RECENT IMPROVEMENTS IN THE SDP-3 COMPUTER

R. A. CLIFF
Flight Data Systems Branch
Spacecraft Technology Division
Goddard Space Flight Center
Greenbelt, Maryland

Abstract The SDP-3 is a small serial computer which will be flown as an engineering experiment on the IMP-I (eye) spacecraft. In this application it will handle data acquisition for four of the scientific experiments. Modifications to the SDP-3 which improve its operation with little change in hardware complexity include: (1) permitting full-word indirect addressing in user mode for all instructions which do not modify the contents of memory, (2) extending output channel capability to cover all of memory, (3) a new subroutine linkage instruction, (4) a conditional skip on memory zero instruction, (5) byte oriented instructions, and (6) improved index register manipulation instructions. The use of 8-bit MSI shift registers produces a significant hardware and power saving at the expense of deleting all left-shifting instructions. Further hardware savings are realized with a special multiplication instruction which allows the MQ register to be deleted. As a result of these, and other, modifications the power dissipation of the SDP-3 has been decreased by 2/3 watt and the number of IC's by 260 while in most respects operation has either been unchanged or improved.

Introduction The SDP (Spacecraft Data Processor) series consists of various small stored-program computers designed for small unmanned scientific spacecraft. These computers are designed to do data compression, data formatting, and control functions for the scientific experiments. The SDP-l is an extremely small machine of limited capability. (Ref. 111) It is a serial machine with a 12-bit word, 512 words of scratch pad memory, and 1024 words of NDRO program memory. We have done a number of experiments with this computer in order to determine its capabilities and limitations. We have done these experiments both with a simulated SDP-1 and with actual hardware. We plan to report on these experiments in another paper.

The SDP-2 is a "paper" machine for which the design was never completed. It was dropped in favor of the more promising SDP-3.

The SDP-3 computer was designed to be the core of the on-board data processing system of a proposed advanced IMP (Interplanetary Monitoring Platform) spacecraft. We are

flying the SDP-3 in an off-line mode as an engineering experiment on the IMP-I (eye) spacecraft. In this application it will service 4 scientific experiments.

With the incorporation of increased capability and a more serial architecture the SDP-3 became the SDP-3A. The SDP-3A (Ref. [2]) and its application to IMP-I (Ref. [3]) have been described previously. This paper covers an improved version of the computer--the SDP-3B. This is the version which will fly on IMP-I. A block diagram of the SDP-3B appears in Figure 1.

Two types of changes have been made to convert the SDP-3A to the SDP-3B. The first type is accomplished by small modifications to the hardware. These changes can have significant effect on the utility of the computer by greatly improving its efficiency in certain applications. Many of these modifications appear as changes in the instruction set of the computer. They are summarized in Appendix I. The second type of change involves extensive modification of the hardware. The object of this type is to reduce either the amount of hardware or the power drain (or both) without significantly degrading the capability of the computer. The two sections which follow cover these changes in detail.

Changes Which Improve Performance A review of the SDP-3A design revealed that improved performance could be easily obtained, at little or no cost in hardware, simply by changing a few connections between registers or moving a few wires in the control hardware. These changes are discussed below.

Interpage Access in User Mode The SDP-3 memory is divided into pages of 256 words each. A program in one page can communicate with another page only through indirect addressing or via the LINK instruction. In the SDP-3A design, interpage access by indirect addressing is blocked when the computer is in “User” mode. For program protection, all programs other than the Time Sharing Monitor (TSM) are run in the user mode. Since the LINK instruction is controlled by the TSM, this effectively prevents undesirable interaction between the various user programs.

It has been discovered, however, that several of the user programs will require more than one page of memory. The only possible communication between the several segments of one of these programs is via the LINK instruction (that is through the TSM). Unfortunately this mode of communication involves a large amount of overhead.

The problem is solved by permitting user programs to employ indirect addressing to access any word in memory for those instructions which do not modify the contents of memory or transfer control. Instructions which do modify the contents of memory or transfer control retain the previous restrictions on interpage access. The hardware required to produce this functional change is minimal. Previously the mode flip-flop

enabled full indirect addressing. Now the mode flip-flop OR'ed with a subset of the outputs of the instruction decoder is used to enable full indirect addressing.

Figure 2 shows a possible application of the new interpage access capability. Page A contains a user program which nearly fills the page and which requires a large data table. The data table is placed in Page B. The user program can read the table at will, as shown by the broken arrow from the table to the user program. The user program cannot directly modify the table, however. When the user program generates new data for the table it is put in a small buffer table in Page A. Then when this buffer is full the user program transfers control, through the TSM via LINK instructions, to a table write subroutine in Page B. This subprogram copies the buffer table from Page A into the large data table in Page B. It then returns control to the user main program through the TSM via LINK instructions.

Subroutine Linkage Linkage of subroutines via the LINK instruction (See Ref. [2]) is reserved for communication between the TSM and the user programs because only a few (16) LINK's are available. Two additional instructions were provided in the SDP-3A to facilitate subroutine linkage. These SDP-3A instructions (now obsolete) will be briefly described in order to provide an appreciation of the improvement provided by the SDP-3B instruction which supplants them.

The first of these instructions, XMPL, exchanged the contents of the Page Counter (PC) and Line Counter (LC) with the location in memory specified by the effective address (EA) of the XMPL instruction. Since PC and LC together determine the address of the next instruction which will be executed, the result of performing an XMPL is a form of "transfer and mark place." Control is transferred to the location specified by the contents of word EA. Simultaneously the former contents of PC and LC are saved in word EA.

If a location SUB is initialized with the address of a subroutine, then

```
XMPL    SUB
```

will initiate the subroutine. Then repeating the above instruction at the end of the subroutine will return control to the main program. Because it permitted transfer to any location in memory, this instruction was restricted to Monitor mode.

The second linkage instruction, STPL, stored the contents of PC and LC at location EA. The next instruction in the calling program would be an unconditional transfer to a subroutine. In order to return control to the main program the subroutine would first add I to the stored return address (to skip over the unconditional transfer instruction), and then do an indirect addressed unconditional transfer back to the main program.

The XMPL instruction had the disadvantage that it could not be employed in user mode because it modified the contents of PC. This restriction could have been removed by exchanging only LC in user mode, but the new linkage instruction which will be described is more flexible. The linkage using STPL was quite awkward and required 5 machine cycles. It was possible, however, to employ the STPL linkage in User Mode because the unconditional transfer is constrained to change only the LC in User mode.

Subroutine linkage in the SDP-3B was streamlined by deleting the XMPL and STPL instructions and adding a new instruction TSPL (Transfer and Store PC and LC). This instruction operates in a more conventional manner than the old ones.

Execution of the instruction

TSPL SUBRT

causes the contents of PC and LC (the address of the instruction following the TSPL instruction in memory) to be stored at location SUBRT. Instruction execution then proceeds from location SUBRT+1 as shown in Figure 3. The TSPL instruction had been considered during the design of the SDP-3A, but it was dropped in favor of XMPL and STPL because they were easier to implement. Subsequent modifications for other purposes supplied some of the hardware required for TSPL; therefore it is now the more attractive alternative. Appendix II gives details of the operation of the TSPL instruction. It is the most complex of the SDP-3B instructions.

Index Register Instructions The XMXR instruction, which exchanged the contents of the index register (XR) with the contents of the right half of the word in memory specified by the effective address, has been deleted. from the instruction list. It was dropped because it was infrequently used.

Index register loading has been modified also. In the SDP-3A the right half of the word specified by the effective address of the LDXR instruction was loaded into the XR. Now, in the SDP-3B, the instruction which loads the XR is called MDXR (Modify Index Register) and the effective address itself (modulo 256) is loaded into XR.

The change to MDXR is desirable for several reasons. It decreases the number of memory accesses (and hence the energy) required to load the XR. It also requires fewer words of core storage in the most frequent applications.

Assume, for example, that the XR contains significant data, but that the index register is needed to control execution of a loop. Since the STXR instruction does not affect the left half of the addressed word, it is possible to save the current contents of the XR in the

right half of an MDXR instruction. Execution of that MDXR instruction subsequent to exiting the loop will then restore the contents of the XR. Shown below is a sample program which uses this technique.

```
          .
          .
          .
          STXR      XRSTO       Save XR
          MDXR      75          Load XR
LOOP        —         —
            .                   Loop 75 times
            .                   Using XR
            .
          TDXR      LOOP
XRSTO     MDXR      **          Restore XR
            .
            .
            .
```

If the LDXR instruction had been used instead of MDXR, then two additional memory locations would have been necessary. One of these would have been used for temporary storage of the XR contents and the other would have held the constant 75.

If indirect addressing without indexing is specified for an MDXR instruction, then the XR is loaded from the word in memory specified by the address field of the MDXR instruction. For example, the instruction

MDXRI ZAP

loads the contents of the right half of location ZAP into the XR. On the other hand, if indexing without indirect addressing is specified for an MDXR instruction, then the XR is incremented by the address field of the MDXR instruction. For example, the instruction

MDXR X 3

will cause the XR to be incremented by 3. If both indirect addressing and indexing are specified in an MDXR instruction, then the XR is incremented by the contents of the location specified by the address field of the MDXR instruction. For example, the instruction

MDXR 1 X GLOM

will cause C(XR) + C(GLOM) to be loaded into the XR. Before the MDXR instruction was created, it took a series of 5 instructions to increment the XR by an arbitrary constant. Now it can be done with a single instruction.

Byte Operations The memory system consumes well over half of the total power used by the SDP-3. Therefore it is important to minimize the memory power drain if possible. Fortuitously, the SDP-3 memory is divided into two separate memories, each with an 8-bit word length. One of these memories supplies the left half of the,16-bit SDP-3 word and the other supplies the right half of the word. These memories draw significant power only when they are accessed. Consequently, only half as much energy is required to read or write an 8-bit half word as is required to read or write a 16-bit full word.

The SDP-3B has been modified to take advantage of half-word memory access in several ways. Indirect addressing in user mode need fetch only an 8-bit line number from the indirect address location for all those instructions which alter the contents of memory and for transfer instructions. The STXR instruction stores a half-word (it leaves the left half of the addressed memory location unchanged) and an indirect addressed MDXR instruction loads a half word. At present, Control Registers A and B (CA and CB) have been reduced to 8-bit length (they are readily re-expandable to 16-bits each); therefore, the LDCA and LDCB instructions now each load a half word.

Memory space is limited in the SDP-3 because the memory is both bulky and expensive. On IMP-I we will fly 4,096 of a possible 65,536 words. On the other hand, data items are frequently 8-bits or less in length. Therefore both memory space and power could often be saved by expanding the use of half-word instructions. Accordingly, the SDP-3 has been modified to include instructions which load or store half of the accumulator. These are LDAL and STAL for the left half (using the left half memory), and LDAR and STAR for the right half (using the right half memory). To facilitate handling tables of half-word data, two half word compare instructions, CMAL and CMAR, have been added to the SDP-3 repertoire. These two instructions compare the specified half of the accumulator with the specified word in the same half of the memory and then skip the next instruction if and only if the two half-words are equal. To further facilitate handling of half-word data, the instructions RRAL and RRAR have also been added. They rotate right the left or right half, respectively, of the accumulator.

Several of the mnemonics for instructions which manipulate the accumulator have been changed because the half-word byte oriented instructions were added. These mnemonics contained “AC” for accumulator. The present convention is to use “AL” for the left half of the accumulator, “AR” for the right half of the accumulator, and “AT” for the total accumulator. Table 1 lists the mnemonics which have been affected by this change. The functions performed remain the same. These instructions are also listed in the comprehensive list of changes in Appendix I.

Table 1

Old Mnemonic	New Mnemonic
LDAC	LDAT
STAC	STAT
XMAC	XMAT
CMAC	CMAT
TACZ	TZAT
TACM	TMAL
SRAC	SRAT
RRAC	RRAT

Output Channel The output channel of the SDP- 3A computer always read out page 15. This was quite sufficient for normal operation. Such a restriction is undesirable, however, when one wishes to dump memory to track down a suspected hardware or software malfunction. For this reason the enable output channel instruction, ENOC, has been modified. Now in the SDP-3B the programmer can specify (by the effective address of the ENOC) which page he wishes the output channel to read. The significant additional hardware required is an 8-bit register in the output channel which holds the number of the page which is to be read out. As there are presently 16 pages, this number is taken modulo 16 in the current version of the computer.

Miscellaneous A skip if memory is non-zero instruction (SKNZ) has been added to the SDP-3B. It is useful as a program switch in conjunction with the store zero (STZE) instruction. SKNZ can also be used to search through a table looking for an unused word, or in conjunction with the arithmetic operations which leave the result in memory. SKNZ is included because it is easy to realize and in some programs it will reduce the number of instructions required.

In the original SDP-3 the HALT instruction performed an unconditional transfer to the location specified by the effective address (when the computer was restarted). This was fine for operational programs; however, during debugging it was impossible to tell which HALT was being executed if there were more than one which transferred to the same location. To cure this problem, HALT has been modified to step to the next sequential instruction when the computer is restarted, rather than transferring. The address field (which now has no effect on the execution of the instruction) is then available for use as an identification field. This field will be found in the line register (LR) when the computer stops. Also, the Page Counter (PC) and Line Counter (LC) will contain the

address of the next sequential instruction, and thus also serve to identify the HALT indirectly.

The register in the SDP-3 which is compared to the Real Time Clock (CL) was formerly designated the Clock Register (CR). Occasionally the functions of the CL and CR were confused by some people. Therefore the CR has been renamed Time Register (TR) to make the distinction more obvious. The instruction which loads the TR is called LDTR; its function is identical to the former instruction LDCR.

These various modifications which have been discussed have considerably improved the usefulness and efficiency of the SDP-3 computer. The cost in hardware is negligible.

Changes Which Reduce the Amount of Hardware There are several reasons for desiring to reduce the amount of hardware in a system. In the SDP-3 the two chief motivations are increased reliability (with fewer parts there is less chance of failure) and decreased power drain. Additional benefits are easier packaging and easier testing. For these reasons, the SDP-3A was analyzed to see how the amount of hardware could be decreased without serious impact on the computer's capability. The resulting modifications which have been incorporated into the SDP-3B are described below.

Multiplication and Division When two N-bit numbers are multiplied together, the resulting product can require up to 2N bits. That this is so can be seen by observing that $(2^N - 1)(2^N - 1) = 2^{2N} - 2^{N+1} + 1$. It would seem, then, that to multiply two N-bit numbers would require 4 N-bit registers–one for the multiplicand, one for the multiplier, and two for the product.

Multiplication in a digital computer is usually done by repetitive shifting and adding using the recursion

$$C_j = a_j B 2^N + \frac{1}{2} C_{j-1} \tag{1}$$

where

$$A = \sum_{i=1}^{N} a_i 2^{i-1} \tag{2}$$

is the multiplier and B is the multiplicand. It is easily shown by induction that C_{N+1} is the product AB (taking $a_{N+1} = 0$). Observe that each bit of the multiplier is used only once and can then be discarded. Also observe that the significant part of the partial product C_j grows one bit larger each time j increases by one. As a result only 3 N-bit registers, rather than 4, are sufficient to perform multiplication. Figure 4 illustrates this method of multiplication.

Notice that the multiplier register is not very busy. It receives bits from the accumulator (one at a time) and has only one output (the multiplier bit currently being considered). One of the unique features of the SDP-3 is its split read/write memory cycle during the E-phase of each instruction. The desired memory location is read into the storage buffer register, the operation (which was set up during the I-phase) is performed, and then the storage buffer register is written back into the same memory location. Consequently instructions such as add to memory (ADDM) are possible.

Using this technique, the hardware multiplier register may be replaced by a pseudo-register in the memory. Instead of using a long right shift (SRAQ in the SDP- 3A) by one bit to shift the new partial product bit into the multiplier register and to dispose of a used multiplier bit, it is possible to use a new instruction. This new instruction (called PMUL, for partial multiply) reads the pseudo-multiplier register from the memory into the storage buffer register, shifts the accumulator and storage buffer register one position right, and then replaces the altered contents of the storage buffer register in the pseudo-multiplier register in memory. The program must of course be able to tell whether the least significant bit of the pseudo-multiplier is a 1 or a 0. This can be accomplished by skipping the next sequential instruction if and only if the bit in question is zero. Figure 5 shows a block diagram of multiplication using the PMUL instruction.

Given that multiplication will be accomplished via the PMUL instruction, the remaining use for the MQ register is holding the quotient and the least significant portion of the dividend during division. It happens that the MQ must be capable of shifting left in order to be useful for division. The section of this paper on MSI registers, however, explains why it is desirable from the standpoint of hardware reduction to remove all leftshifting instructions. It is anticipated that division will be a comparatively rare operation in the SDP-3; therefore, a fairly involved and lengthy software division process should be adequate. As a result, the MQ register can be deleted from the SDP-3A without serious loss of capability and with significant reduction of hardware and power drain.

Deletion of the MQ produces a direct saving of about 40 integrated circuits (IC’s) which dissipated about 140 mw. In addition, all instructions which manipulated the MQ may be deleted. These instructions loaded and stored the MQ (LDMQ and STMQ) and rotated and shifted the MQ (RRMQ, SRAQ,ARAQ). Three more instructions also used the MQ. They were leftshifting instructions and were also deleted for that reason. These instructions were Shift and Invert Bit Order (SIBO), Rotate Left AC and MQ (RLAQ), and Normalize (NORM). Thus an additional saving of the circuitry and power necessary to implement 8 instructions is realized.

Actually, the PMUL instruction is considerably more involved than the foregoing description implies. The SDP-3 is a two’s-complement machine; therefore it is desirable to have the capability of directly multiplying numbers in two’s-complement form. This

can be accomplished by the Booth algorithm. (Ref. [4] and Ref. [5]) For this algorithm the recursion formula is

$$C_j = (a_{j-1} - a_j) \cdot 2^N + \frac{1}{2} C_{j-1} \qquad (3)$$

where a_0 is taken to be 0. Now, the computer must either shift, shift and add, or shift and subtract depending on the values of a_i and a_{j-1} . The PMUL instruction allows for these three possibilities by either passing control to the next sequential instruction, skipping one instruction, or skipping two instructions. (See Table 2.)

Table 2
PMUL Instruction

a_j	a_{j-1}	Desired Function	Number of Instructions Skipped
0	0	Shift	2
0	1	Shift and add	0
1	0	Shift and subtract	1
1	1	Shift	2

At any stage of the multiplication process, a_j will be found in SR_8 (the right-most bit of the right half of the storage buffer register). At the same time a_{j-1} will be in a new 1-bit register, SX. The SX register is set to 0 at the beginning of multiplication (so that $a_0 = 0$) by the TOIM instruction. The TOIM instruction also sets the overflow flip-flop to zero. If the overflow flip-flop was a changed from 1 to 0 by the TOIM instruction, the instruction transfers control to the address specified. Otherwise, execution continues at the next sequential instruction. In the SDP-3B the TOIM instruction replaces the TOVF instruction of the SDP-3A.

In the SDP-3A each add or subtract instruction set the overflow flipflop to 0 if that instruction did not cause an overflow and set the overflow flip-flop to I if the instruction did cause an overflow. In contrast, the add and subtract instructions in the SDP-3B cannot set the overflow flip-flop to zero; they can only affect the overflow flip-flop by setting it to 1 if the operation overflows. Consequently, a TOIM instruction tests whether <u>any</u> addition or subtraction has overflowed since the last previous execution of a TOIM instruction. This allows a single TOIM instruction to check for overflow in any of the 16 possible additions or subtractions in the multiplication subroutine. This can save as many as 15 instructions (more than 1.1 ms) per multiplication as compared to checking each addition or subtraction separately.

A multiplication subroutine which uses the PMUL instruction is shown in Appendix III. This subroutine also demonstrates.the use of the TOIM, TSPL, and MDXR instructions. A program uses the multiplication subroutine by storing the multiplier at location MPYR and storing the multiplicand in location MCND. Then the sequence

```
          .
          .
          .
FAZZ    TSPL    MULT
        TUNC    ERROR
          .
          .
          .
```

calls the subroutine. Normally the subroutine returns control at location FAZZ+2. It leaves the most significant part of the product in the accumulator and the least significant part of the product in location MPYR. In case of overflow, the subroutine returns control at location FAZZ+1. This subroutine takes an average of 52 instruction cycles (4.0 rns).

MSI Registers Space flight qualified, low-power, 8-bit, TTL Medium Scale Integration (MSI) shift registers (Texas Instrument SN54L91R) have recently become available. These registers are compatible with the Fairchild LPDTμL circuits which were used exclusively in the SDP-3A. The MSI registers consume significantly less power than their “discrete” counterparts (18 mw vs. 36 mw) and occupy less space (1 package vs. 9 packages). The disadvantage of the MSI registers is that only serial output is available; therefore, the original “discrete” registers must be retained in all instances where parallel access to the contents of the register is required.

There are 11 registers in the SDP-3 which can trivially be replaced by MSI registers. These registers are the page counter PC , line counter (LQ, index register (XR), 4 registers in the input channel (DA, DB, DC, DD) and 4 registers in the output channel (BA, BB, BP, BL). The resulting saving is 88 IC’s and about 200 mw.

The original Real Time Clock subsystem consisted of a 16-bit ripple counter for the clock itself (CL) and a 16-bit shift register for the Time Register (TR). In addition there was a 16-bit parallel comparator which generated a priority interrupt request whenever the contents of CL were the same as the contents of TR. In the SDP-3B the 16-bit ripple counter is replaced by 2 8-bit MSI shift registers and a serial adder. The 16-bit shift register is replaced by 2 8-bit MSI shift registers. Then, the 16-bit parallel comparator is replaced by a 1-bit serial comparator. Furthermore, the contents of the new serial CL register are now transmitted in serial form directly to the memory during the store clock (STCL) instruction.

In the SDP-3A a 16-bit parallel gate was used to transfer the contents of the CL register (then a ripple counter) to the RT register (a multipurpose serial to parallel and parallel to serial converter) ‘ Another similar 16-bit parallel gate which transferred the state of the interrupt service request lines into the RT register (for the “Store Interrupt Request”

(STIR) instruction) was also eliminated. In order to accomplish this, the STIR instruction now uses the same transfer gates that the priority interrupt system uses. Consequently, the STIR instruction now stores the state of each interrupt service request line ANDed with the corresponding bit of the Mask Register (MR). This is not a serious restriction because one can disable the interrupt system, load the MR with all ones, perform an STIR instruction, restore the MR to its previous state, and then reenable the interrupt system and thereby perform the function formerly performed by the STIR instruction alone. Together, the modifications of the Real Time Clock and the STIR instruction save about 60 IC's and 50 mw.

The MSI shift registers (as previously mentioned) do not have parallel inputs or outputs. Furthermore, any one register can shift in only one direction. Therefore it is impossible to use MSI registers for left shifting in the SDP-3 unless they are loaded in serial from a non-MSI bidirectional register. All but one of the left-shifting instructions in the SDP-3A used the MQ register which has been deleted in the SDP-3B. Furthermore the left-shifting instructions added relatively little to the capability of the SDP-3; they were infrequently used and they can be simulated (painfully) by subroutines consisting of only right shifting instructions. For this reason the remaining left shifting instruction "Shift Left AC" (SLAC) has been deleted and the AC (now called the A register) has been converted to use MSI registers. The saving is 35 IC's and about 100 mw.

Miscellaneous The SDP-3A Experiment Interface Unit (EIU) contained provision for interfacing with 8 Experiment Interface Packages (EIP's). On IMP-I there are only 4 EIP's. Since we have no firm idea how many EIP's would be used in future applications (it would probably be more than 8 anyway) we decided to delete the EIU hardware which serviced the 4 nonexistent EIP's on IMP-I. We thereby saved 14 IC's and about 30 mw. Furthermore we helped alleviate a connector pin shortage by saving 16 pins.

Similarly, less than half of the bits of the Control A Register (CA) and less than half of the bits of the Control B Register (CB) are being used for IMP-I. Therefore each of these registers has been shortened to half-word length. The resulting saving is 18 IC's and 72 mw. We also save 16 more connector pins. An additional benefit is that half-word load instructions (LDCA and LDCB) use less energy than full word ones would. LDCA loads from the left half of the addressed memory word and LDCB loads from the right half of the word. This facilitates conservation of memory space.

The Mask Register (MR) and Control B Register (CB) were composed of clocked flip-flops in the SDP-3A. These flip-flops are packaged I per IC and consume more than 4 mw apiece. In the SDP-3B these registers have been converted to set-reset flip-flops made of NAND gates which are packaged4 per IC. It takes 3 NAND gates per register

stage to realize a flip-flop plus loading gate. The power dissipation using this method is 1 mw per stage. The savings realized are 6 IC's and 72 mw. for the 24 stages affected. The Control A Register (CA) must continue to use clocked flip-flops. The transient 0 output which would occur from any stage which was a I both before and after an LDCA instruction would disturb external devices connected to the CA register if set-reset flip-flops were used. There is no problem with the CB register because its output is a short pulse. The MR is sampled only at times when it is not being loaded, so there is no problem with it either.

Summary Full-word indirect addressing in user mode and the subroutine linkage instruction TSPL have greatly facilitated the writing of user programs for the SDP-3B. Byte oriented instructions and improved index register instructions conserve power and memory space, both desirable goals in a spaceborne computer. Table 3 summarizes the hardware and power savings obtained by the use of MSI registers and the other modifications which were discussed. The SDP-3B exclusive of the memories consists of 500 IC's which consume about 1.3 watts. Reference to Table 3 reveals that the SDP-3A had in excess of 50% more IC's and used in excess of 50% more power than the SDP-3B.

Table 3

Modification	Integrated Circuits Saved	Power Saved (mw)
Delete MQ	40	140
MSI Registers	88	200
Real Time Clock	60	50
Delete Left AC	35	100
EIU Reduction	15	72
CA and CB	18	35
MR and CB	6	72
	Total 262	669

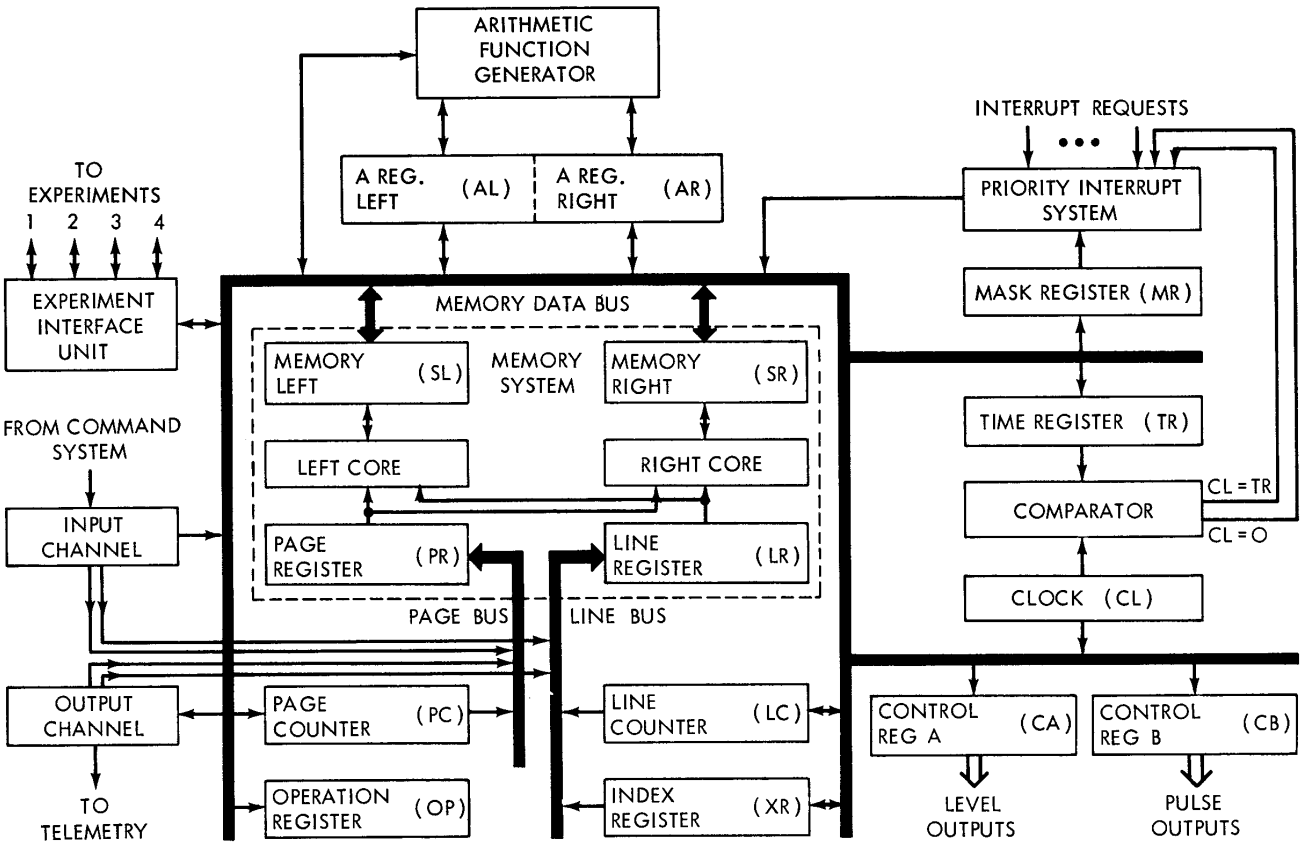

Fig. 1 - SDP-3B Block Diagram

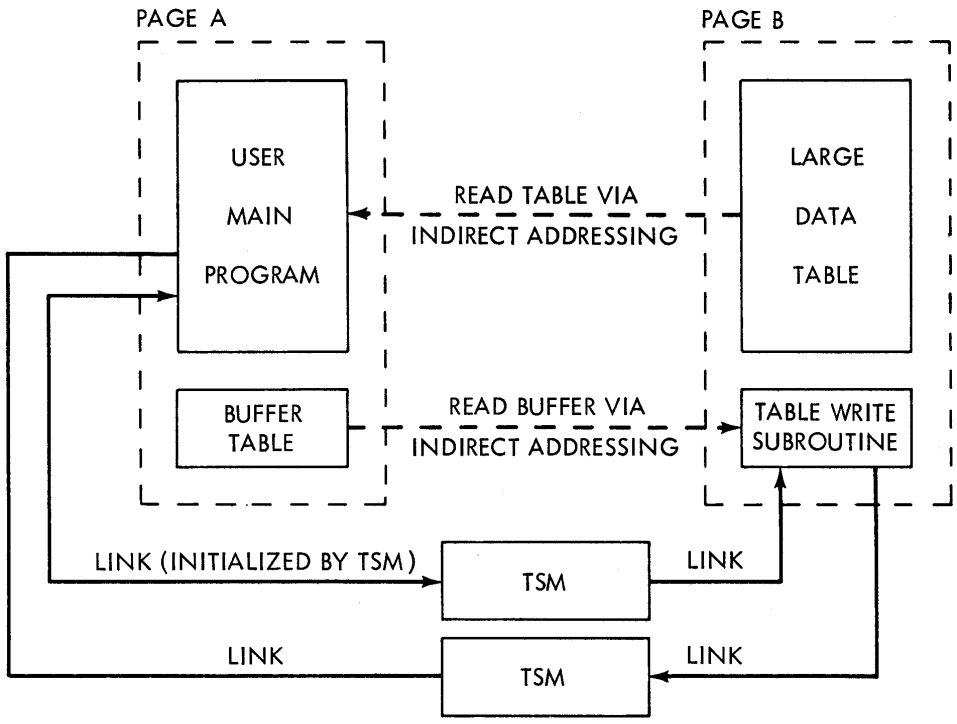

Fig. 2 - Multipage Program Example

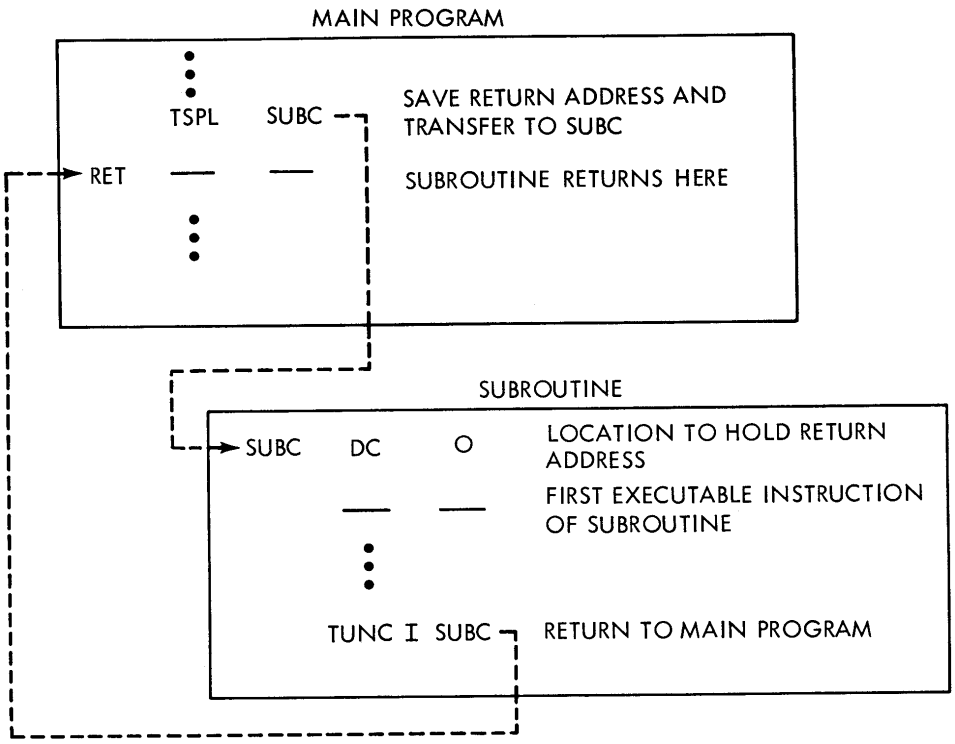

Fig. 3 - TSPL Subroutine Linkage

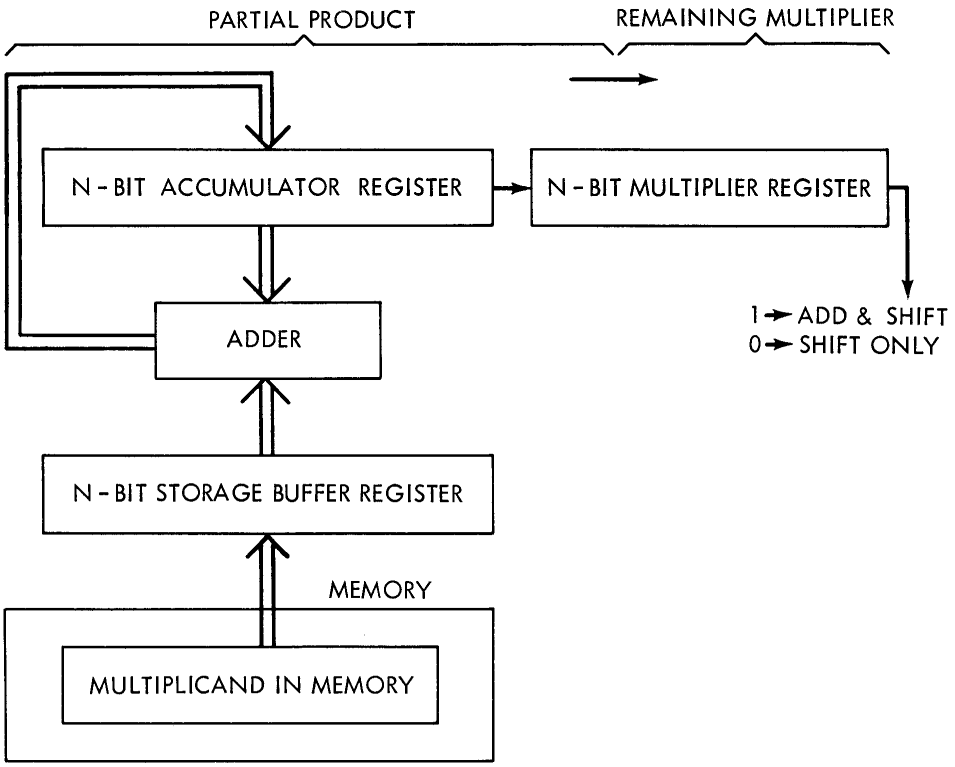

Fig. 4 - The Usual Method of Multiplication

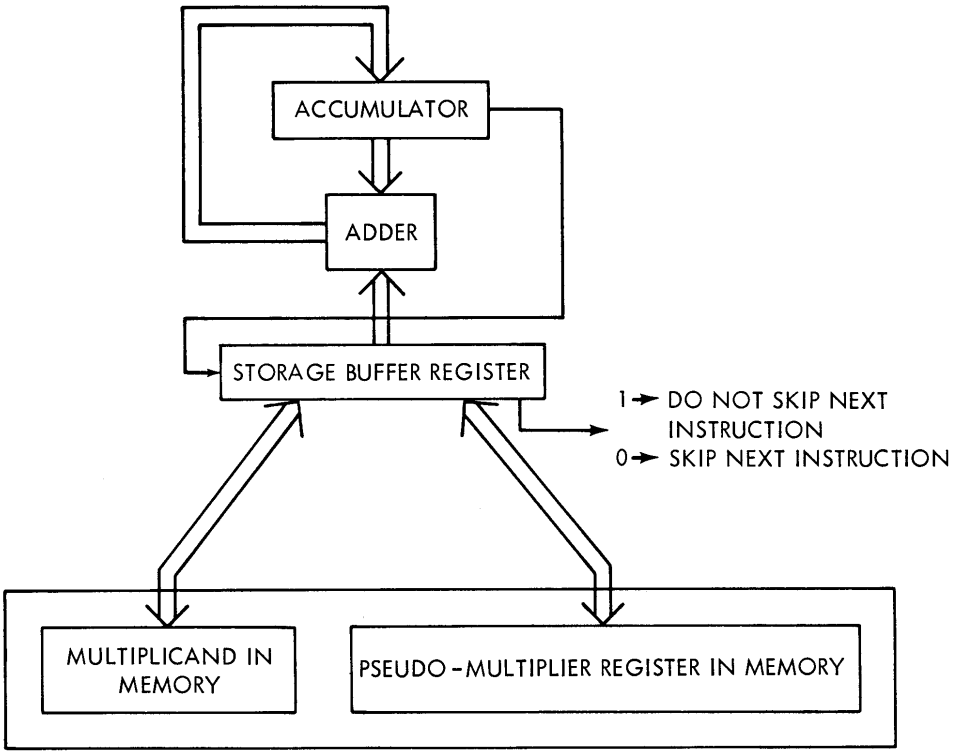

Fig. 5 - Multiplication Using PMUL

Summary of Instruction Set Changes

A. The SDP-3A Instruction Set With Corresponding SDP-3B Instructions

OP Code	SDP-3A Mnemonic	SDP-3B Mnemonic (If Different)	Changes in Operation for SDP-3B
00	HALT		Does not branch
01	XMPL	Deleted	Deleted
02	XMIN		
03	DINT		
04	LDCA		Uses left half of word
05	LDCB		Uses right half of word
06	LDMR		
07	LDCR	LDTR	
10	LINK		
11	CMAC	CMAT	
12	TUNC		
13	TDXR		
14	TACZ	TZAT	
15	TACM	TMAL	
16	TMQE	TEAR	Uses least significant bit of A
17	TOVF	TOIM	Resets SX and Overflow
20	SIBO	Deleted	Deleted
21	RLAQ	Deleted	Deleted
22	SLAC	Deleted	Deleted
23	SRAC	SRAT	
24	SRAQ	Deleted	Deleted
25	ARAQ	SRNU	Uses A only
26	RRAC	RRAT	
27	RRMQ	Deleted	Deleted
30	STAC	STAT	
31	STMQ	Deleted	Deleted
32	STXR		Uses right half of word

OP Code	SDP-3A Mnenomic	SDP-3B Mnemonic (If Different)	Changes in Operation for SDP-3B
33	STCL		
34	STPL	Deleted	Deleted (see TSPL)
35	STIR		Stores requests ANDed with MR
36	STZE		
37	XMXR	Deleted	Deleted
40	LDAC	LDAT	
41	LDMQ	Deleted	Deleted
42	LDXR	MDXR	Loads Effective Address (EA) into XR
43	XMAC	XMAT	
44	ADDA		
45	ADDM		
46	SUBA		
47	SUBM		
50	IORA		
51	IORM		
52	ANDA		
53	ANDM		
54	EORA		
55	EORM		
56	MPAA		
57	MPSA		
60	NORM	Deleted	Deleted
61			
62			
63			
64			
65			
66			
67			
70	ENOC		Can read any page
71	DSOC		

OP Code	SDP-3A Mnemonic	SDP-3B Mnemonic (If Different)	Changes in Operation for SDP-3B
72	XMIB		
73	XMIA		
74	EINT		
75			
76			
77			

B. The SDP-3B Instruction Set With Corresponding SDP-3A Instructions

OP Code	SDP-3B Mnemonic	SDP-3A Mnemonic (If Different)	Changes in Operation for SDP-3B
00	HALT		Does not branch
01	LDTR	LDCR	
02	XMIN		
03	DINT		
04	LDCA		Uses left half of word
05	LDCB		Uses right half of word
06	LDMR		
07			
10	LINK		
11	CMAT	CMAC	
12	TUNC		
13	TDXR		
14	TZAT	TACZ	
15	TMAL	TACM	
16	TEAR	TMQE	Uses least significant bit of A
17	TOIM	TOVF	Resets SX and Overflow
20			
21			
22	RRAR	none	Byte Operation
23	SRAT	SRAC	
24	RRAL	none	Byte Operation

OP Code	SDP-3B Mnemonic	SDP-3A Mnemonic (If Different)	Changes in Operation for SDP-3B
25	SRNU	ARAQ	Uses A only
26	RRAT	RRAC	
27			
30	STAT	STAC	
31	STCL		
32	STXR		Uses right half of word
33			
34	SKNZ	none	New Instruction
35	STIR		Stores request ANDed with MR
36	STZE		
37	PMUL	none	New Instruction
40	LDAT	LDAC	
41	TSPL	none	New Instruction
42	MDXR	LDXR	Loads Effective Address (EA) into XR
43	XMAT	XMAC	
44	ADDA		
45	ADDM		
46	SUBA		
47	SUBM		
50	IORA		
51	IORM		
52	ANDA		
53	ANDM		
54	EORA		
55	EORM		
56	MPAA		
57	MPSA		
60			
61	LDAL	none	Byte Operation
62	LDAR	none	Byte Operation
63	CMAL	none	Byte Operation

OP Code	SDP-3B Mnemonic	SDP-3A Mnemonic (If Different)	Changes in Operation for SDP-3B
64	CMAR	none	Byte Operation
65	STAL	none	Byte Operation
66	STAR	none	Byte Operation
67			
70	ENOC		Can read any page
71	DSOC		
72	XMIB		
73	XMIA		
74	EINT		
75			
76			
77			

Appendix II.
TSPL Instruction Dissected

Time Interval	Bit Times	Register Transfers	Function
q_A	8	LC → LR LC+1 → LC	Address of TSPL
		PC → PR PC → PC	instruction to memory
q_B	4	GM⟨PR:LR⟩ → SL:SR	Fetch and restore TSPL instruction
		0 → GM⟨PR:LR⟩	
q_C	4	SL:SR → GM⟨PR:LR⟩	
		SL:SR → SL:SR	
q_D	8	SL → OP, ∅ → SL	TSPL to operation register
		SR → LR SR → SR	Address Indirect address Location
q_E	4	f(GM⟨PR:LR⟩) + f'(SL:SR) → SL:SR	Fetch and restore indirect address location
		f(0) + f'(GM⟨PR:LR⟩) → GM⟨PR:LR⟩	
q_F	4	f(SL:SR) + f'(GM⟨PR:LR⟩) → GM⟨PR:LR⟩	
		SL:SR → SL:SR	
q_G	8	t[S(XR,SR)] + t' SR → LR	Compute Effective Address (EA)
		∅ → SR ∅ → SL XR → XR	
		mf SL + (mf)' PR → PR	
		x{t[S(XR,SR)] + t' SR} + x' LC → LC	
		xmf SL + (xmf)' PC → PC	
q_H	4	0 → GM⟨PR:LR⟩	Clear word in memory
q_M	8	LR → LR, LR → LC LC → SR	Swaps old and new contents of PC and LC
		PR → PR PR → PC PC → SL	
q_N	8	LC+1 → LC	Increment LC to skip one instruction
q_J	4	SL:SR → GM⟨PR:LR⟩	Store former PC and LC contents

Notes on Appendix II

x = 1 for instructions which transfer (includes TSPL)

m = 1 for Monitor Mode

t = 1 for indexing (tag bit)

f = 1 for indirect addressing (flag bit)

∅ is any combination of 1's and 0's

GM <PR:LR> is the location in memory addressed by PR and LR

Appendix III.

Multiplication Subroutine

```
MULT    DC      0           RETURN ADDRESS FOR OVERFLOW
        LDAT    ONE         GENERATE RETURN ADDRESS
        ADDM    MULT          FOR NO OVERFLOW
        LDAT    ZERO        INITIALIZE
        MDXR    15          INITIALIZE
        TOIM    *+1         INITIALIZE
*
PM      PMUL    MPYR        SHIFT
        TUNC    ADD         GO ADD MCND
        SUBA    MCND        SUBTRACT MCND
        TDXR    PM
*
GO      TOIM    OFLOW
        PMUL    MPYR        C = 1/2 C(16)
        TUNC I  MULT        RETURN
        TUNC I  MULT        RETURN
        TUNC I  MULT        RETURN
*
ADD     ADDA    MCND
        TDXR    PM
        TUNC    GO
*
OFLOW   MDXR    -1
        TUNC I X MULT       OVERFLOW RETURN
*
ZERO    DC      0
ONE     DC      1
MPYR    DC      0
MCND    DC      0
```

References

[1] R. A. Cliff, “The SDP-1 Stored Program Computer,” Trans IEEE, Vol. AES-4, No. 6, pp 864-870, Nov. 1968.

[2] R. A. Cliff, “The SDP-3--A Computer For Use On Board Small Scientific Spacecraft,” EASCON 168 Record, pp 521-5Z7, Sept. 1968.

[3] R. A. Cliff and S. Paull, “The IMP-I Computer Experiment,” NTC 169 Record, pp 176-183, April 1969.

[4] A. D. Booth, “A Signed Binary Multiplication Technique,” Quart. Journ. Mech. and Applied Math., Vol. IV, Pt. 2, pp 236-240, 1951.

[5] Y. Chu, “Digital Computer Design Fundamentals,” McGraw-Hill, N. Y., N. Y., 1962, pp 32-34.