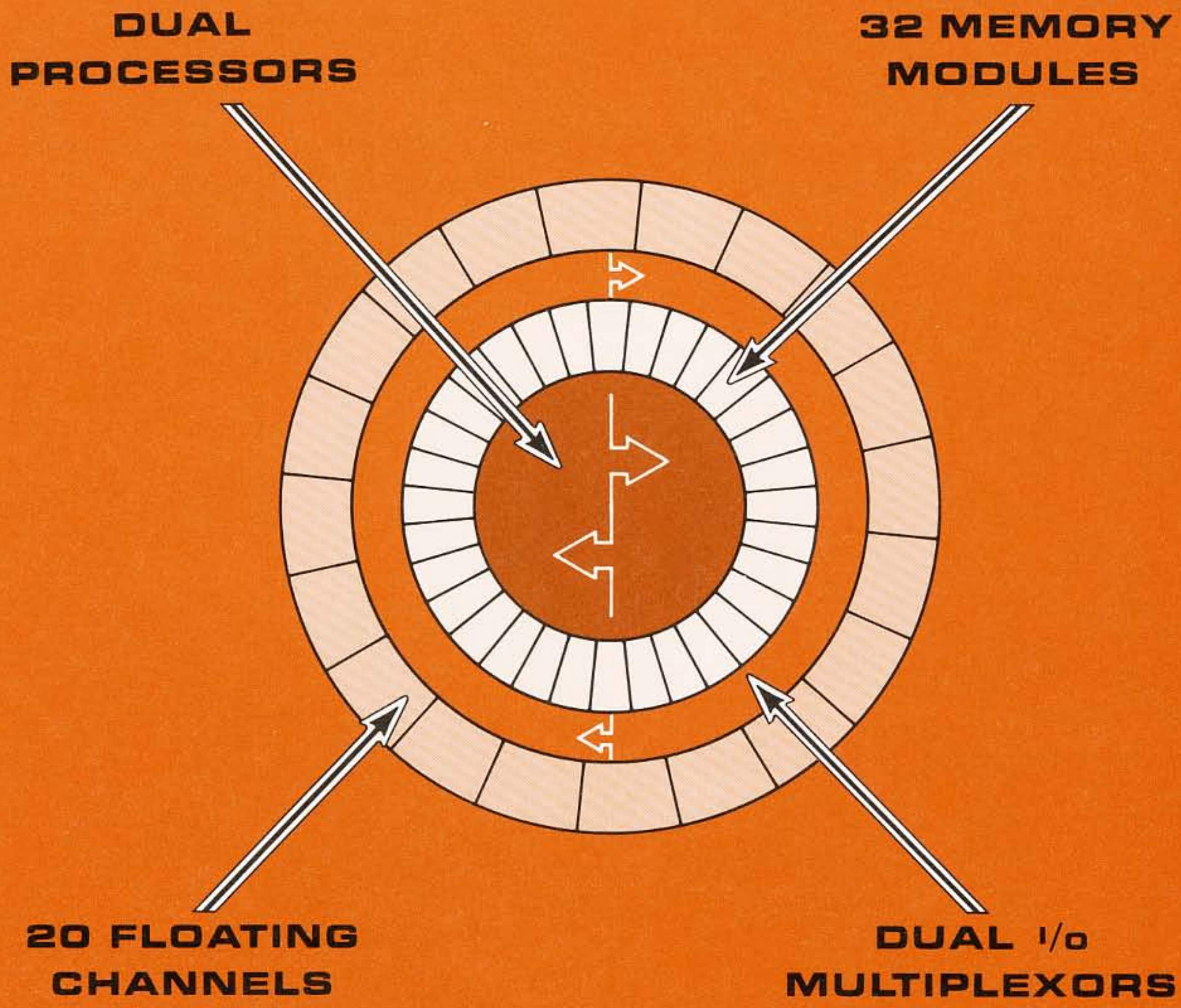
	A	B	+	C	/	E←
Register 1	A	B		C		E
Register 2		A	(A+B)	(A+B)	(A+B)/C	(A+B)/C
Memory						

In the example above, a memory cycle brings ''A'' into the top of the stack. The value ''B'' is next fetched, automatically pushing ''A'' down. Encountering the arithmetic operator for addition causes the contents of the two registers to be added automatically. The value ''C'' is fetched next . . . the proper division takes place with the topmost values in the stack . . . the memory location ''E'' is fetched . . . and the result of the expression is finally stored.

PROBLEM SOLVING FROM THE INSIDE OUT . . .

The logic of program compilation and execution has been greatly simplified by the architecture of these Burroughs systems. This is because the problems of man-machine communication have been handled by hardware design . . . and not by ''programing around the equipment.'' Burroughs has removed the need for storing and retrieving intermediate results of arithmetic expressions. And the machine language of the systems is compiler-oriented . . . removing the need for wasteful analysis of the program source language . . . speeding compilations . . . and producing highly efficient machine code.

Burroughs design philosophy is manifested in the B 5500, B 6500 and B 7500 systems . . . the most sophisticated, powerful, and useable systems in their class. Multiprocessing . . . multiprograming . . . priority scheduling . . . dynamic resource allocation . . . reentrant programing . . . variable memory allocation . . . and other important advances that the industry is talking about doing, have already been done by Burroughs. And scores of Burroughs users benefit from these proven advances every working day!



DUAL PROCESSORS

32 MEMORY MODULES

20 FLOATING CHANNELS

DUAL I/O MULTIPLEXORS

B 6500/7500

Wherever There's
Business There's / **Burroughs**

Burroughs

B 5500
B 6500
B 7500

REENTRANT PROGRAMING

REENTRANT PROGRAMING...

ANOTHER ADVANCED BURROUGHS Computer Concept...Working Today!

Six years ago Burroughs Corporation began marketing an advanced multiprocessing system. This was an unequalled accomplishment at the time, and Burroughs leadership in this area still stands. Multiprocessing, multiprograming operating systems are complex and difficult to develop. Yet Burroughs faced and solved the developmental problems years ago as the Master Control Program (MCP) was being debugged.

A NEW TECHNIQUE...

Since then, Burroughs has gone beyond the conventional concepts of multiprocessing. Today, it has again increased the efficiency of its multiprocessing systems through a new technique called reentrant programing.

> As incorporated in the B 5500, B 6500 and B 7500 computers, reentrant programing allows many people to use the same single copy of a program in memory . . . at the same time.

Figure 1 illustrates the memory allocation for a variety of typical jobs in a mix. Conventionally, a separate copy of the application program or compiler is required for each person using the system. Now, with the new Burroughs systems, each programmer uses the same copy of a compiler that everyone else is using . . . and the same is true for any program in the system, as indicated in Figure 2.

COBOL user	PAYROLL user
COBOL user	PAYROLL user
COBOL user	FORTRAN user

Fig. 1—Conventional Multiprocessing
Six concurrent users demand
all of available memory.

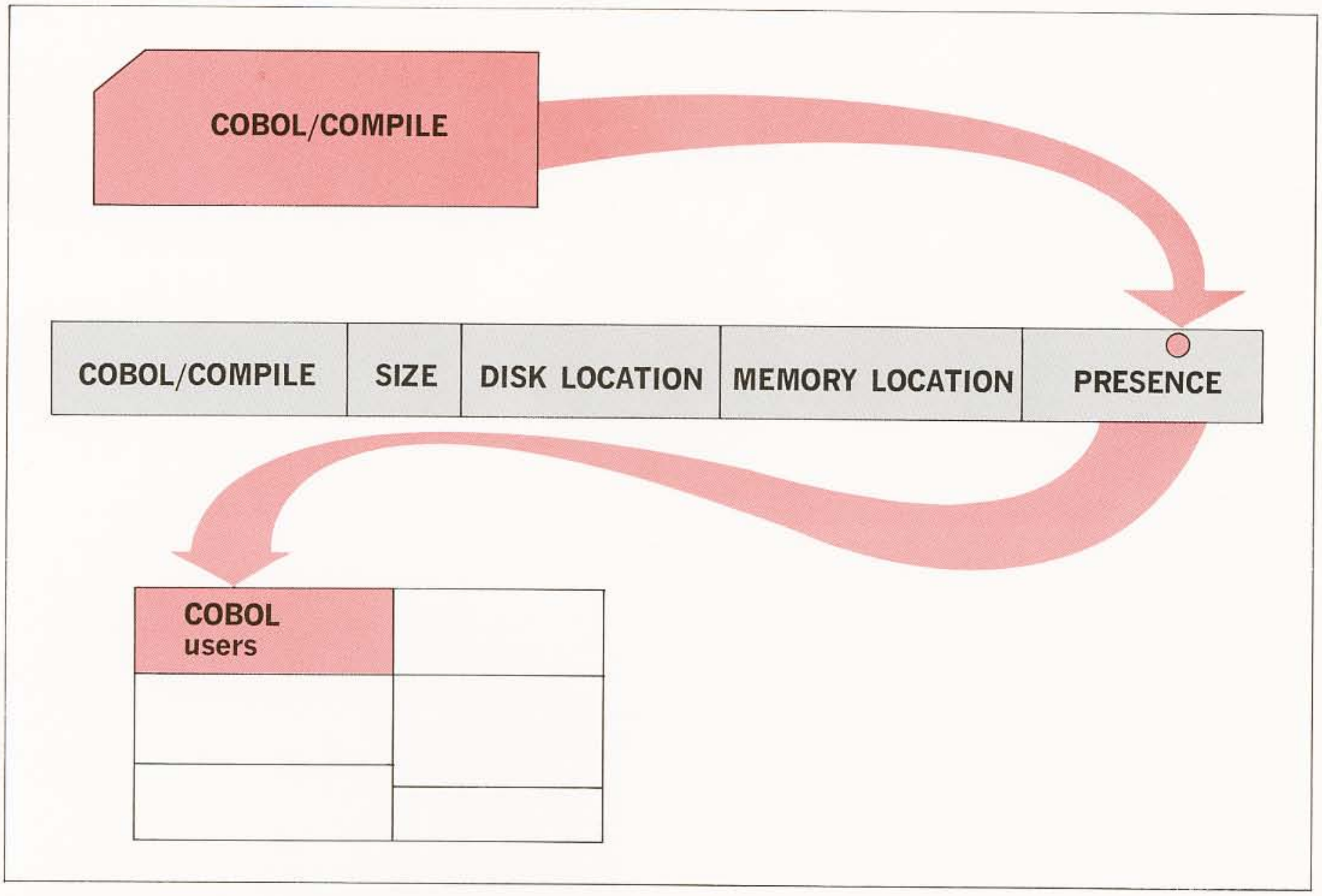
COBOL users 1, 2, 3	FORTRAN users 1, 2, 3, 4, 5
Data Communication Users 1—30	Inventory users 1, 2
PAYROLL users 1, 2, 3, 4	Analysis users 1

Fig. 2—Burroughs Reentrant Multiprocessing
Dozens of users have access to more
programs in the same or even smaller
amount of memory.

Burroughs has also abolished the job of writing programs to "fit in memory." Automatic segmentation handles it now. So, programing can be accomplished without regard for memory size!

Reentrant user programs and compilers save memory. They let the Burroughs user put more productive work into his system at a time. They let more people have access to the system's resources at a time. And they eliminate the need for retrieving and initiating programs every time a new user demands access. This cuts overhead time drastically.

The Burroughs user can access his system instantly and efficiently. The Master Control Program operating system checks its dictionary of programs currently resident in the system. If a single ''presence'' bit is ON for the program needed, the user can begin running his program immediately.



COMBINED HARDWARE/SOFTWARE DESIGN TEAM . . .

Reetrant programing development involved many difficult problems. But Burroughs was able to solve them by thoroughly defining software requirements before building hardware. A significant part of this solution involved the cross-training of Burroughs engineering and programing teams and merging them into a single integrated design team.

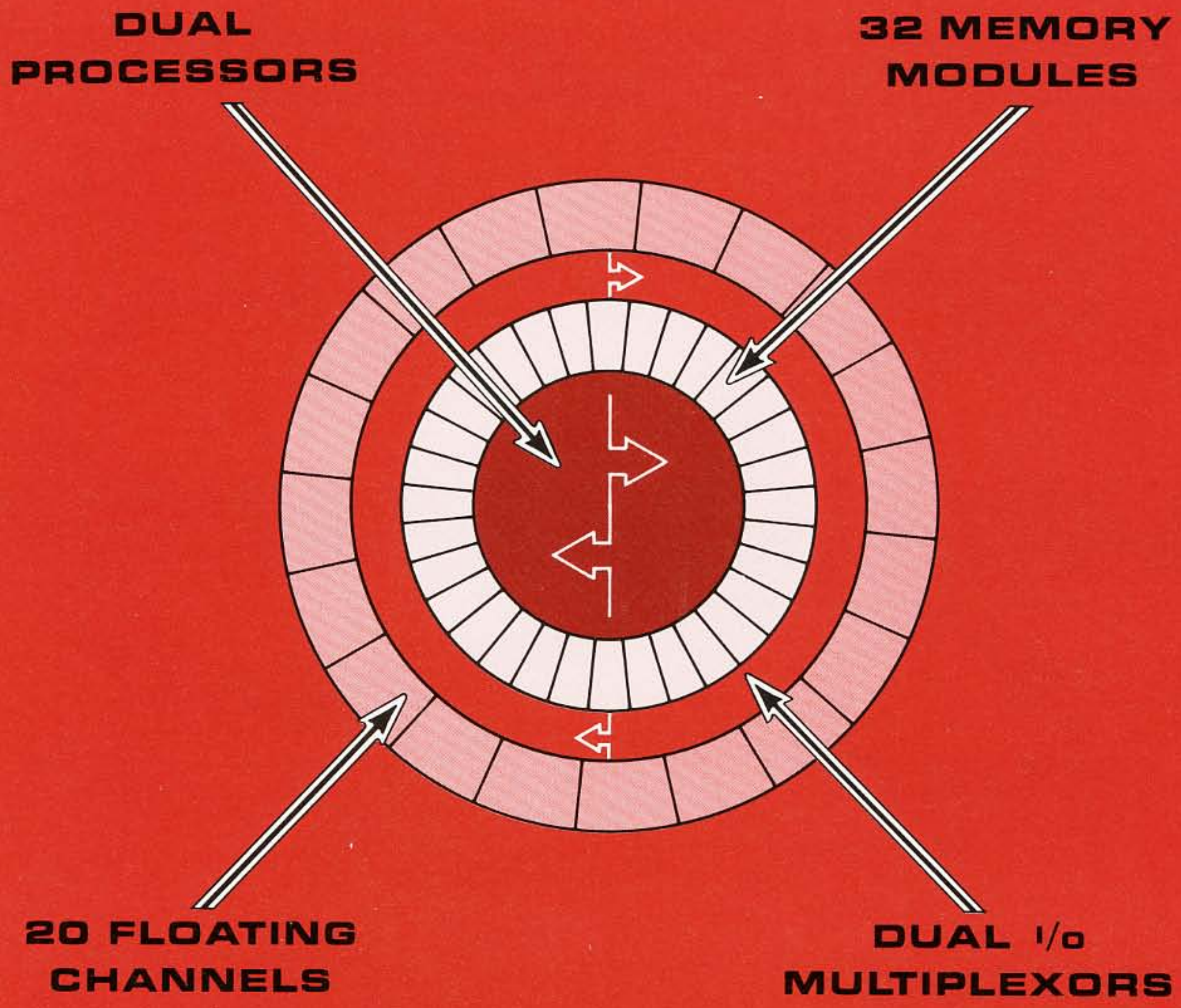
The result of this integrated team effort was the B 5500. Its new companion third generation systems, the B 6500, B 7500, and B 8500 offer the same proven base of successful multiprocessing experience.

FAR MORE SYSTEM POWER—AND 'PEOPLE' POWER . . .

Burroughs reentrant programing brings unique power to scientific and business problem-solving alike. It's a highly important technique for a data communications environment . . . for multiple access accounting . . . for management information . . . for real time applications . . . for new application development and testing.

The Burroughs B 6500 and B 7500 systems can give their users access to more than 3,000,000 bytes of main memory . . . head-per-track disk storage up to 19 billion bytes . . . hundreds of tape drives . . . more than 2,000 independently-functioning data communications lines . . . and more. They can process scientific work, management information, and data communications faster than you would have thought possible. After all, multiprocessing is their normal mode of operation . . . with 20, 30, 40 or more jobs running concurrently.

The B 5500, B 6500, B 7500 and B 8500 are uniquely useable systems, and Burroughs understanding of what computer users want and need will continue to underlie our systems philosophy.



B 6500/7500

Wherever There's Business There's Burroughs

1031770 4-68 Printed in U. S. America

Burroughs

B 5500
B 6500
B 7500

AUTOMATIC PROGRAM SEGMENTATION

Automatic Program Segmentation...

The BURROUGHS method for maximum memory utilization.

Can a multiprocessing computer system use **all** of its memory **all** of the time?

Can a computer handle programs that **exceed** its memory capacity?

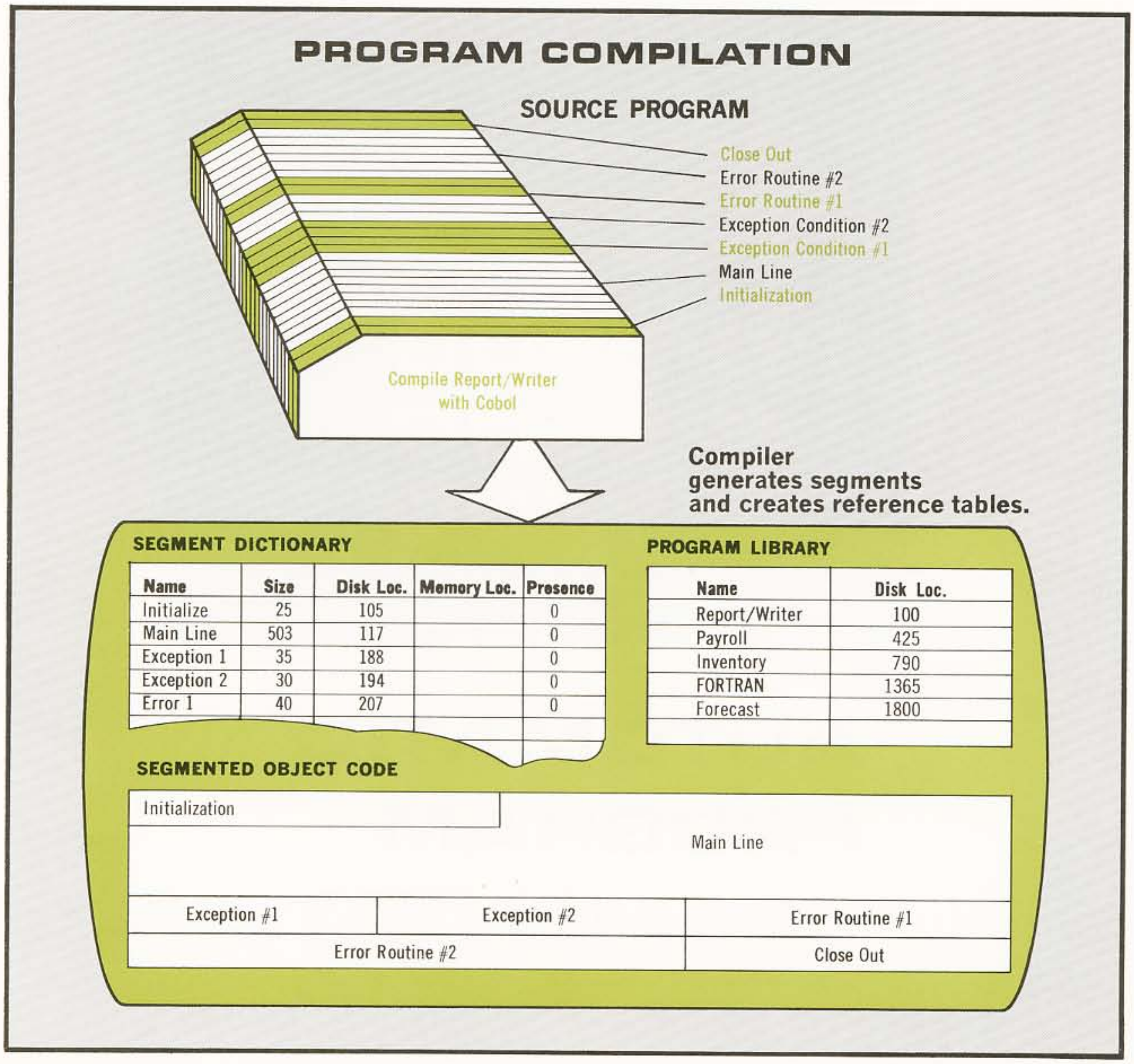
The answer is "yes." Burroughs Corporation, through Automatic Program Segmentation, has a practical answer that works today.

Burroughs Automatic Program Segmentation is based on the use of compilers as program preprocessors. These Burroughs compilers logically segment all programs at compilation time and create a detailed record of how the segmenting is performed.

Segmentation . . . Basic to Burroughs Software Design

Compiler languages naturally lend themselves to logical and direct segmentation. They were designed that way. COBOL programs are written in paragraphs, ALGOL programs in blocks, and FORTRAN programs in subroutines.

Users of B 5500, B 6500, and B 7500 computers write their programs in these higher level languages exclusively. The Master Control Program executive system (MCP) which controls each of these computers recognizes that a compiled program has already been segmented and that a segment dictionary has been prepared for it.



SEGMENT DICTIONARY

Name	Size	Disk Loc.	Memory Loc.	Presence
Initialize	25	105		0
Main Line	503	117		0
Exception 1	35	188		0
Exception 2	30	194		0
Error 1	40	207		0

PROGRAM LIBRARY

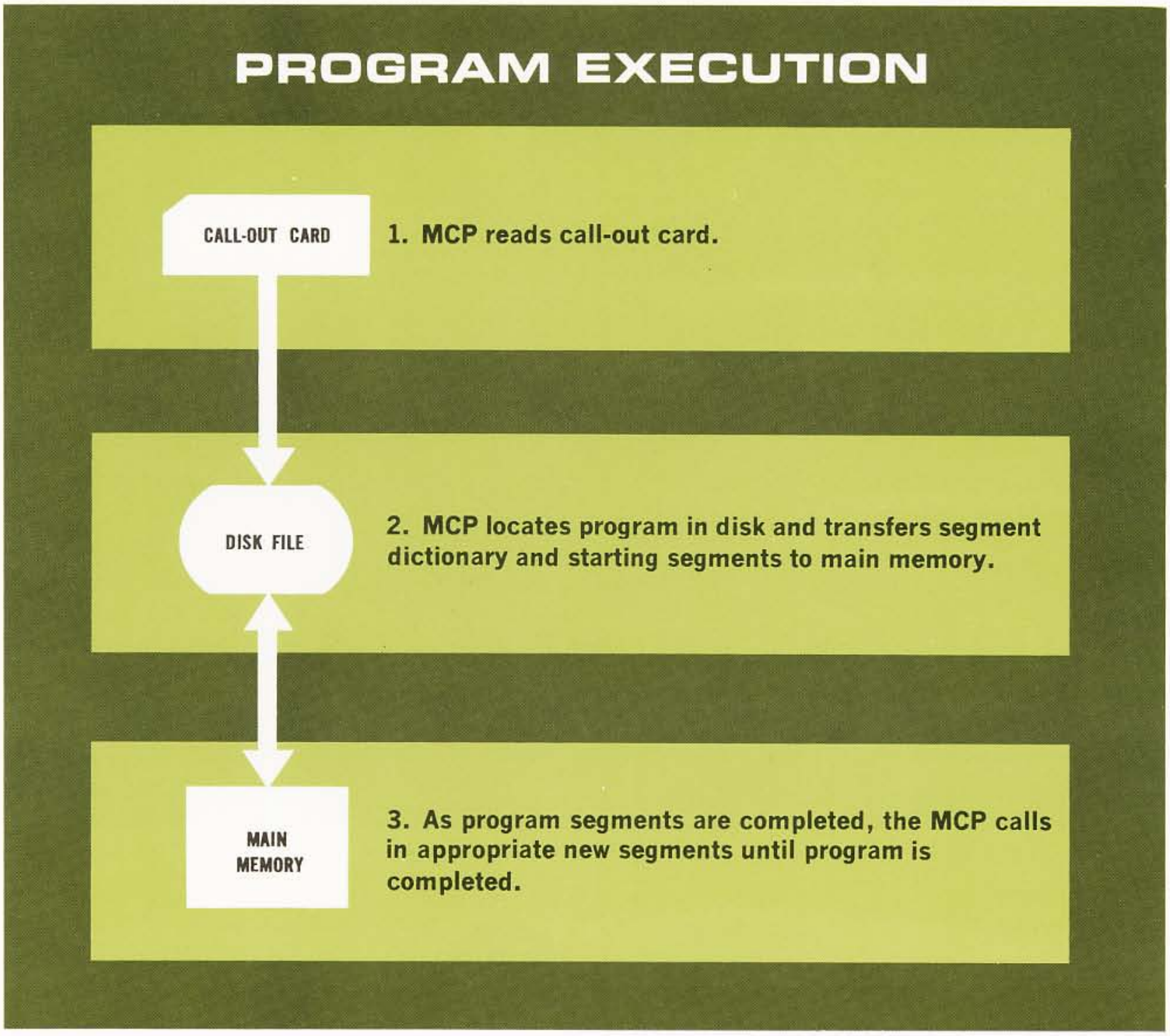
Name	Disk Loc.
Report/Writer	100
Payroll	425
Inventory	790
FORTRAN	1365
Forecast	1800

Segmentation in Action

On command, the MCP uses the program directory to determine where the required program and its segment dictionary are located in the program library. The MCP then fetches the segment dictionary in to main memory and enters the program into the current job mix.

Using the segment dictionary, the MCP brings program segments into main memory as needed and performs the overlaying operation. To insure the most efficient use of main memory space, the MCP keeps a running record of: program segments currently in memory, their location; available memory; and those areas of memory that can be safely overlayed. As soon as a program segment completes its operations, the MCP assigns another segment or program to the newly available memory location.

This automatic segmentation means that memory requirements result from segment size rather than program length. It assures that memory is not wasted or left idle. It provides memory protection by using segment length and location as protective barriers. The Burroughs Master Control Program takes care of all this automatically, and at a fraction of the cost in resident memory space and execution time of the best competitive executive systems now available.



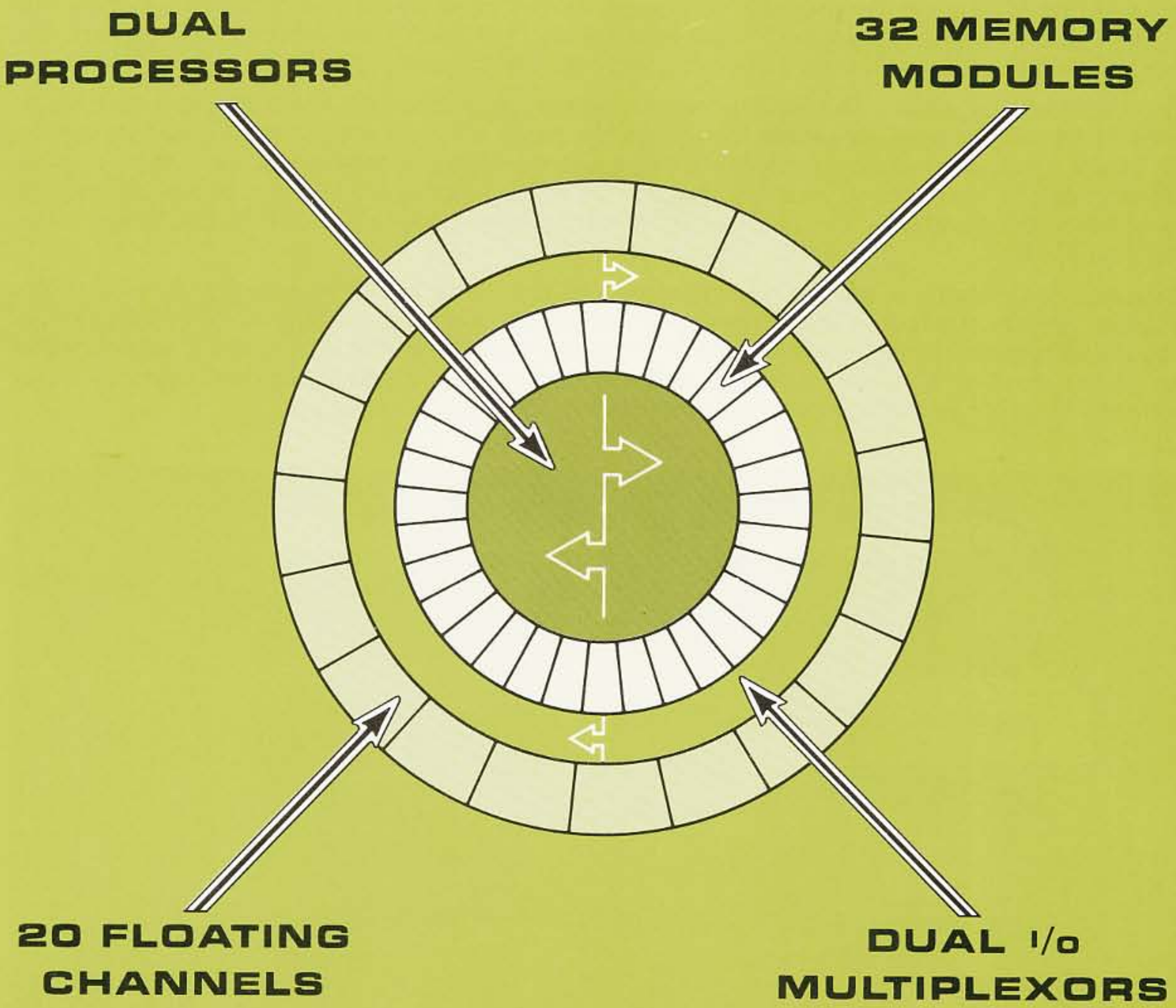
1. MCP reads call-out card.

2. MCP locates program in disk and transfers segment dictionary and starting segments to main memory.

3. As program segments are completed, the MCP calls in appropriate new segments until program is completed.

Putting Concepts to Work

Burroughs customers **use** Automatic Program Segmentation **today**, in normal operations, to run programs of any size, at any time, without concern for the other jobs run concurrently. They are also **using** Multiprograming and Parallel Processing, Priority Scheduling, Reentrant Programing, Dynamic Resource Allocation, and other advanced concepts pioneered and proven by Burroughs. Burroughs can put them to work for you . . . right now.



B 6500/7500

Wherever There's
Business There's / **Burroughs**

1032901 4-68 Printed in U. S. America

Burroughs

B 5500
B 6500
B 7500

THIRD GENERATION SOFTWARE EVALUATION

THE A-B-C's OF SOFTWARE EVALUATION . . . Are You Asking the RIGHT Questions?

Just a few years ago, there were some standard questions to ask in evaluating a new computer. For example, did the new machine offer:

- more memory?
- faster access time?
- lower price?

Of course, the answers lost their meaning as compilers, operating systems, and utilities came into use. Because system software could degrade hardware performance, a second evaluation technique, "kernel timing", was developed. "Kernel timing" is a method of hand timing the instructions that make up a representative set of jobs. And it works pretty well as a basis of comparison, provided that the job set selected is truly representative and the jobs are to be processed serially, as they were when computers first came into commercial use.

But in many instances serial processing wastes a major portion of the third-generation computer's potential. More and more users are demanding multiprocessing capability, either through an automatic interleaving of program operations (multiprograming), or through simultaneous execution of two or more programs or program segments (parallel processing). Multiprocessing involves "overhead"—time the computer devotes to controlling its own operations. "Kernel timings" cannot begin to measure the effects of "overhead" on a multiplicity of jobs contending for and sharing the resources of a computer.

So, it's time to ask some new and definitive questions about computer performance. How **can** you evaluate multiprocessing systems, and where do you start?

One logical place to start is with the manufacturer's background and performance record. That, of course, brings you right to Burroughs Corporation!

Burroughs has been demonstrating and installing multiprocessing computers since 1963. By that time Burroughs had implemented the first really successful automatic operating system for computers—the Master Control Program (MCP)—and had solved most of the operating system problems that the rest of the industry is struggling with today. Since then we have been refining our already successful MCP and adding to its capabilities.

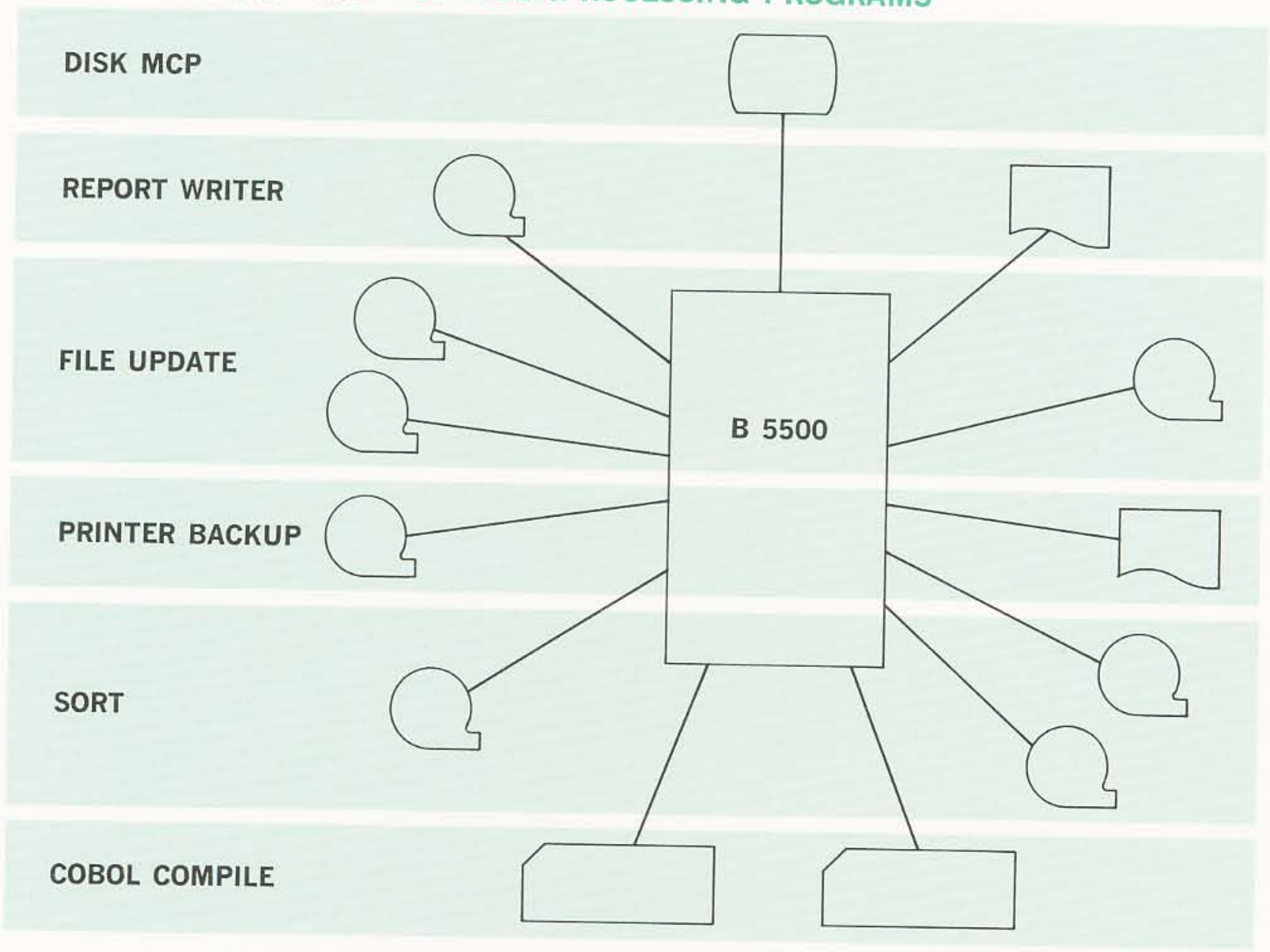
But see for yourself! The true value of the MCP operating system . . . of variable priorities . . . of reentrant programing . . . of dynamic modularity . . . of compiler-oriented system design . . . and other exclusive features will become dramatically clear when you **see Burroughs third generation software running.**

See Burroughs multiprocessing demonstrated, and then . . .

ASK YOURSELF THESE KEY QUESTIONS:

- Were you invited to participate in the demonstration? . . . to change some of the parameters? . . . to prove to yourself that it was a "live" demonstration using real situations?
- Was communication between the operator and the system simple and easy to understand?
- Was there true device independence? What happened when peripheral devices were switched off? Did the operating system recognize the fact and reallocate the workload to other units?
- Was the Job Control Language simple and easy-to-use or was it complex and error-prone?
- Did you actually see multiple jobs being run through the system?
- Were you able to find out how much processor time each job used?
- Did you see "dynamic memory allocation"? Was additional memory allocated to long jobs as soon as the shorter jobs were finished? Did the longer jobs begin to run faster automatically?

REPRESENTATIVE "MIX" OF MULTIPROCESSING PROGRAMS



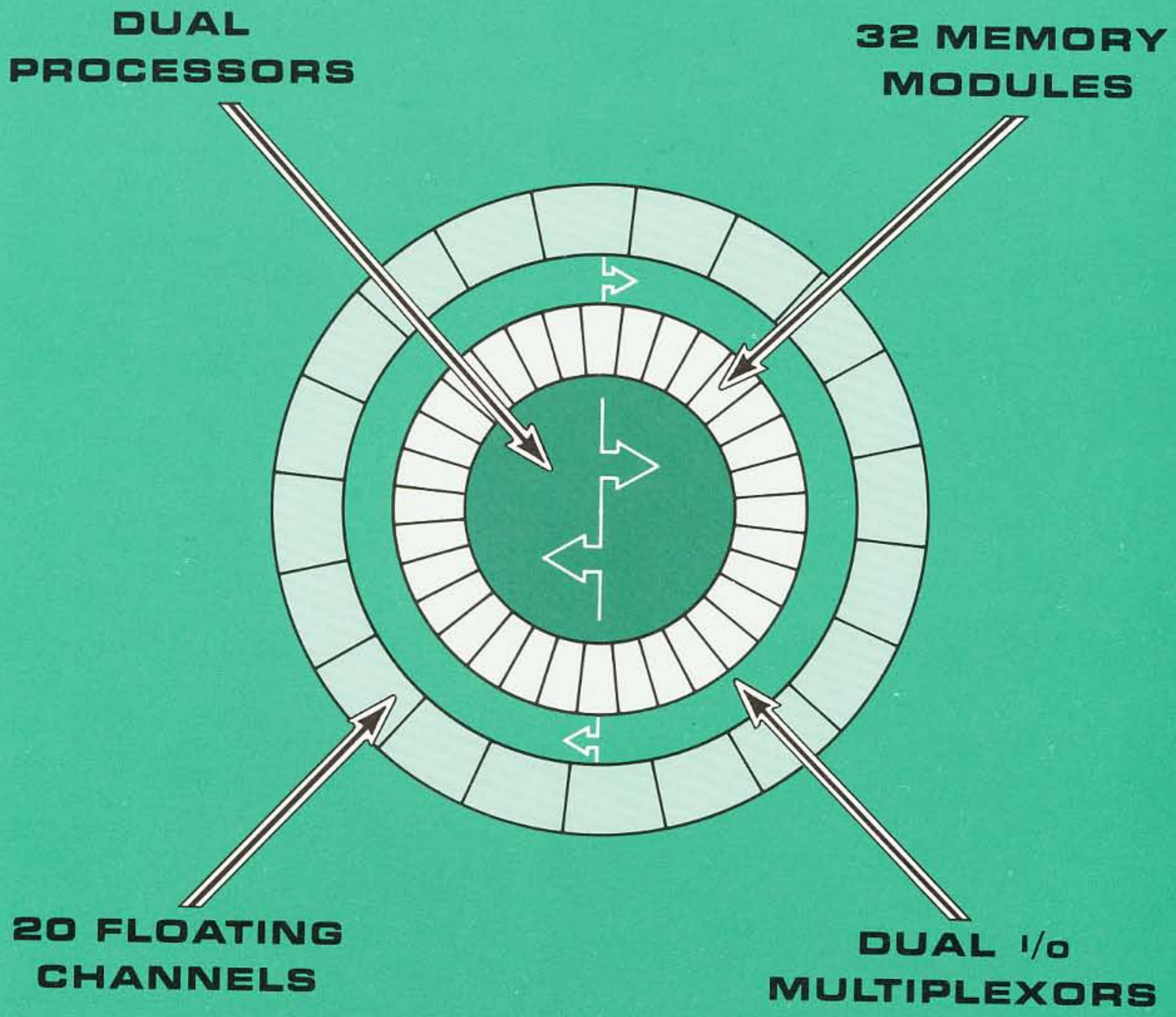
- Did "dynamic expansion" allow new devices to be added to the system and put to immediate use, automatically?
- Were jobs initiated from a remote terminal device such as a teletype?
- Were data communications jobs run concurrently with background multiprocessing?

SEEING IS BELIEVING

When you have seen a demonstration of these and other features, and have satisfied yourself that Burroughs systems answers these questions positively, you may still ask: Is multiprocessing practical?

Consider this: Multiprocessing is a standard operating mode for Burroughs 500 Systems users across the nation. Our customers have not been required to install hundreds of thousands of extra memory positions, or extra tape drives, or extra disk or drum memories; the resident nucleus of the Burroughs B 5500 MCP occupies as little as 4,000 words of main memory! Nor do these systems build up excessive, non-productive overhead; instead, better than 95 percent of their time goes to productive work!

Is multiprocessing practical with Burroughs third generation operating software? Ask some of our customers who have been multiprocessing since 1963 . . . they know it's practical!



B 6500/7500

Wherever There's
Business There's Burroughs

12-67

Printed in U. S. America

Burroughs

B 5500
B 6500
B 7500

FLOATING CHANNELS

Burroughs FLOATING CHANNELS
THE MOST EFFECTIVE INPUT/OUTPUT SYSTEM IN THE INDUSTRY

The channel concept is not new in the computer industry. Years ago, people began to realize that much of a large computer's power would be lost without some efficient method of executing simultaneous input/output operations. For a long time, the answer seemed to be to have a small peripheral computer dedicated to processing input/output data for the larger system. Buffering slow speed devices also provided a partial answer.

As technology advanced, it became clear that independent access paths could be built into the computer. The access paths, called "channels", can be considered as traffic routes which allow two or more data streams to flow simultaneously into and out of the computer.

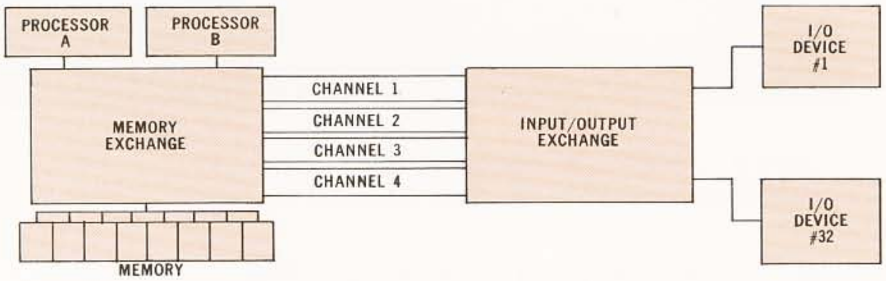
Industry's approach to keeping channel capacities on a par with faster and faster computers has been to increase the number of channels and to make them faster. But this has not provided a completely adequate answer. With most systems, a device may be kept waiting for a specific channel until I/O traffic for that channel is completed, even though other channels are free.

Some computer manufacturers have attempted to solve this problem by allowing the programmer to search for an available channel if the normally assigned channel is found to be busy. This is an attempt to overcome the contention for channels which typically exists in those systems having many devices *physically attached to fixed channels.* This technique imposes a heavy burden on the programmer. In fact, the job becomes nearly impossible when the programmer is forced to consider not only his own task, but also several other jobs running in a multiprograming or parallel processing mode.

The only clear answer is to make the channels themselves able to respond to a request for access by automatically switching the access to the first free channel path. Burroughs Corporation has taken this "floating channel" approach as part of the B 5500, B 6500, and B 7500 design philosophy.

THE B 5500 CHANNELS . . . YEARS AHEAD OF THE INDUSTRY

In the original B 5500 design, Burroughs engineers and software specialists recognized that a true multiprocessing system would have to be capable of making I/O channel assignments itself, on an unrestricted, dynamic basis. The resulting design efforts produced a type of system architecture which, today, remains the most advanced in the industry.



Working through an input/output exchange and a memory exchange, any B 5500 I/O channel is able to answer the demand of any I/O device at any time. No device is tied to a specific channel. The programmer is free of any consideration for or attention to channel optimization. This system automatically optimizes use of the I/O channels.

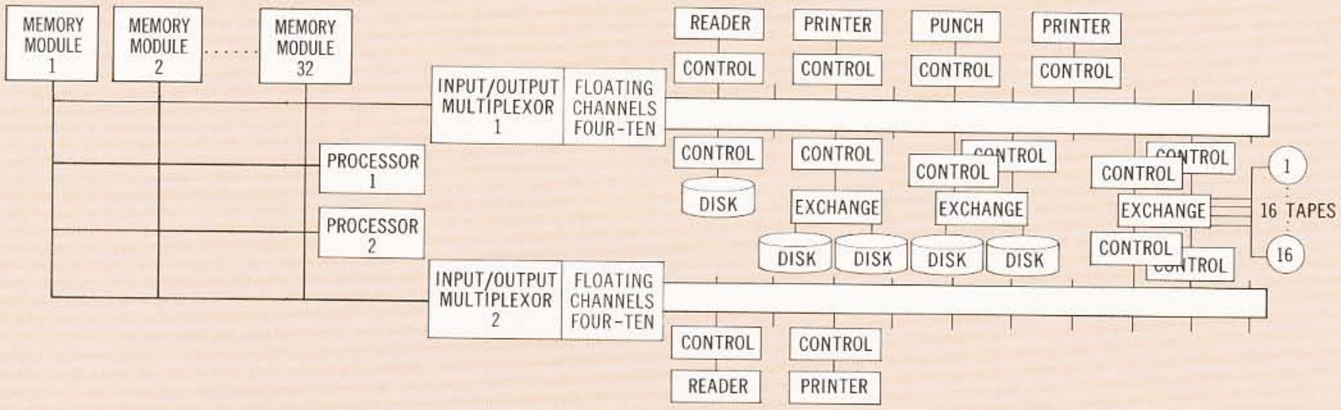
The efficiency of its floating channels, combined with other years-ahead features such as multiprocessing and full operating system control, have allowed the B 5500 to compete with systems costing twice as much.

THE B 6500 AND B 7500 . . . A NEW GENERATION OF INPUT/OUTPUT CONTROL

The concept of automatic floating channels was proved with the B 5500. Now, with the B 6500 and B 7500 this capability is advanced another order of magnitude.

One or two Input/Output Multiplexors may be attached to the B 6500 and B 7500 systems. Each of the multiplexors may control up to 10 floating channels. This means that on a maximum system, 20 input/output operations may take place simultaneously.

The input/output subsystem can be structured to provide automatic back-up as well as multiple paths to clusters of devices. Each channel can be switched, under control of the computer's automatic operating system, to answer the peripheral demands of the moment.



Hundreds of magnetic tape drives can be attached to the B 6500/B 7500 . . . up to 19 billion bytes of on-line disk file storage . . . input/output devices in a myriad of configurations. And up to 20 of these devices can be accessed simultaneously by continually changing, continually optimized access paths.

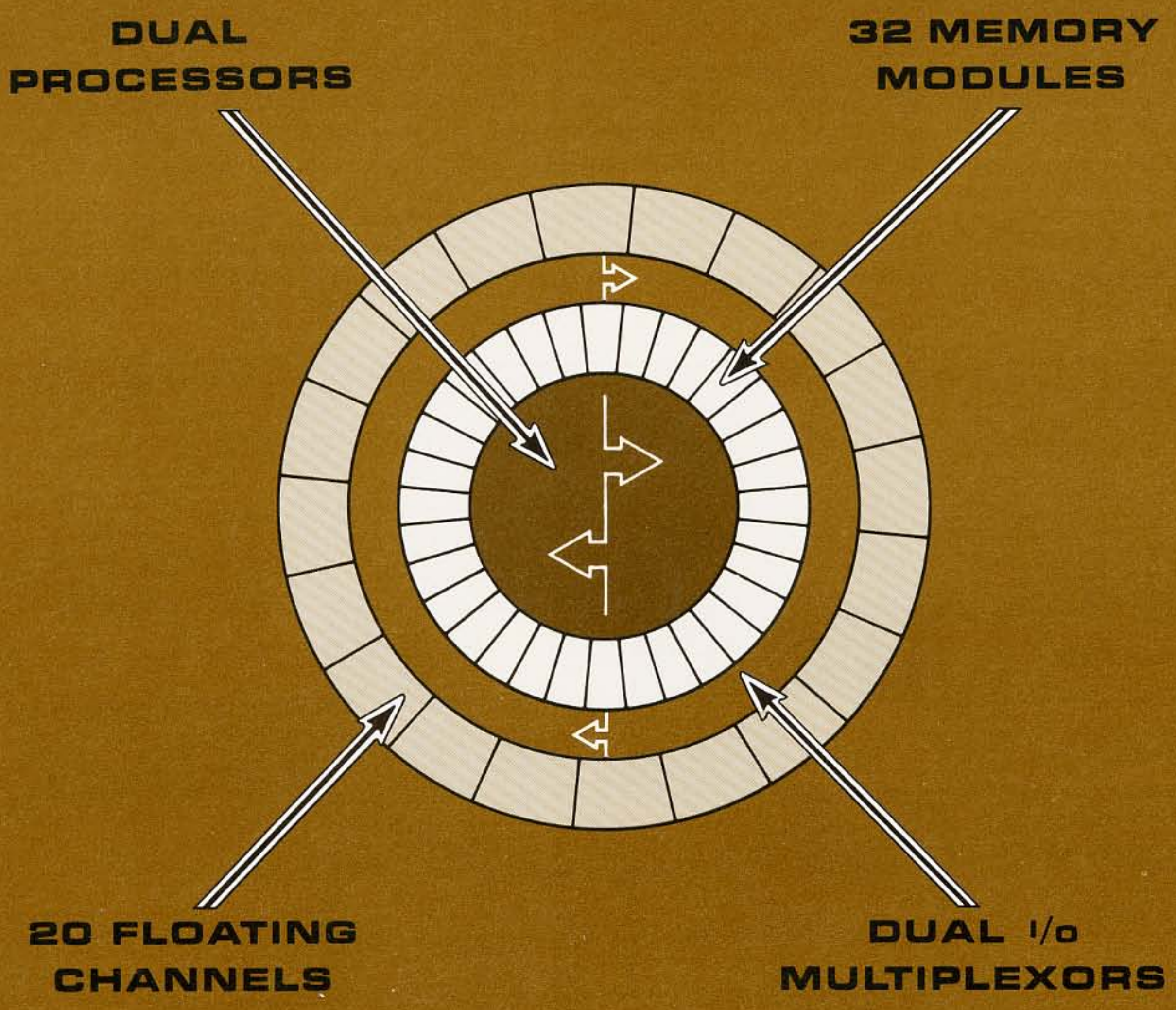
EXPANDED CONTROL OF COMMUNICATIONS AND REAL TIME APPLICATIONS

In addition to providing the most powerful and sophisticated I/O processing in the industry, the Burroughs Input/Output Multiplexors vastly extend the system's ability to handle communications and real time jobs. Up to four Data Communications Processors can be attached to each I/O Multiplexor. These DCP's are small computers designed specifically to handle communications line discipline and management for up to 256 independent lines. This means that a maximum configuration will serve 2048 communications lines without interfering with the rest of the system.

For real-time data acquisition, each multiplexor can accommodate a Real-Time Adapter for communication with process control devices and instrumentation. And each real time device has a direct, separate path to the processor(s) so that interrupts can be processed without accessing memory.

BURROUGHS 500 SYSTEMS . . . THIRD GENERATION REALITIES

With Burroughs multiprocessing systems, third generation concepts are working, practical realities. Burroughs "floating channels" were years ahead of the industry when they were introduced in the early 1960's . . . and today, working with the Burroughs Master Control Program operating system, they still provide "years ahead" benefits.



B 6500/7500

Wherever There's Business There's **Burroughs**

1031788 12-67 Printed in U. S. America

Burroughs B 5500 INFORMATION PROCESSING SYSTEM

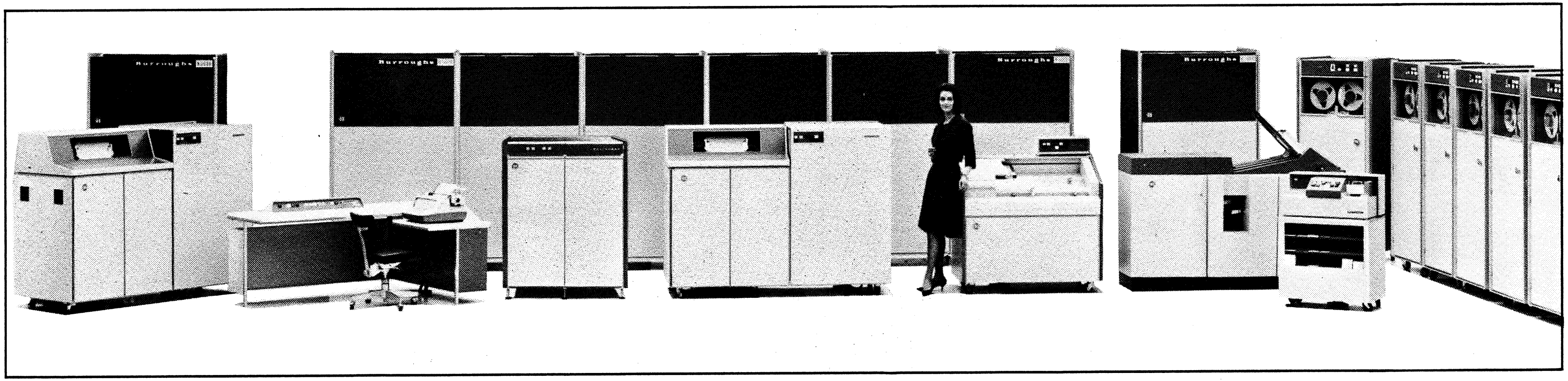
ADVANCED CONCEPTS - PROVEN IN USE

Burroughs B 5500 INFORMATION PROCESSING SYSTEM

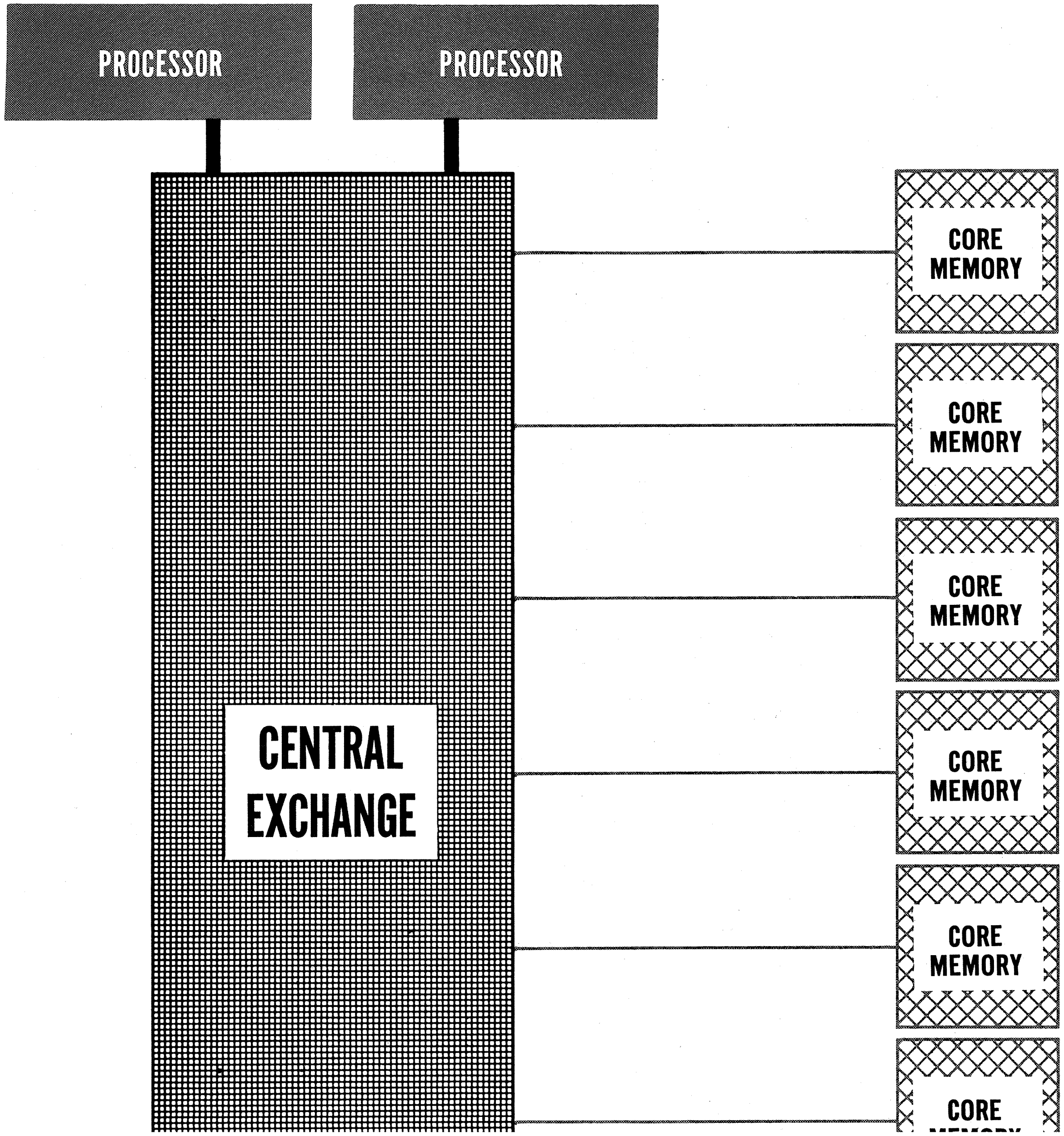
The Burroughs B 5500 is a highly advanced information processing system that spans the medium, intermediate and large scale ranges of computer equipment. The design concepts and software of the B 5500 have been thoroughly tested in over 50 installations of its predecessor systems—the commercial Burroughs B 5000 and military Burroughs D 800 Series modular data processing systems.

These earlier Burroughs computer systems have accomplished, in thousands of hours of customer use:

- automatic control through a full-scale operating system;
- multiprocessing of independent programs, with dynamic scheduling by the operating system;
- modularity, for expansion without reprograming;
- efficient, effective use of problem oriented languages.



ORGANIZATION OF THE Burroughs B 5500



PROCESSORS

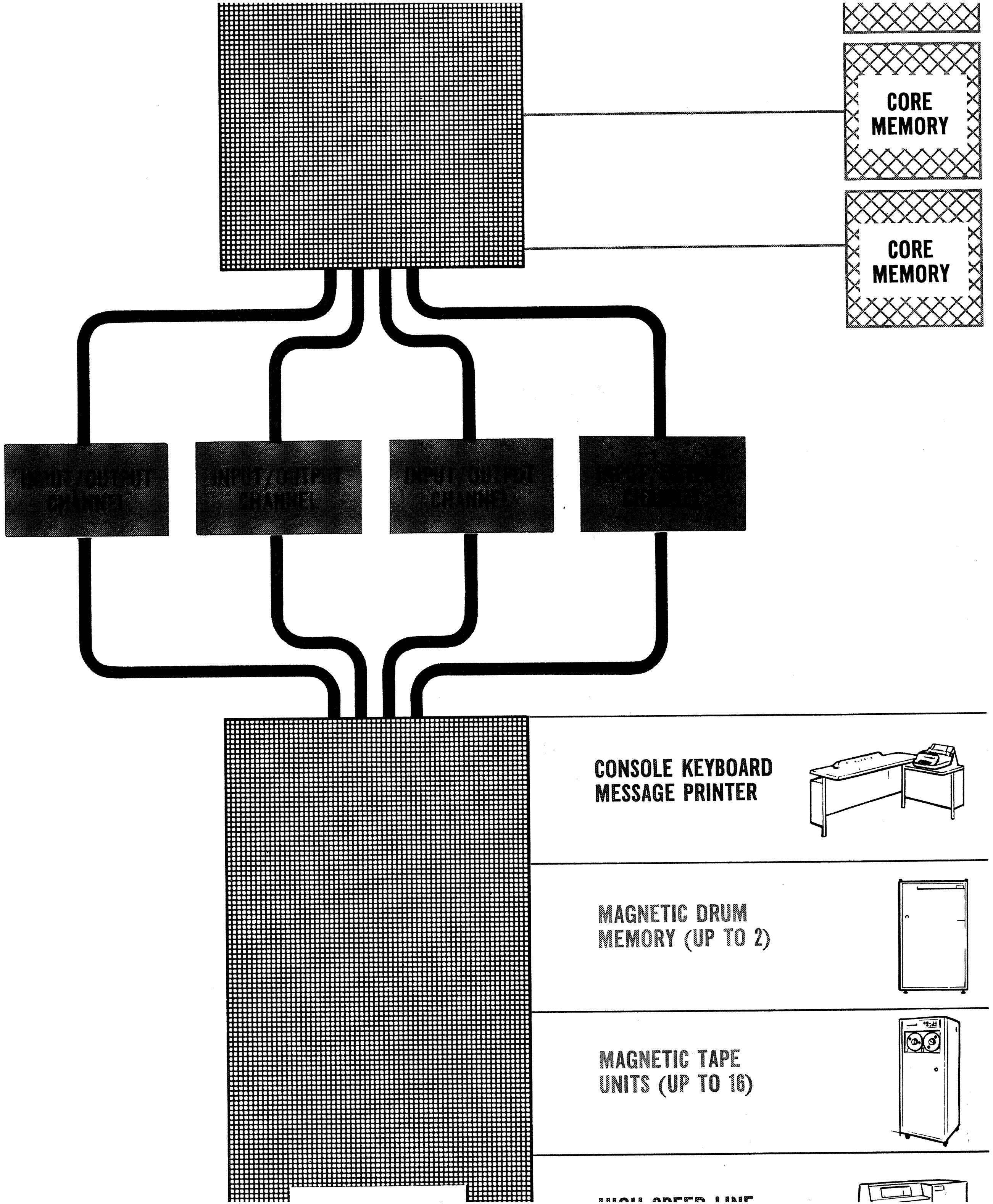
One or two parallel, independent, solid state processors. All transfers full word parallel at one megacycle. Word and/or character operation. Fixed/floating point. Word: 49 bits including 1 parity bit, interpreted binary or alphanumeric. Instruction format: 12-bit syllables packed four per word. Unlimited indexing. Powerful, comprehensive interrupt system.

MEMORY

Magnetic core: One to eight modules of 4,096 words each. Memory modules are functionally independent, permitting simultaneous access through the Central Exchange by processors and I/O channels. Read access time: 2 usec/word, 250 nanoseconds/character.

Magnetic drum: One or two drums, of 32,768 words capacity each. Average access time: 8.5 milliseconds.

Disk file: One to 100 modules, 9.6 million alphanumeric characters per module. Average access time: 20 milliseconds. Read/write head for each track. 100KC transfer rate.

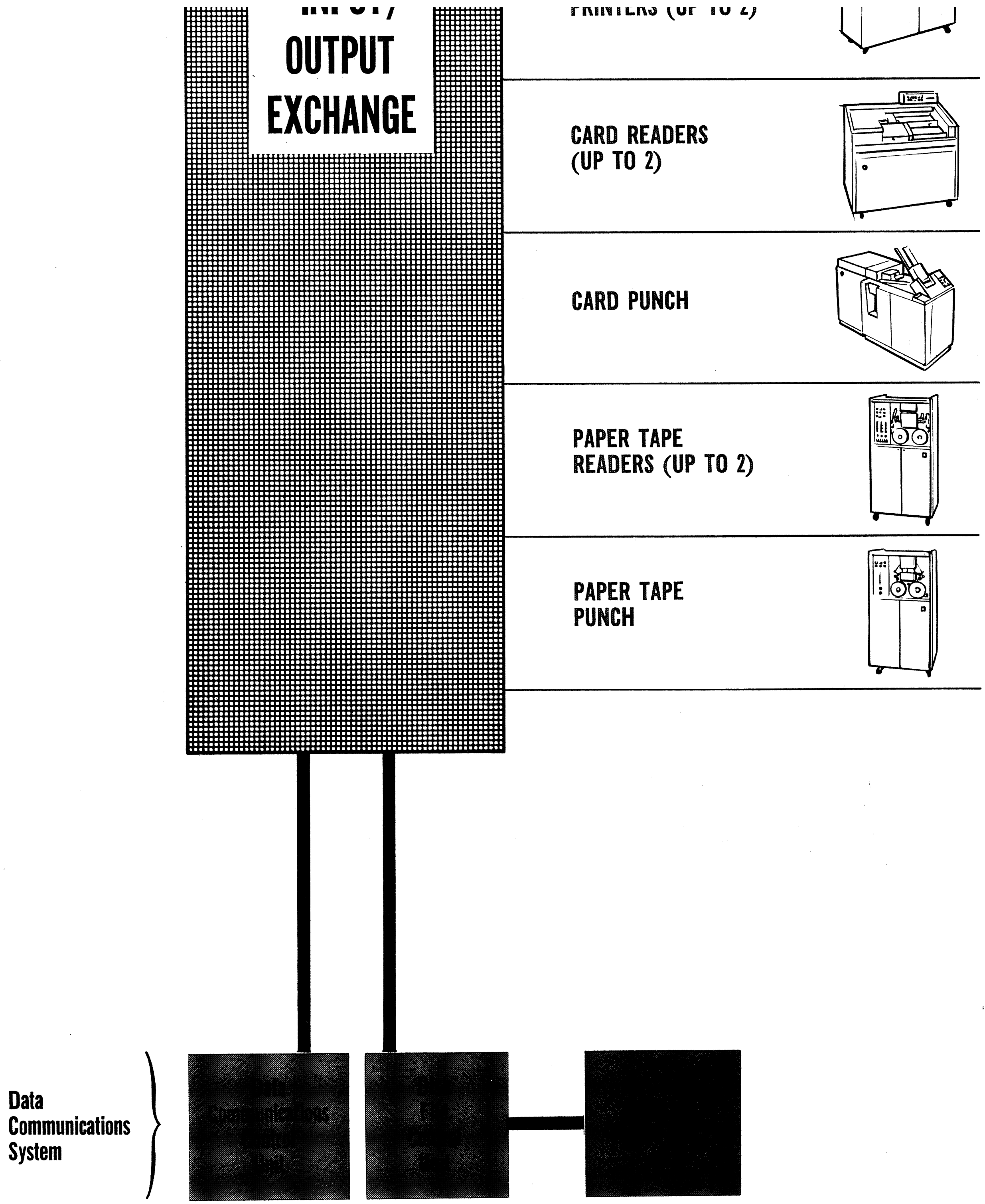


I/O CHANNELS

One to four independent input/output channels. Any channel may establish communication between any area of core memory and any I/O device. Up to four I/O operations may be performed simultaneously with computation by one or two processors. There are no fixed assignments of I/O devices to I/O channels. All I/O operations are under control of the MCP.

I/O DEVICES

Through the Input/Output Exchange, I/O devices may be connected to the I/O channels through 32 input and 32 output points. The kinds and maximum number of standard I/O devices of each kind that can be used with the B 5500 are shown on the chart. The B 5500 can be easily adapted to accept a wide variety of special I/O equipment.



DATA COMMUNICATIONS

Through the I/O Exchange the Burroughs Data Communication Control Unit has full data communications network control capabilities. Dial TWX, teletype, inquiry typewriter, and other terminal units may be specified.

The high performance of the Burroughs B 5500 results from a high degree of simultaneous operation. This is made possible by the unique system organization shown on the accompanying chart. Among the key features of this organization are:

- The ability to expand the system to two processors, doubling computational performance.
- The ability to expand the system to eight independent magnetic core modules, permitting simultaneous memory accesses.
- The ability to expand the system to four "free-floating" input/output channels, each channel available to any I/O device and any core memory module.
- The ability to build your B 5500 system from a wide range of peripheral devices, including a flexible data communications network and the computer industry's fastest mass random access system.
- Two switching networks—the Central Exchange and the Input/Output Exchange—unique in their ability to allow fully flexible communication between processors, memory, I/O channels, and I/O devices.

TURN LEAF FOR CHART

ALL THESE PROVEN ACCOMPLISHMENTS, AND MORE, HAVE BEEN BUILT INTO THE NEW, MORE POWERFUL B 5500, GIVING YOU THESE IMPORTANT BENEFITS:

MORE THROUGHPUT PER DOLLAR INVESTED

Because of its advanced logical design, the Burroughs B 5500 works as a balanced system to accomplish more, in the same period of time, than more expensive computer systems which may appear (from raw specifications) to be faster. For example, the B 5500 has the most flexible input/output channel organization of any computer system on the market. And it employs thoroughly proven multiprocessing techniques that allow the system to handle automatically a variety of production, debugging and compiling runs simultaneously.

REDUCED PROGRAMING COSTS

Programing is simpler and less costly because of exclusive hardware/software features that enable the B 5500 to rapidly compile efficient programs written in COBOL, ALGOL, FORTRAN II and FORTRAN IV. With the ease of programing in these languages come more effective review and control of the programing activity, based on standard documentation and procedures.

REDUCED OPERATING COSTS

The Master Control Program of the B 5500 is the most complete, most advanced, most tested automatic operating system ever used to control and schedule computer operations. No other operating system provides the degree of automatic control found in the MCP—control that eliminates human error and uses the computer itself to assure efficient operation. Yet, no other operating system has been so thoroughly tested, in the "real world" of customer installations.

EXTENSIVE CAPACITY FOR EXPANSION, WITHOUT REPROGRAMING

The B 5500 can grow to a very large system (chart, next page). The modularity of the system is dynamic: you may expand or contract the system at any time without reprograming—or recompiling. This principle of dynamic, program-independent modularity is exclusive with Burroughs large scale computer systems. The MCP balances the current total program mix against the available system configuration, for most efficient operation under varying conditions.

Burroughs

SUPPORT IN DEPTH FOR PROFITABLE OPERATION

The on-site support you can expect from Burroughs Corporation is unsurpassed in the computer industry. From placement of an order to the successful operation of your B 5500—and beyond—you'll find competent, experienced Burroughs personnel at hand to assist you.

SYSTEMS COUNSEL, TRAINING, AND INSTALLATION CONTROL.

Your local Burroughs sales and technical representatives offer expert counsel in systems analysis and design, installation planning, and operation of your B 5500 installation. Your personnel are thoroughly trained in programing techniques and systems operation. In the important pre-installation period, on-site personnel are assisted by a computer-based Installation Control System monitored by an experienced staff at our Detroit, Michigan headquarters. These local and headquarters Burroughs personnel help keep you always on schedule, help you get your B 5500 on the air smoothly.

MANAGEMENT SCIENCES AND PROFESSIONAL SERVICES.

Consulting services are available to B 5500 users in the design and development of advanced management systems. Techniques in statistical analysis, linear programing, and network analysis are available, with a complete library of scientific routines.

SYSTEMS MAINTENANCE AND MTL.

A high level of system maintainability, unique in the computer industry, results from an exclusive Maintenance Test Logic (MTL) built into the B 5500. Armed with MTL, our specially trained Field Engineers are able to pinpoint trouble and correct it in far less time than by conventional techniques.

Burroughs
Burroughs

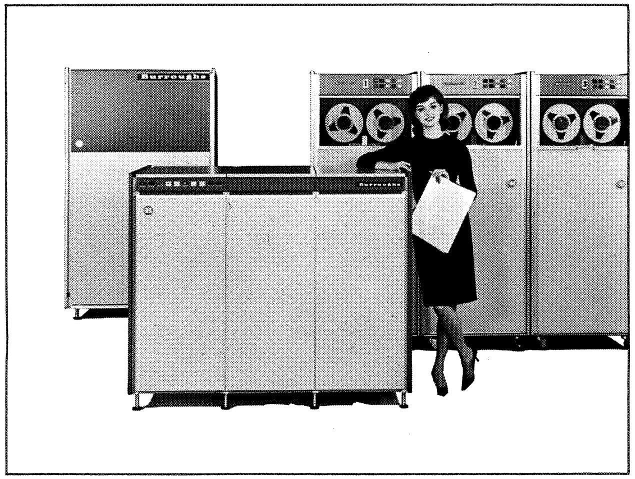
WORLD'S FASTEST, EASIEST TO USE MASS RANDOM ACCESS DEVICE

Burroughs ON-LINE DISK FILE

The powerful advanced systems concepts of the Burroughs B 5500 are fully complemented by the revolutionary Burroughs On-Line Disk File subsystem. With its "head-per-track" design, the Disk File provides all-electronic access to any record throughout the file in an average of 20 milliseconds.

File organization, programing, and use are simplified because access is entirely by electronic switching, with no moving arms, card drops, or the like. Each record segment is equally available regardless of physical location on the disks. Multiple segments can be transferred with a single instruction.

Module size is four disks totalling 9.6 million alphanumeric characters of information capacity. Up to 100 of these modules may be used with the Burroughs B 5500, effectively extending the memory of the computer system by almost a billion characters. Transfer rate is 100,000 characters per second.



Wherever There's Business There's Burroughs

Printed in U.S. America 6-65

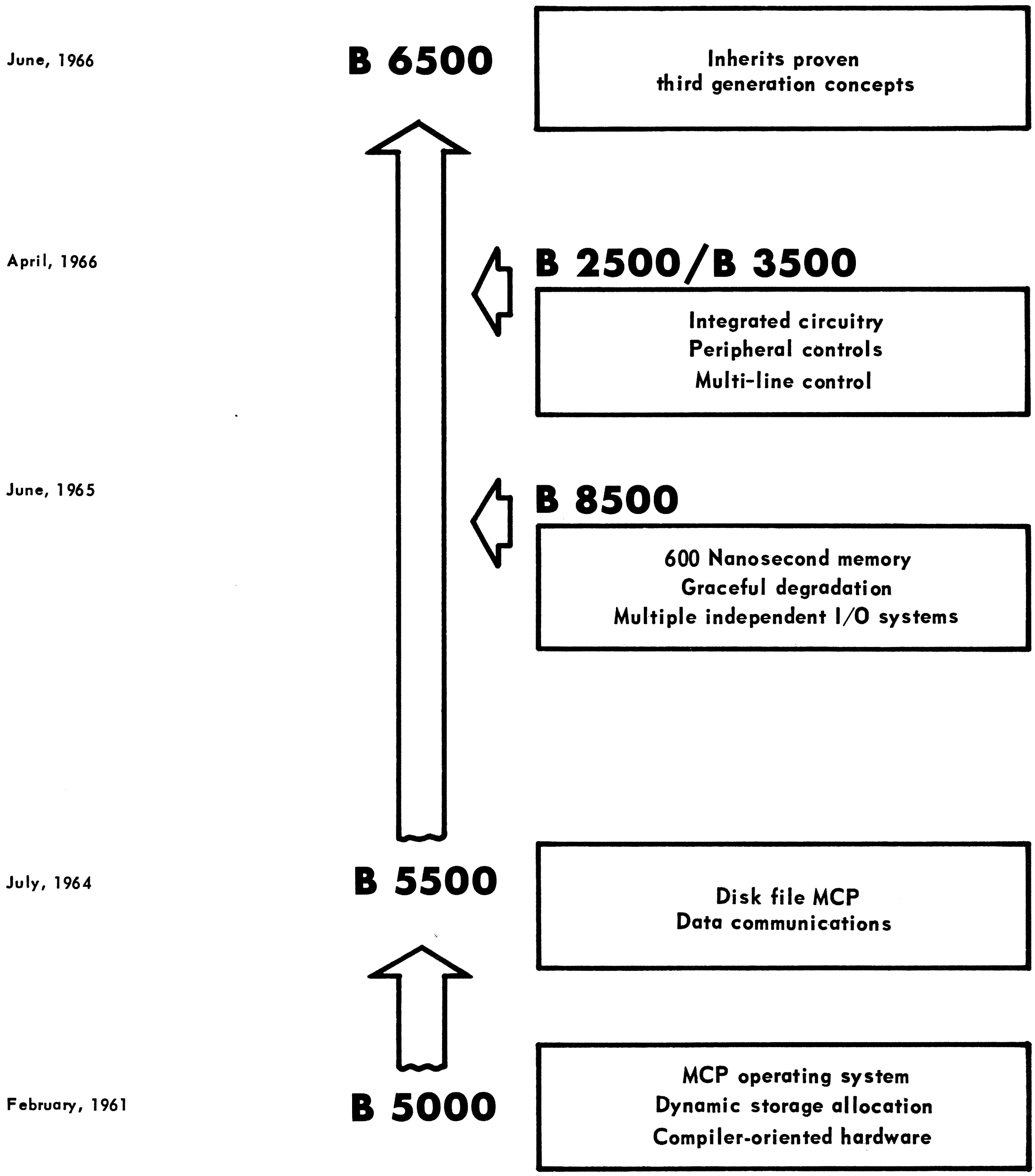
Burroughs B 6500

ELECTRONIC DATA PROCESSING SYSTEM

This report describes the B 6500 Electronic Data Processing System—a new Burroughs 500 System. A large scale system with capabilities falling between the intermediate scale B 5500 and the very large B 8500, the B 6500's normal mode of operation is multiprocessing and remote time-sharing under full operating system control. Compatible with the B 5500, the B 6500 makes direct use of that system's advanced, proved-in-use software and powerful hardware/software system design.

CONTENTS

Origin of the B 6500



June, 1966

B 6500

Inherits proven
third generation concepts

April, 1966

B 2500/B 3500

Integrated circuitry
Peripheral controls
Multi-line control

June, 1965

B 8500

600 Nanosecond memory
Graceful degradation
Multiple independent I/O systems

July, 1964

B 5500

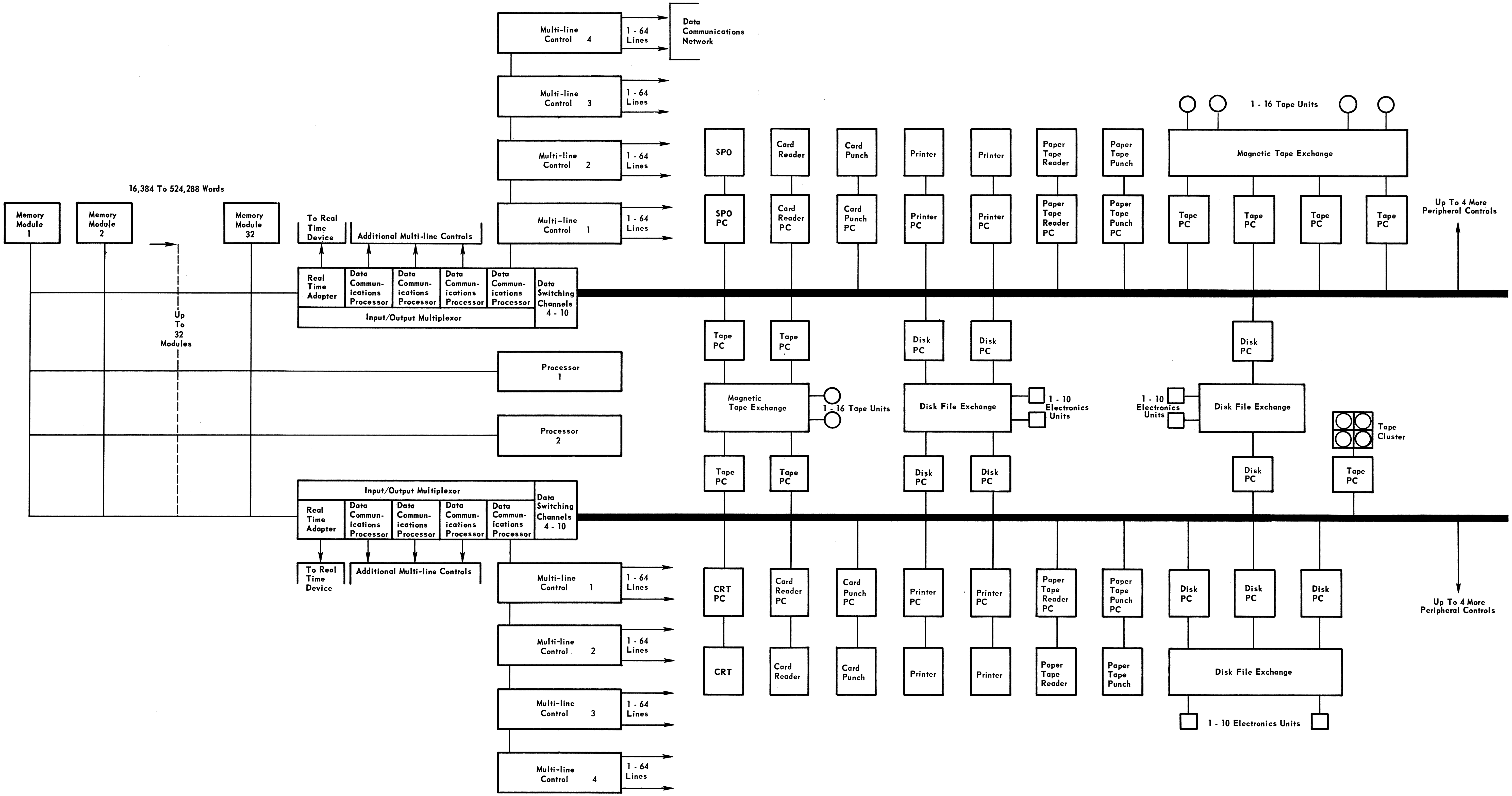
Disk file MCP
Data communications

February, 1961

B 5000

MCP operating system
Dynamic storage allocation
Compiler-oriented hardware

Burroughs B 6500 Schematic Diagram



I. THE ORIGIN OF THE BURROUGHS B 6500

As this report will show, the Burroughs B 6500 Electronic Data Processing System is highly impressive in its hardware and software specifications. Of equally impressive significance is the background of the manufacturer, Burroughs Corporation, with computer systems having both multiprocessing and self-regulating capabilities.

Burroughs 500 Systems, including the B 6500, all share these advanced capabilities. There are now five of these systems.

The B 2500 and B 3500, announced in 1966, are the first low- to medium-priced computer systems to offer both a comprehensive operating system and automatically controlled multiprocessing. They are characterized by high memory and operating speeds and multiple, simultaneous input/output processing capabilities. The B 2500 and B 3500 use integrated circuits and very high speed read-only memory and address memory to achieve high level performance.

The B 5500, announced and first delivered in 1964, is a more powerful version of the intermediate size Burroughs B 5000. This machine was the first computer system to be designed with a teamwork approach which saw software experts — particularly in the compiler and operating system disciplines — join with equipment engineers in formulating the basic specifications and equipment design. Common to all Burroughs 500 Systems, this teamwork design approach, coupled with years of installation experience, is the basis of Burroughs Corporation's unparalleled abilities in the successful design and support of self-regulating, multiprocessing computer systems. The soundness of this trend-setting design concept is affirmed by the many new third-generation computer systems which partially emulate the B 5500's design objectives, and proven by its solid record of success in B 5500 installations.

The B 6500, a binary machine like the B 5500, is a larger scale computer system capable of direct utilization of B 5500 software and program libraries.

The B 8500 is the most powerful computer system ever designed. It is extremely fast, and accommodates combinations of I/O modules and processors to a maximum of 16.

II. B 6500 HARDWARE

A. FUNCTIONAL DESIGN

The accompanying schematic diagram shows the functional design of the Burroughs B 6500. This hardware design is integrated with the design of key software components, especially the Master Control Program operating system, to provide for optimum execution of an object program for any hardware configuration. Automatic compensation is made for changes in configuration.

For example, when a B 6500 is expanded on the user's site to accommodate mounting workloads, neither reprograming nor recompiling is required. The Master Control Program (MCP) recognizes that a larger hardware configuration is available, and takes full advantge of the new environment by an immediate reallocation of memory, processor, I/O and peripheral resources to object programs.

Similarly, components may be removed from the system without destroying its ability to perform. As the diagram shows, the logical functions of the B 6500 are not disturbed by the removal of memory modules, Data Switching Channels, peripheral devices, a processor, an I/O multiplexor, or data communications controllers.

The MCP is able to recognize such omissions and perform its tasks accordingly. Thus, no single component failure can cause a complete system shutdown. The concept of "graceful degradation" allows operations to continue while corrections are being made.

B. CIRCUIT DESIGN

The clock rate of the B 6500 is five megacycles, allowing extremely fast instruction execution. Complementary transistor logic is employed throughout. This is the newest of several types of monolithic integrated circuits, and is the fastest, most flexible, least expensive type available.

A complete functional electronic stage is diffused into a piece of single crystal silicon .040 inches across; and is, conservatively, 10 times faster than conventional discrete element circuits.

C. MAIN MEMORY

The B 6500 main memory cycle time is 600 nanoseconds (billionths of a second) per word. The word size is 52 bits, consisting of 48 information bits, three control bits, and one parity bit. Memory is expandable in increments of 16,384 words to a current maximum of 524,288 words, the equivalent of 3,145,728 eight bit bytes.

The modular design of main memory allows for future expansion beyond one million words of storage.

Memory protection (preventing any program from affecting any other) is provided by a combination of hardware and software devices. One of the hardware features is automatic detection of an attempt by a program to index beyond an assigned data area. Another, the addition of a memory protect bit to each word, prevents user programs from writing into words of memory which have the protect bit set. Thus, a user program cannot change program segments, Data Descriptors, Program Descriptors, or various MCP tables during execution.

The eight bit Extended Binary Coded Decimal Interchange Code (EBCDIC) is the primary internal code of the B 6500, although input/output media recorded in four and six bit Binary Coded Decimal format is also compatible.

Second level memory for the B 6500 consists of Burroughs disk file subsystems. The disk file's head-per-track design simplifies the task of large volume storage of both program and data segments, and makes possible the very fast access speeds essential for effective utilization of second level storage techniques. The B 6500 MCP automatically transfers program or data segments to main memory as they are needed. Up to four transfer operations to or from each disk file subsystem can occur simultaneously through use of an optional Disk File Exchange. Multiple disk file subsystems totaling up to 19 billion bytes of storage may be attached to a B 6500 system.

D. PROCESSORS

The B 6500 accommodates one or two processors, both of which can access main memory. Expanding from a single-processor to a dual-processor configuration, when a rising workload reaches adequate proportions, results in a significant increase in the computational throughput of a B 6500 system at a modest increase in cost. The second processor can be installed on site. No reprograming is required to take full advantage of the expanded system.

The B 6500 Data Processor is a parallel machine operating at a clock frequency of five megacycles. Its operations are in the binary mode but extensive string manipulation capabilities allow handling of character-oriented data of four, six and eight bits. Since no character-set dependency is built in, the processor can handle data from many sources.

Extensive interrupt facilities allow Master Control Program intervention to provide resource allocation and object program error detection. Each processor can handle its own interrupts without disturbing the other. They may also share the processing of external interrupts generated by input/output operations occurring on either multiplexor. The command structure allows for Polish string notation and the automatic subroutine linking.

The processor is designed to implement higher level languages and to function under MCP control. For example, the major registers and control flip-flops in each of the processors contribute to the system's multiprocessing capabilities.

An aggressive hardware method of detecting and servicing system interrupts contributes to the B 6500's ability to process a mix of independent programs in an efficient manner. Under the constant, automatic management of the MCP, multiprocessing is the normal mode of operation for the Burroughs B 6500. With one processor in the configuration, multiprograming is the method used. Dual-processor B 6500 systems combine multiprograming and parallel processing.

The processor can function in either of two stages: the control state under the MCP, or the normal state in which the user program operates. When there are two processors in the system, each handles the interrupts associated with itself, e.g., both may be in control state at the same time.

E. INPUT/OUTPUT SYSTEM

The internal processing speed of the B 6500 is complemented by equally powerful input/output hardware to achieve a well-balanced computing system.

The transfer of data between memory and all peripheral devices is controlled by input/output multiplexors, independent of the processors. One or two of these multiplexors may be attached to a B 6500, each one capable of processing up to ten I/O operations simultaneously, from any of 255 peripheral devices.

Functioning through the I/O multiplexor, but independent of other peripherals, is a vast data communications network which can serve as many as 2,048 remote lines.

A multiplexor option, the Real Time Adapter, communicates with and controls such real-time operations as wind tunnels, rocket test stands and various process control equipment.

F. PERIPHERAL UNITS

The peripheral input/output equipment provided to accommodate engineering/ scientific and business application needs include the following:

Magnetic Tape Units

A choice of six models is available:
9393
Tape speed: 90 ips. Rewind speed: 300 ips.
Recording mode: 9-channel phase modulation.
Recording density: 1600 bpi.
Transfer rate: 144KB.

9392
Tape speed: 90 ips. Rewind speed: 300 ips.
Recording mode: 9-channel direct NRZ.
Recording density: 200 or 800 bpi.
Transfer rate: 18KB or 72KB.

9391
Tape speed: 90 ips. Rewind speed: 300 ips.
Recording mode: 7-channel direct NRZ.
Recording density: 200, 556 or 800 bpi.
Transfer rate: 18KC, 50KC or 72KC.

9383 (Magnetic Tape Cluster)
Two, three, or four tape stations.
Tape speed: 45 ips. Rewind speed: 90 ips.
Recording mode: 7-channel direct NRZ.
Recording density: 200, 556 or 800 bpi.
Transfer rate: 9KC, 25KC or 36KC.

9382 (Magnetic Tape Cluster)
Two, three, or four tape stations.
Tape speed: 45 ips. Rewind speed: 90 ips.
Recording mode: 9-channel phase modulation.
Recording density: 1600 bpi.
Transfer rate: 72KB.

9381 (Magnetic Tape Cluster)
Two, three, or four tape stations.
Tape speed: 45 ips. Rewind speed: 90 ips.
Recording mode: 9-channel direct NRZ.
Recording density: 200 or 800 bpi.
Transfer rate: 9KB or 36KB.

Line Printers

9243
1040 lines/minute.
120 or 132 print positions/line.

9242
815 lines/minute.
120 or 132 print positions/line.

9241
1040 lines/minute.
120 or 132 print positions/line.

9240
700 lines/minute.
120 or 132 print positions/line.

Card Readers

9112
1400 cards/minute.

9111
800 cards/minute.

Card Punches

9211
300 cards/minute.

9210
100 cards/minute.

Paper Tape Reader

9120
500 or 1000 characters/second.
5- through 8-channel tape.
Optional code translator.

Paper Tape Punch

9220
100 characters/second.
5- through 8-channel tape.
Optional code translator.

Disk File Devices

9372
10 million byte storage module.
20 millisecond access time.

All of the following disk files are expandable to a maximum of 19 billion bytes per B 6500.

9375-0
100 million byte data memory bank.
Additional 20 million byte increments.
23 millisecond access time.

9375-2
100 million byte data memory bank.
Additional 20 million byte increments.
40 millisecond access time.

9375-3
100 million byte data memory bank.
Additional 25 million byte increments.
60 millisecond access time.

G. DATA COMMUNICATIONS

Because the Burroughs B 6500 is designed for continuous multiprocessing and time sharing, the system readily accommodates applications and procedures requiring data communication. Real-time operations, remote computing, remote inquiry, and on-line programing merely become additions to the multiprocessing job mix of the B 6500.

The Data Communications Processor is the heart of the data communications network. Four of these processors may be attached to each multiplexor, each one communicating through four multi-line controls, with each control handling up to 64 remote lines. The Data Communications Processor is programable and contains the necessary registers and circuits to carry on such communication functions as polling, identification of control codes and delimiters, and all other line-checking features, independently of the main processor. The modular design of the data communications network allows each line to be adapted to serve virtually any data communications device. The programable nature of the data communications processor also allows generalized adapters to be used so that any remote device serviced by a given data set can be handled over one line to the B 6500.

H. TIME SHARING

Time sharing is a multiprocessing function for the Burroughs B 6500. The system has all the facilities required for highly efficient time sharing: very large and fast main memory capacity; the industry's fastest mass storage device with billions of bytes of capacity; independently functioning processors and I/O multiplexors; large, modular and easily adaptable data communications networks; and an interval timer which can be set in microsecond increments. All this hardware is monitored by a highly sophisticated Master Control Program which gives the B 6500 the very significant advantage of being able to maintain a time sharing network while simultaneously handling large volumes of on-site work.

This powerful capability eliminates the rigid time allocations ordinarily assigned for on-line vs. off-line processing. At a given moment, a Burroughs B 6500 may be processing a payroll, performing a matrix calculation, running a simulation problem, compiling programs written in ALGOL, FORTRAN, and COBOL, processing remote order entry and inventory management, and serving the users of a time sharing system. Every program run on the B 6500 is re-entrant: one copy of the program is capable of simultaneously serving many users. This results in faster service to the user, lower memory storage demands, and more efficient hardware utilization. Multiprocessing takes advantage of this re-entrant feature and is the key to a practical, workable time sharing operation. The B 6500 may be compared only to the B 5500 and the giant B 8500 in its dynamic multiprocessing capabilities.

III. B 6500 SOFTWARE

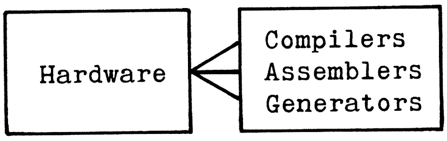
Software is defined as being those programs used to enhance systems efficiency, as opposed to those programs written specifically to solve particular problems.

A. MASTER CONTROL PROGRAM

The B 6500 Master Control Program is a most significant factor contributing to the system's smooth functioning, multiprocessing power, and balanced hardware utilization through simultaneity of operation. The MCP is the most advanced and the most comprehensive automatic operating system in existence. Yet, with the addition of expanded capabilities to reflect increases in hardware capability and configuration size, it is the same MCP found today in Burroughs B 5500 installations throughout the United States and abroad. Thus it is fully tested and proved in the "real world" of customer installations.

The Master Control Program redefines the basic relationship between the user and manufacturer of electronic computers. In the past, the computer manufacturers viewed the split of responsibilities between themselves and their customers in this manner:

THE VENDOR SUPPLIES:



High speed peripherals

Fast internal speeds

Software languages and generators

THE USER SUPPLIES:

Job scheduling

Data management

Program management

Resource management

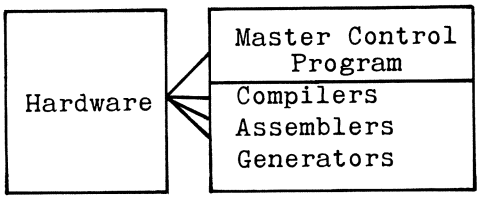
Problem definition for solution

The manufacturers thought of themselves as suppliers of raw power. The emphasis was on speed and power, with the design criteria established by the various engineering groups.

The software groups of computer manufacturers were used to build a modified logic bridge from this raw power to the user, by supplying languages, generators and service routines, designed to make it easier for the user to place his problem on the machine, get it checked out, and run successfully.

With the Master Control Program, however, a considerable amount of the burden has been transferred from user to manufacturer:

THE VENDOR SUPPLIES:



High speed peripherals

Fast internal speeds

Software languages and generators

Job scheduling

Data management

Program management

Resource management

THE USER SUPPLIES:

Problem definition, in the language best suited to his needs

The MCP for the Burroughs B 6500 also performs the following functions:

1. Scheduling and loading programs from a continuous flow job queue, and keeping a record of the programs in the mix.
2. Transferring program or data segments from the disk file subsystem to main memory as they are needed.
3. Maintaining an inventory of resources covering present status and availability of memory and peripheral units.
4. Allocating and overlaying main memory storage automatically.
5. Assigning peripheral units as required by the program.
6. Initiating input/output operations and recognizing their completion.
7. Removing completed programs from memory, and suspending lower priority programs for those with higher priority.
8. Notifying operator of required action.
9. Maintaining a log of systems operations, including total processor and peripheral time used by each program executed.

Knowledgeable computer users, consultants, and manufacturers alike predict that the comprehensive automatic operating system will have a profound effect on the computing and data processing function and organization. These effects include simplified computer room operations, dramatic improvements in turnaround times for engineering/scientific jobs and in throughput for data processing tasks, more extensive time-sharing capabilities, and better utilization of both the computer system and the people who work with it. These benefits are already being enjoyed by users of the MCP-controlled Burroughs B 5500.

This background gives Burroughs a dramatically unique position within the computer industry. Here is the only company building a totally integrated operating system upon the solid foundation of hundreds of user-years of multiprocessing, compiler-oriented experience.

B. PROGRAMING LANGUAGES

Higher level programing languages comprise the sole programing media for the B 6500. Complete diagnostic and debugging aids within the computer, plus hardware characteristics which implement fast, efficient compiling, eliminate the need for assembler-level coding. Programing and coding personnel are never required to work at the machine language level. Programing errors detected either during compilation or execution are automatically referenced back to the specific source language statement causing the error.

ALGOL

The most powerful program language in the engineering and scientific disciplines is implemented for B 6500 users by the Burroughs ALGOL–60 Extended compiler. This one-pass, fully tested compiler has been used extensively by B 5500 users for compile-and-go operations.

FORTRAN

The B 6500's compatibility with this widely used engineering/scientific language makes its services available to not only skilled technical programmers, but to professional engineers and scientists as well. Compatibility with existing program libraries minimizes conversion problems.

COBOL

For the business data processing user of the B 6500, a fully tested and operational one-pass COBOL compiler is available which provides unusual power and flexibility. Features of this compiler include the powerful SORT verb, and source language debugging.

Other computer systems, of course, are equipped with compilers for one or more of these languages. The Burroughs B 6500, however, is capable of simultaneous compilation of programs written in each language; of simultaneous compilation of more than one program written in the same language; and of simultaneously compiling multiple programs, in multiple languages, while performing production runs and handling time sharing work.

IV. CONCLUSION

The B 6500 design team has taken full advantage of the industry's most advanced hardware technology and software techniques to mold a completely modular system controlled by a proven, comprehensive operating system. Providing multiprocessing and time sharing as the normal mode of operation, these evolutionary systems are designed to meet the needs of intermediate to very large installations and to be gracefully expanded through this range. Dynamic, program-independent modularity allows the user to take full advantage of each new component without reprograming or recompiling.

The B 6500 offers new industry standards in main memory size and speed:

524,288 words or the equivalent of 3,145,728 bytes

600 nonasecond cycle time per 52 bit word

in mass storage speeds and accessibility:

20 to 60 millisecond average access

19 independent access paths

in data communications volume and flexibility:

2,048 remote lines

eight individual communications processors

in data processing power and capacity:

20 simultaneous input/output operations

12.8 million characters per second

and in proven software concepts:

the B 5500 Master Control Program

The dynamic, expanding organization will find the B 6500 System able to grow with it now and in the years to come.