An Intelligent, Fast Disk Controller
Using the Am29116

By Paul Chu, Brad Kitson, and Otis Tabler
Advanced Micro Devices
Until recently, advances in high-performance disk systems were limited mainly by the state of the art in Read/Write circuits and head. Today, track densities and transfer rates are becoming so high that the design of the controller is becoming a bottleneck. The need for high bandwidth is accompanied by demands for more powerful command sets and the transfer of many operating system software tasks into the controller firmware.

To implement intelligent high-bandwidth controllers, flexible and very fast VLSI building blocks are needed. This article shows how two such building blocks, the Am29116 Bipolar Microprocessor and the Am9520 Burst Error Processor, can be combined to form a disk controller with over 20MHz bandwidth, and incorporate such features as detection and correction of burst errors up to 11 bits long, I/O request queue sorting, sector caching, device transparency, logical record I/O, and associative (content-addressed) reading and writing of logical records.

The Am29116 performs 10 million instructions per second within a 16-bit parallel architecture and 32 x 16 register file. Its 16-bit barrel shifter allows an operand to be masked and rotated from 1 to 15 places and then optionally compared with a second operand within a single instruction cycle. Within a single cycle, it is also possible to rotate an operand and merge it with a second operand under a mask.

Other important features of the Am29116 includes its generation of forward and reverse CRCs; its ability to prioritize event and status bits under mask; and its ability to set, reset, and test arbitrary bits. The Am29116 is the largest and most complex such bipolar device produced. Fabricated using AMD's proprietary ion-implanted oxide-isolated (IMOX™) process, it contains emitter-coupled logic (ECL) circuitry scaled to VLSI proportions. Although ECL is used internally, all input and output buffers are fully TTL-compatible.

The Am9520's features, which make it a cornerstone of this design, include the ability to generate check bits and detect and correct single and burst errors for four different modified Fire code polynomials—including the popular 48-bit polynomial and the exceptionally powerful 56-bit polynomial used in this design. High throughput of the Am9520 is achieved by using an 8-bit parallel network of exclusive OR gates that accomplishes the equivalent, in a single clock, of eight clockings of a linear feedback shift register. In less than 200 microseconds, the correct high speed mode of the Am9520, which is used in this design, permits correction of a maximum-length error burst (11 bits) anywhere within a 256-byte sector using the microcode logic shown and the 56-bit polynomial. The Am9520 performs the correct high-speed function by simultaneously dividing the data input by all of the factors (except the first) of the polynomial. Location and correction of the error burst is fast because the periods of the factors are short compared with the period of the composite polynomial.

IMOX is a trademark of Advanced Micro Devices, Inc.
Am29116 Organization
The Am29116 includes a 32 x 16 RAM with latched outputs, a 16-bit accumulator, a 16-bit data input latch, a 16-bit barrel shifter, a three-input arithmetic/logic unit, a 16-bit priority encoder, a status register, a condition-code generator/multiplexer, 16 tristate output buffers and a 16-bit instruction latch and decoder (Figure 1).

The single-port RAM has output latches that are transparent when the clock input CP is HIGH and latched when CP is LOW. Data is written into the RAM while the clock is low if the TEN input is also LOW and if the instruction being executed selects the RAM as destination. Data is written into the low-order 8 bits of the addressed word for byte instructions and into all 16 bits for word instructions. Separate read and write RAM addresses may be used by supplying a multiplexer on instruction inputs I4-I0 using CP as the select signal.

The accumulator, which is edge-triggered, accepts data on the LOW-to-HIGH transition of CP if TEN is also LOW and if the instruction being executed selects it as the

Figure 1. Am29116 Organization
destination. As with RAM locations, byte instructions modify only the lower half of the accumulator while word instructions modify the full register.

The data input latch (D-latch) holds the data input to the ALU on the bidirectional Y bus. The latch is transparent when the DLE input is HIGH and latched when the DLE input is LOW. The sources of the ALU operation are the RAM, the accumulator, the D-latch and the instruction inputs during IMMEDIATE instructions.

The ALU, which can operate on one, two, or three operands depending upon the instruction being executed, contains full carry lookahead across all 16 bits. All ALU operations can be performed in either word or byte mode. Status outputs Carry (C), Negative (N), and Overflow (OVR) are generated at the byte level for byte-mode operations and at the word level for word-mode operations. A fourth flag, Zero (Z), is generated outside the ALU and also operates in either byte or word mode. The Stored Carry (QC) bit of the status register may be selected (along with 0 and 1) as the ALU carry input to support multi-precision arithmetic operations. This is used by the correct high speed microcode of the disk controller, which employs coefficients as large as 2,647,216.

The priority encoder produces a binary-weighted code to indicate the location of the highest-order non-masked one at its input. If none of the masked bits is HIGH, the output of the priority encoder is zero. If bit i is the most significant HIGH bit then the output of the priority encoder is equal to $s - i + 1$ where $s$ is the position of the sign bit and is equal to 15 in word mode and 7 in byte mode. To understand why $s - i + 1$ is used in place of $s - i$ (the usual priority encoding), consider the following example (Figure 2). The eight Attention Drive signals are presented on the time-

![Figure 2. Using the Am29116 Prioritize Instruction](image-url)
multiplexed drive command/data bus and are read through the Y-bus and data input latch of the Am29116. If the controller has already serviced all attention requests from drives 1-3 and wishes to service the highest priority attention request (if any) from drives 4-8, it executes a Prioritize instruction in byte mode using hexadecimal 001F as the mask, followed by a Branch if Not Zero into a jump table indexed by the priority encoder output.

The 8-bit status register and the condition-code generator/multiplexer contain the information and logic necessary to develop 12 condition-code test signals. The multiplexer selects one test signal and places it on the condition-code test (CT) output for use by the microprogram sequencer. The multiplexer is addressed in two ways. In the first, which is used here to maximize throughput, the T-bus is used in input-only mode to specify the multiplexer select position directly. In the second, the CT output is selected through a test instruction.

The output enable Y-bus (OEY) input enables the 16 tristate output buffers when it is LOW. When OEY is HIGH, the output buffers are read in the high-impedance state (allowing read/write and status data to reach the D-latch from the controller's 16-bit system data bus).

The 16-bit instruction latch is normally transparent to allow decoding of the 16 instruction inputs I15-0 into internal control signals for the Am29116 and the execution of the instruction within a single clock cycle. The only exceptions to this rule are the immediate-operand instructions, which execute in two clock cycles rather than one. These are captured in the instruction latch during the first clock and executed during the second. It is during the second clock that the immediate operand, which resides in the I15-0 field of the next microinstruction, is fetched and execution is completed. Immediate instructions are used extensively in the disk controller microcode whenever masks and special arithmetic constants are needed. (The Am29116 allows the addition or subtraction of $2^N$, and the use of $2^N$ and its complement as a mask, for any $N$ between 0 and 15 within a single clock, so that for these 16 common numbers and 32 common masks, an immediate instruction is not required).

Am29116 Instructions

The 16-bit instructions of the Am29116 can be grouped into eleven types which correspond in a natural way with the Am29116's internal instruction decoding logic: single operand, two operand, single bit shift, rotate and merge, bit oriented, rotate by n bits, rotate and compare, prioritize, cyclic redundancy check, status, and no-op. The microprogrammer needs to be familiar with these groupings (and certain subgroupings) because the System 29 AMDASM DEF file provides mnemonics that correspond to them. For example, the AMDASM SRC file line

```
SOR W,INC,SORY,R1
```

increments the full 16-bit contents of Am29116 RAM location 1 by one and places it onto the Y-bus and

```
TOR1 B,SUBR,TORAR,R2
```

subtracts the low-order byte of the
accumulator from the low-order byte of RAM location 2 while leaving the high-order byte of location 2 unchanged.

Table 1 summarizes the basic operations that Am29116 instructions can perform within a single cycle. (Two cycles are used if one operand is immediate data.) Note that for a typical line of this table, there are several Am29116 mnemonic operation codes, depending upon the choice of operand source(s) and destination.

**TABLE 1. SINGLE-CLOCK Am29116 OPERATIONS**

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
</tr>
<tr>
<td>Add with Carry</td>
</tr>
<tr>
<td>Add $2^N$</td>
</tr>
<tr>
<td>And</td>
</tr>
<tr>
<td>Complement</td>
</tr>
<tr>
<td>Accumulate forward CRC</td>
</tr>
<tr>
<td>Accumulate reverse CRC</td>
</tr>
<tr>
<td>Exclusive Nor</td>
</tr>
<tr>
<td>Exclusive Or</td>
</tr>
<tr>
<td>Increment</td>
</tr>
<tr>
<td>Load $2^N$</td>
</tr>
<tr>
<td>Load $2^N$ Complemented</td>
</tr>
<tr>
<td>Move</td>
</tr>
<tr>
<td>Nand</td>
</tr>
<tr>
<td>Negate (2's complement)</td>
</tr>
<tr>
<td>Nor</td>
</tr>
<tr>
<td>Or</td>
</tr>
<tr>
<td>Prioritize under mask</td>
</tr>
<tr>
<td>Reset bit N</td>
</tr>
<tr>
<td>Reset status bit</td>
</tr>
<tr>
<td>Rotate N bits</td>
</tr>
<tr>
<td>Rotate N bits and compare under mask</td>
</tr>
<tr>
<td>Rotate N bits and merge according to mask</td>
</tr>
<tr>
<td>Set bit N</td>
</tr>
<tr>
<td>Set status bit</td>
</tr>
<tr>
<td>Single bit shift</td>
</tr>
<tr>
<td>Subtract</td>
</tr>
<tr>
<td>Subtract with Carry</td>
</tr>
<tr>
<td>Subtract $2^N$</td>
</tr>
<tr>
<td>Test bit N</td>
</tr>
<tr>
<td>Test status bit</td>
</tr>
</tbody>
</table>
Many of the operations prove particularly useful when implementing intelligent disk controllers. For example, the packing of ASCII characters (which occupy 8-bit bytes in main memory yet need only occupy 7-bit contiguous frames in the disk record) is accomplished efficiently and at high speed by Rotate and Merge instructions as shown in Figure 3. The microinstructions shown on the arrows perform the bit mapping indicated by them. In this example, 8 ASCII bytes requiring 64 bits of main storage are packed into 56 bits (8 7-bit contiguous frames) prior to being written to disk. In general, the ability of the Am29116 to rotate a 16-bit operand by N bits and merge it with a second 16-bit operand under mask within a single cycle makes the manipulation of arbitrary-length, arbitrarily-aligned data fields efficient and simple. Other operations that are especially valuable in this application are provided by the CRC Forward instruction (used to generate or check the integrity of header records), the instructions which add and subtract 2^N, load 2^N and its complement, reset, set, and test bit N, and (as indicated above) the masked Prioritize instruction. If the intelligence incorporated into the controller includes associative retrieval based upon recognition of an arbitrary bit string within the data record, the instructions which rotate by N bits and then (within the same cycle) compare under mask are almost indispensable.

**Functions of An Intelligent Controller**

The interface signal names, polarities, and functions used in this article are similar to those used in the current ANSI standard for rigid disks. However, the methods and

![Figure 3. Packing ASCII Fields by Means of the Rotate and Merge Instruction](image)
functions discussed can be used for most current rigid or flexible disk drives. With minimal external logic, the Am29116 and Am9520 perform all the functions needed to format, read, and write disks at over 20 MegaBits per second. These include generating and checking header CRCs, performing header-sector acquisitions, enabling and disabling drive read/write circuits at the appropriate times, managing data flow through a high-speed buffer memory, generating check bits during writes, and detecting and correcting single and burst errors up to 11 bits long during reads.

Except for seeks, retries, and formatting, all of the above have been microcoded. The microcode fits within 256 words (one-fourth of the microprogram memory used in the design), and it is appropriate here to describe some additional intelligent functions that can be microprogrammed:

Maintaining I/O Request Queues. To maximize throughput, the controller orders its read and write request queues by sector, head, drive, and cylinder. (Cylinders appear last in the order of sorting because a seek on one drive may be overlapped with a read or write on another.) The Am29116 maintains the read/write request queue in its 4096 x 16 high-speed buffer memory.

Selective sorting of the read/write request queue is performed by the controller. Each new request is assigned a "bump count" of 0 when the controller receives it. The request is then placed into the queue at the position determined by the following:

1. Behind all requests whose bump counts equal N ("Queue 1")
2. Inserted in sorted order into the remaining queue of requests whose bump counts are less than N ("Queue 2") as follows:
   (a) By type (read after write)
   (b) By sector number
   (c) By head number
   (d) By drive number
   (e) Finally, by cylinder number

3. Before each new request is queued, Queue 2 is scanned head-to-tail. Each request encountered during the scan that has a bump count of N is removed from Queue 2 and placed at the end of Queue 1.

4. After each new request is queued, the bump count is increased by 1 for each Queue 2 member that has been bumped by it (i.e., now follows it).

It should be noted that the choice of N is application-dependent, since increasing N increases throughput but also lengthens response time for some read/write requests.

2. Avoiding Redundant Reads. The Am29116 also maintains copies of the last eight sectors read from or written to disk. Before each read request is entered into the queue, the Am29116 compares it with
a list of buffer memory-resident sector images. If a match is found, the contents of the sector images are used to satisfy the read request and no enqueueing is performed.

3. Performing Logical Record I/O and Maintaining Device Transparency. The Am29116 translates I/O requests by logical record number into physical select, seek, and I/O operations by drive, track, head, and sector numbers. The CPU software need not concern itself with the characteristics of the particular drives attached, and it is minimally affected by deletions and additions of drives of varying types.

4. Performing Associative Logical Record I/O. The Am29116 reads, writes, or returns the logical record numbers of logical records that contain specified fields. The CPU software merely specifies the type of operation to be performed and the length, relative position within the logical record, and value of the content-addressing field.

5. Performing Data Compression and Expansion. Much of the information routinely stored on disk as 8-bit bytes is character data. While it is convenient to manipulate these data in the central processor in 8-bit EBCDIC notation, they can usually be stored much more efficiently on disk as either 6-bit BCD (or FIELDATA) bytes or 7-bit ASCII bytes. The usefulness of compressing information in this manner depends entirely upon the database. For example, most accounting and management information system data do not involve lower-case alphabatics and can be recorded in 6-bit BCD (or FIELDATA), giving approximately a 25% reduction in disk storage occupied and a 33% increase in storage effectiveness. Most word processing data involve lower-case alphabatics but can be recorded in 7-bit ASCII, giving approximately a 12.5% reduction in disk storage occupied and a 14.3% increase in storage effectiveness. The recording of data compressed in this manner is accomplished by a translation from EBCDIC to BCD/FIELDATA or ASCII followed by packing and an unformatted write operation. Compressed data are read by an unformatted read operation followed by unpacking and a translation from BCD/FIELDATA or ASCII to EBCDIC. The translations are performed two bytes at a time by the two 2048 x 8 Am27S291 PROMs illustrated in Figure 8. The three microcode bits labelled XLAT2 -XLATO select one of eight code translations; four are used by the BCD/FIELDATA and ASCII compression algorithms and four are spares.

Many other types of application-dependent data compression can be performed directly by the controller. The following IBM VM/370 CMS commands perform various types of compression depending upon the old file type:

(1) COPY ,old file name. ,old file type. ,old file mode. ,new file name. ,new file type. ,new file mode. (REP PACK)
All the functions of COPY (PACK) and COPY (UNPACK) can be performed by the Am29116 and Am9520-based controller. The controller allows packed files to be read and written as if they were unpacked, just as it allows 6-bit BCD/FIELDATA and 7-bit ASCII files to be read and written as if they were 8-bit EBCDIC files.

**System Organization**

Figure 4 is an overall block diagram of the disk controller. The interface to the drives includes separate bit-serial data paths for read data and write data, and byte-parallel paths for commands, disk addresses, and disk status as described in the current ANSI standard. The Am2910 microsequencer and 1K x 8 Am27S35 registered microprogram memory drive the 80-bit control bus that directs the actions of the other components. Data flows serially and asynchronously at over 20 MegaBits per second between the drives and the time-division multiplexed serial input/serial output ports of the 16 x 16 FIFO array. Data flows synchronously in 16-bit parallel form between the FIFO array and the 4K x 16 Am9147 buffer memory. In this design, it is assumed that support of disk transfer rates of close to 30Mbit/sec. is desirable. This is why the burst error processor, which can handle data up to 20Mbit/sec, is placed in parallel with the first-in-first-out memory array and the Am9147 RAM buffer*. During

*AMD now offers the Am9520-1, a 30Mbit/sec part which will simplify the microcode shown in the application note.

![Figure 4. Block Diagram of the Am29116/Am9520 Disk Controller](image)
disk reads, the Am29116 maintains two pointers: a write pointer for transferring data from the FIFO array to the buffer memory at a rate close to 30MHz, and a read pointer for concurrently transferring data from the buffer memory to the burst error processor at a rate equivalent to 15MHz. During disk writes, in which the timing of the checksum calculation is more critical, the transfers are not overlapped. If the data transfer rate needed is 20Mbit/sec or less in an alternative design, the burst error processor can be placed in line with the FIFO array. Table 2 lists the interface signals between the controller and up to eight drives.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PROSE SIGNAL NAME</th>
<th>SIGNAL SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMC</td>
<td>Address Mark Control</td>
<td>Controller</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>BCK</td>
<td>Bus Acknowledge</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>BOUT</td>
<td>Bus Direction Out</td>
<td>Controller</td>
</tr>
<tr>
<td>BUSY</td>
<td>Busy</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>CBPA</td>
<td>Control Bus Parity</td>
<td>Controller or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Selected Drive</td>
</tr>
<tr>
<td>CBDA0-7</td>
<td>Control Bus Data (multiplexed with SADR0-7)</td>
<td>Controller or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Selected Drive</td>
</tr>
<tr>
<td>CREQ</td>
<td>Command Request</td>
<td>Controller</td>
</tr>
<tr>
<td>INDX</td>
<td>Index</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>PENB</td>
<td>Port Enable</td>
<td>Controller</td>
</tr>
<tr>
<td>PREQ</td>
<td>Parameter Request</td>
<td>Controller</td>
</tr>
<tr>
<td>RDCM</td>
<td>Read/Reference Clock -</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>RDCP</td>
<td>Read/Reference Clock +</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>PROSE SIGNAL NAME</td>
<td>SIGNAL SOURCE</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>RDDM</td>
<td>Read Data -</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>RDDP</td>
<td>Read Data +</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>RDGA</td>
<td>Read Gate</td>
<td>Controller</td>
</tr>
<tr>
<td>SADR&lt;sub&gt;0-7&lt;/sub&gt;</td>
<td>Select/Attention Drive&lt;sub&gt;0-7&lt;/sub&gt; (multiplexed with CBDA&lt;sub&gt;0-7&lt;/sub&gt;)</td>
<td>Controller or Selected Drive</td>
</tr>
<tr>
<td>SAMD</td>
<td>Sector/Address Mark Detected</td>
<td>Selected Drive</td>
</tr>
<tr>
<td>SAST</td>
<td>Select/Attention Strobe</td>
<td>Controller</td>
</tr>
<tr>
<td>WRCM</td>
<td>Write Clock -</td>
<td>Controller</td>
</tr>
<tr>
<td>WRCP</td>
<td>Write Clock +</td>
<td>Controller</td>
</tr>
<tr>
<td>WRDM</td>
<td>Write Data -</td>
<td>Controller</td>
</tr>
<tr>
<td>WRDP</td>
<td>Write Data +</td>
<td>Controller</td>
</tr>
<tr>
<td>WRGA</td>
<td>Write Gate</td>
<td>Controller</td>
</tr>
</tbody>
</table>

The host CPU and memory interface is through either DMA or a host data channel, depending upon the host machine and application. Although the interface is not shown in detail, it can readily be implemented using the Am2940 DMA Address Generator and the Am2950 Parallel I/O Data Port.

Figure 5 depicts the byte-sync logic and timing logic for the FIFO buffer. It has been assumed here that the encoding scheme used by the drives is one that employs all-zero preambles (e.g., Modified Frequency Modulation). If 3PM or any other non-zero-preamble scheme is used, the byte-sync logic shown must be appropriately redesigned. Redesign of the byte-sync logic will also be necessary for drives that suppress transmission of part or all of the preamble.

Byte sync is achieved by three binary counters, which present and maintain a low output as soon as at least K zeroes followed by binary 11111110 (hexadecimal FE) have been encountered. The value of K may be "programmed" by means of the D, C, B, A
Figure 5. Byte-Sync and Timing Logic
inputs to U1 and U2. These inputs are shown tied to hexadecimal F7. Since \( FF_{16} - F7_{16} = 08_{16} = 08_{10} \), \( K = 8 \) for this instance. Higher values of \( K \) may render the detector unduly sensitive to phase locktime jitter and should be avoided. The bits first encountered during a sync burst are the least likely to be sampled correctly, since the drive's clock/data separator is still acquiring phase lock with the sync byte train. The optimal choice for \( K \) depends upon the acquisition rate and other characteristics of the clock/data separator.

Figure 6 depicts the serial-to-parallel and parallel-to-serial conversion interface using an array of 9403As operated in

![Diagram of serial parallel interfacing](image)
parallel at an aggregate rate of 30Mbit/sec per second. The FIFOs themselves are individually operated at 7.5Mbit/sec per second, and the 30Mb aggregate data rate is achieved by an alternate clocking scheme (Figure 7). This same scheme is used for both read and write clocking and that the FIFO serial input and output clocks, CPSI and CPSO, are falling-edge active. Pipelining is used to satisfy the setup time requirements of the FIFO serial inputs, DS. The FIFO serial outputs QS are also pipelined. However, the FIFO parallel inputs and outputs, D3-D0 and Q3-Q0, are fast enough to communicate with the buffer memory bus without pipelining.

The major elements of the remaining portion of the data path are the Am29116, the Am9520 and 4096 words of Am9147-55 buffer memory (Figure 8). These elements interface through an internal 16-bit data bus. The Am29116 is connected to this bus through two Am2949 bidirectional bus transceivers. During data compression operations, the read and write data are actually routed through two sets of Am27S291 translation PROMs. The Am29116 also generates and maintains the buffer memory addresses. The buffer memory comprises sixteen Am9147-55 4096 x 1 RAMs. It contains images of the last eight sectors read from or written to disk, the I/O request queues, and additional

Figure 7. FIFO Alternate Clocking
Figure 8. Processors and Buffer Memory
housekeeping tables. The 8-bit data input and output lines of the Am9520 are connected to the 16-bit internal data bus through a low and high byte bidirectional I/O port using two Am2950s. The instruction (C2-0) and read error pattern (REP) inputs of the Am9520 are generated by the Am29116 and are strobed into the command register under microprogram control. The Am9520 status signals—located error pattern (LP3-0) and pattern match (PM4-2)—are communicated to the Am29116 through the Am2959 buffer during high-speed error correction. In addition, the ANSI Control Bus Data (CBDA7-0) and the Select/Attention Drive (SADR7-0) signals to and from the selected drive are multiplexed and connected to the least significant byte of the internal data bus through an Am2949 bidirectional bus transceiver.

The Am2910 microprogram sequencer generates the next address to 1K words of control memory (Figure 9). The control memory is 80 bits wide and is configured using ten Am27S35 1024 x 8 registered PROMs. The test condition (CC) input to the Am2910 comes from one of sixteen sources (including a forced HIGH and a forced LOW) selected through multiplexers by five microinstruction bits. Except for the Am29116 CT status output, all of the test conditions are synchronized by the microinstruction clock (MICK) because they are from such asynchronous sources as the disk drives and the FIFO array.

**Microinstruction Format**
The format of the 80-bit microinstruction is outlined in Figure 10. The intent here is not to create a minimum-width, shared-field control word but to demonstrate microcoding the controller in a straightforward manner. Table 3 details the definition for each of the fields. A microinstruction word and field definition (DEF) file incorporating these is available to System 29 users.

Sample microcode has been written (and a source (SRC) file is available to System 29 users) for uncompressed sector read and write operations. The header and data segment format is shown in Figure 11. The code includes header and sector acquisition, error checking of the header (via CRC), and error checking and correction of the data segments (via the Am9520 and its 56-bit modified Fire code polynomial) (Figure 12).

The sector input/output microroutine (SECTIO) performs input or output of a single 256-byte sector. Seek and retry operations are the responsibility of the calling microprogram.

At entry to SECTIO, RO contains 0 to request a sector read, or +1 to request a sector write. R1 contains the I/O head number in its upper byte. The I/O track number is split between the lower byte of R1 and the upper byte of R2, while the lower byte of R2 contains the I/O sector number. R3 contains the buffer memory start address.

SECTIO first checks to see whether (RO) = +1 and, if so, uses the Am9520 to calculate the 56-bit modified Fire Code check bits that are to be appended during write. The check bits are stored in buffer memory immediately following the data.
Figure 9. Microinstruction Sequencing
### Figure 10. Microinstruction Format

#### TABLE 3. MICROINSTRUCTION FIELDS

<table>
<thead>
<tr>
<th>MICROINSTRUCTION FIELD</th>
<th>MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITs WIDTH</td>
<td>MNEMONIC</td>
</tr>
<tr>
<td>(BITs)</td>
<td>Am29116 Instruction</td>
</tr>
<tr>
<td>79-64</td>
<td>Am29116 Conditional Test Select</td>
</tr>
<tr>
<td>63-60</td>
<td>Am29116 Status Register Enable</td>
</tr>
<tr>
<td>58</td>
<td>Am29116 Output Enable Y-Bus</td>
</tr>
<tr>
<td>57</td>
<td>Am29116 Instruction Enable</td>
</tr>
<tr>
<td>56</td>
<td>Am29116 Data Latch Enable</td>
</tr>
<tr>
<td>55-52</td>
<td>Am2910 Instruction</td>
</tr>
<tr>
<td>51-42</td>
<td>Am2910 Direct Input</td>
</tr>
<tr>
<td>41-36</td>
<td>Test Multiplexer Condition and True/False Select</td>
</tr>
<tr>
<td>35</td>
<td>Address Mark Control (Table 1)</td>
</tr>
<tr>
<td>34</td>
<td>(Enable Memory) Bus From (Disk Drive) Control Bus</td>
</tr>
<tr>
<td>33</td>
<td>(Enable Memory) Bus From Translate PROM</td>
</tr>
<tr>
<td>32</td>
<td>(Enable Memory) Bus From 9403A FIFO Array</td>
</tr>
<tr>
<td>31</td>
<td>(Enable Memory) Bus From Am29116 Y-Bus</td>
</tr>
<tr>
<td>30</td>
<td>(Enable Memory) Bus Lower Byte From Am9520 Q-Bus</td>
</tr>
<tr>
<td>29</td>
<td>(Enable Memory) Bus Upper Byte From Am9520 Q-Bus</td>
</tr>
<tr>
<td>28</td>
<td>Bus Direction OUT (Table 1)</td>
</tr>
<tr>
<td>27</td>
<td>(Enable Memory) Bus To 9403A FIFO Array</td>
</tr>
<tr>
<td>26</td>
<td>(Enable Memory) Bus To Am29116 Y-Bus</td>
</tr>
<tr>
<td>25</td>
<td>(Enable Memory) Bus Lower Byte To Am9520 D-Bus</td>
</tr>
<tr>
<td>24</td>
<td>(Enable Memory) Bus Upper Byte To Am9520 D-Bus</td>
</tr>
<tr>
<td>23</td>
<td>(Enable Memory) Bus To Am9520 REP, P3-P0, &amp; C2-C0</td>
</tr>
<tr>
<td>22</td>
<td>Clock Enable Am9520 To Lower-Byte Bus Interface Register</td>
</tr>
<tr>
<td>21</td>
<td>Clock Enable Memory Bus To Am9520 Interface Registers</td>
</tr>
<tr>
<td>20</td>
<td>Clock Pulse For Am9520 (Microcoded Waveform)</td>
</tr>
</tbody>
</table>
### TABLE 3. MICROINSTRUCTION FIELDS (Cont.)

<table>
<thead>
<tr>
<th>MICROINSTRUCTION FIELD</th>
<th>MNEMONIC</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>CREQ</td>
<td>Command Request (Table 1)</td>
</tr>
<tr>
<td>18</td>
<td>INPT</td>
<td>(Enable Serial Data) Input To 9403A FIFO Array</td>
</tr>
<tr>
<td>17-16</td>
<td>JMPI</td>
<td>(Enable) Jump Indirect Am29116 Y-Bus (Double-Rail)</td>
</tr>
<tr>
<td>15</td>
<td>MADR</td>
<td>(Enable Loading Of Buffer) Memory Address Register</td>
</tr>
<tr>
<td>14</td>
<td>MREA</td>
<td>(Enable Buffer) Memory Read</td>
</tr>
<tr>
<td>13</td>
<td>MWRT</td>
<td>(Enable) Memory Write Operation</td>
</tr>
<tr>
<td>12</td>
<td>OUPT</td>
<td>(Enable Serial Data) Output From 9403A FIFO Array</td>
</tr>
<tr>
<td>11</td>
<td>PENB</td>
<td>Parameter Enable (Table 1)</td>
</tr>
<tr>
<td>10</td>
<td>PFPM</td>
<td>(Enable Setting Of Am9520) P Bits From Am9520 PM Bits</td>
</tr>
<tr>
<td>09</td>
<td>PF03</td>
<td>(Enable) Parallel Fetch From 9403A FIFO Array</td>
</tr>
<tr>
<td>08</td>
<td>PLO3</td>
<td>(Enable) Parallel Load Of 9403A FIFO Array</td>
</tr>
<tr>
<td>07</td>
<td>PREQ</td>
<td>Parameter Request (Table 1)</td>
</tr>
<tr>
<td>06</td>
<td>RDGA</td>
<td>Read Gate (Table 1)</td>
</tr>
<tr>
<td>05</td>
<td>RTIF</td>
<td>Reset 9403A FIFO Array</td>
</tr>
<tr>
<td>04</td>
<td>SAST</td>
<td>Select/Attention Strobe (Table 1)</td>
</tr>
<tr>
<td>03</td>
<td>WRGA</td>
<td>Write Gate (Table 1)</td>
</tr>
<tr>
<td>02-0</td>
<td>XLAT</td>
<td>Translate Table Select For Data Compression PROM</td>
</tr>
</tbody>
</table>

![Header and Data Segments](image)

**Figure 11. Header and Data Segments**

SECTIO then (both for reads and for writes) uses the Am29116 to calculate the CRC of the header contained in R1 and R2. This CRC is saved in R4.

A search is then made of the entire track for a header whose head, track, sector, and CRC fields match the contents of R1, R2, and R4. If the search fails, RO is loaded with +1 (defective or missing header) and control is returned to the calling microprogram. If the search is successful, control is passed (via (RO) and table BRTABL) to either the read sector (RDSEC1) or the write sector (WRSEC1) microcode module.
Read Sector Microcode Module (RDSECl)

This module transfers synchronized information, a 16-bit word at a time, from the 9403A FIFO array to buffer memory and from buffer memory to the Am9520 operating in Read High-Speed mode. Since the current Am9520 data sheet only guarantees operation at 20MHz, some form of buffering must be used between the 30MHz disk and the Am9520. This is accomplished by using R4 as a memory buffer pointer for transfer in from the 9403A's and R5 as a pointer for transfer out to the Am9520. For simplicity in the microcode loop, R5 increments at half (rather than two-thirds) the rate at which R4 increments.

At the end of the read loop, R5 has advanced halfway through the data read in and a second loop is executed to process the remaining half of the data through the Am9520.

When all the data have been processed by the Am9520 Read High-Speed operation, the Am9520 error (ER) flag is tested to determine whether an error was detected. If ER is low (no error), RO is loaded with 0 (operation completed successfully) and control is returned to the calling program.

If ER is high, error correction is performed using the Am9520's correct high speed mode. This uses the Chinese Remainder Theorem method to calculate the error location (as a bit displacement from the end of the data segment) and error pattern (a 12-bit mask). The error is corrected by exclusive or-ing the error pattern with the 12-bit data field beginning at the error location. The capabilities of the Am9520 and the properties of the 56-bit modified Fire Code polynomial make this correction technique extremely fast. Less than 200 microseconds are required for a worst-case error location and correction using the microcode shown.

The location of an error burst is calculated by:

\[ L = N \times K - (M_1 \times A_1 + M_2 \times A_2 + M_3 \times A_3 + M_4 \times A_4) \]

where:

- \( L \) is the difference in position between the last bit transferred and the beginning of the burst error.
- \( N \) is the composite period of the 56-bit polynomial and is equal to 585,442.
- \( K \) is the smallest integer such that \( L \) is positive.
- \( A_1, A_2, A_3, \) and \( A_4 \) are Chinese Remainder Theorem coefficients:
  - \( A_1 = 452,387 \)
  - \( A_2 = 2,521,904 \)
  - \( A_3 = 578,864 \)
  - \( A_4 = 2,647,216 \)

- \( M_1, M_2, M_3, \) and \( M_4 \) are factor match clock counts that are accumulated by the microcode while clocking the Am9520 in Correct High-Speed mode. For burst errors of length not exceeding 11 bits, it can be shown that \( M_1 \) will never exceed 22 (the period of the first factor of the 56-bit polynomial); \( M_2 \) will never exceed 13 (period of the second
factor); \( M_3 \) will never exceed 89 (period of the third factor); and \( M_4 \) will never exceed 23 (period of the fourth factor).

Consequently, the maximum number of Am9520 clock cycles needed to locate an 11-bit (or shorter) error burst is the sum of the first period and the maximum of the remaining three periods:

\[
22 + \text{MAX}(13, 89, 23) = 22 + 89 = 111
\]

It should be noted that the above number of Am9520 clock cycles is far less than the composite period, 585,442, which is the upper limit for correct normal operations and is representative of how long a less sophisticated part would require to locate and correct the error burst.

To perform error location and correction, the Am9520 is placed in correct high-speed mode and its clock enable PO is set high for factor match clock count \( M_1 \) accumulation. R8 is initialized to 0 to serve as the \( M_1 \) counter. PF \(_1\) (the maximum permissible value for \( M_1 \), which will be exceeded only for multiple bursts or bursts longer than 11 bits) is loaded into the accumulator (ACC). The EP output is tested. If EP is low, alignment exception (AE) is tested while the ACC is decremented and the Am9520 is clocked. If AE is high, the burst error is not on a byte boundary and R8 is incremented by 1. If AE is low, R8 is incremented by 8. The ACC is now tested. If positive, PFI is not exceeded and a loop back to the EP test is performed. If negative, an uncorrectable error exists; RO is set to +2; and control is returned to the calling microprogram. If EP is high, the \( M_1 \) calculation is complete; the error pattern is available; and \( M_2 \) through \( M_4 \) can now be accumulated.

The inherent parallelism of the Am9520 is then exploited by concurrently accumulating \( M_2 \) through \( M_4 \). This reduces the number of Am9520 clocks required from the sum of the three periods (125) to their maximum (89). R9 through R11 serve as the counters for \( M_2 \) through \( M_4 \). The microprogram flow of control reflects the completeness or incompleteness of each factor match by looping through a jump table indexed by the Am9520 Pattern Match (PM\(_2\) through PM\(_4\)) outputs, and by selectively disabling the \( P_1 \) through \( P_3 \) clock enables with the same PM\(_2\) through PM\(_4\) outputs. This yields eight possible paths (Figure 12), in each of which the appropriate combination of R9 through R11 can be operated upon and tested to see if it exceeds period factor limits (i.e., a multiple-burst error or an error burst longer than 11 bits has been encountered).

Once \( M_1 \) through \( M_4 \) have been obtained, the expression:

\[
(M_1 \times A_1 + M_2 \times A_2 + M_3 \times A_3 + M_4 \times A_4)
\]

is evaluated by calling a specialized multiply subroutine (MUL) four times. This subroutine utilizes the special nature both of the period factor values and of the Chinese Remainder Theorem coefficients to maximize throughput. A specially optimized divide subroutine (DIV) is then called to calculate:

\[
(M_1 \times A_1 + M_2 \times A_2 + M_3 \times A_3 + M_4 \times A_4) / N
\]
leaving a remainder of \((-L + N)\). One additional subtract obtains \(L^*\).

The word-boundary address of the error burst in buffer memory is extracted from \(L\) using the Am29116 Rotate and Merge instruction. A 16-way branch on the low-order 4 bits of \(L\) is used to enter a table (TAB2) of Rotate and Merge instructions.

These align the error pattern (using a single ROTM instruction if the error burst does not cross a word boundary and two instructions if it does). The error burst is then exclusive OR-ed with the aligned error pattern; RO is loaded with 0 (operation completed successfully); and control is returned to the calling microprogram.

**Write Sector Microcode Module (WRSEC1)**

This module transfers information one 16-bit word at a time to the 9403A FIFO array. The information transferred comprises a data preamble (13 all-zero bytes), data sync byte (hexadecimal FE), 256 data bytes, 7 check bytes, and a data postamble (5 all-zero bytes). Both the data bytes and the check bytes are located in buffer memory, beginning at word (R3). (Calculation of the check bytes has already occurred at the beginning of SECTIO).

RO is loaded with 0 (operation completed successfully) and control is returned to the calling program.

**Conclusion**

The high-speed and parallel architecture of the Am29116 and Am9520-based controller allows handling of high data transfer rate disk drives and complex data manipulation and management. The availability of cost-effective microprogrammable building blocks in the Am2900 Family has led to systems with increasingly distributed control. This allows functions to be performed at system locations that optimize overall cost/performance.

Significant improvements in host computer system performance can be realized by downloading many time-consuming operating system tasks into the controller firmware. This allows mainstream processing of the application programs to proceed with minimal I/O overhead. System response is enhanced and main storage usage, software requirements and system overhead are reduced.

* The method used here to obtain the error location is not the only one possible. One alternative is to subtract some form of the Chinese Remainder Theorem coefficients iteratively instead of multiplying and dividing. With each subtraction \(L\) would be tested. If negative, \(N\) would be added to \(L\). This approach still exploits the parallel nature of the Am9520.
This .DEF file (DISKCTLR.DEF) was created by editing CONTROLR.DEF; by adding DEF and EQU statements, deleting some others, and by changing the basic microword format. The bulk of the effort required to create such a file was considerably reduced by beginning from the "master" file (CONTROLR.DEF) rather than typing a new file from scratch.

This particular .DEF file was created for a specific Am29116-Am9520 disk controller, described in the AMD application note: "A High-Performance Intelligent Disk Controller," by Otis Tabler and Brad Kitson, to be released by AMD in early 1982. The source file is DISKCTLR.SRC.

The major difference between this DEF file and the CONTROLR.DEF file is the approach to the microprogramming. This file makes heavy use of DEF statement overlays while the other uses the comma-positional notation. The choice is a matter of preference. The Am29116 mnemonics and instruction layout are identical in these files.

This file may also be used as a master file which the user can edit to suit his/her application.

Anyone finding an error in this file is requested to send a marked listing or portion thereof to: AMD APPLICATIONS or AMD CUSTOMER EDUCATION CENTER

PO BOX 453 MS#70
PO BOX 453 MS#71
SUNNYVALE, CA 94086
490-A LAKESIDE DRIVE
SUNNYVALE, CA 94086

Advanced Micro Devices reserves the right to make changes in its product without notice in order to improve design or performance characteristics. The company assumes no responsibility for the use of any circuits or programs described herein.

Am29116 Mnemonics Copyright © 1982 Advanced Micro Devices, Inc.

General Mnemonics

** WORD 80 **

GENERAL MNEMONICS

***********

BYTE - WORD MODE SELECT [M] <-------- referenced by DEF statements

B: EQU 1B#0 ; BYTE MODE
W: EQU 1B#1 ; WORD MODE

***********

N SELECT [N]

N0: EQU H#0 ; 0
N1: EQU H#1 ;
N2: EQU H#2 ;
N3: EQU H#3 ;
N4: EQU H#4 ;
N5: EQU H#5 ;
N6: EQU H#6 ;
N7: EQU H#7 ;
N8: EQU H#8 ;
N9: EQU H#9 ;
NA: EQU H#A ;
ND: EQU H#D ;
NC: EQU H#C ;
NE: EQU H#E ;
NF: EQU H#F ;
32 RAM REGISTERS (R)

RO: EQU 5D#0 00000
R1: EQU 5DU
R2: EQU 5DI2
R3: EQU 5D#3
R4: EQU 5D#4
R5: EQU 5DI5
R6: EQU 5D#6
R7: EQU 5D#7
R8: EQU 5Di8
R9: EQU 5D#9
R10: EQU 5DI10
R11: EQU 5D#11
R12: EQU 5D#12
R13: EQU 5DU3
R14: EQU 5D#14
R15: EQU 5DIl5
R16: EQU 5D#16
R17: EQU 5DU7
R18: EQU 5D#18
R19: EQU 5DI19
R20: EQU 5DI20
R21: EQU 5D,21
R22: EQU 5D,22
R23: EQU 5D#23
R24: EQU 5DI24
R25: EQU 5DI25
R26: EQU 5DI26
R27: EQU 5DI27
R28: EQU 5D#28
R29: EQU 5DI29
R30: EQU 5D130
R31: EQU 5D#31

SINGLE OPERAND INSTRUCTIONS

OPCODES

MOVE: EQU HiC 1100
COMPo EQU HiD 1101
INC: EQU H#E 1110 INC INCREMENT
NEG: EQU H#F 1111 NEG INCREMENT

SOURCE-DESTINATION SELECT

SORA: EQU H#O RAM ACC
SORY: EQU H#2 RAM Y BUS
SORS: EQU H#3 RAM STATUS
SOAR: EQU H#4 ACC RAM
SODR: EQU H#6 D RAM
SOIR: EQU H#7 I RAM
SOZR: EQU H#8 0 RAM
SOZER: EQU H#9 D(OE) RAM
SOSER: EQU H#A D(SE) RAM
SORR: EQU H#B RAM RAM

SOURCE-DESTINATION REGISTER MODE QUAD_OPCODE SOURCE-DEST_REGISTER

- Refer to Proper EQU groups
SOURCE (R/S) [3]

SOA:  EQU  H#4  ;  ACC
SOO:  EQU  H#6  ;  D
SOS:  EQU  H#7  ;  I
SOE:  EQU  H#8  ;  0
SOE#:  EQU  H#9  ;  D(0E)
SOE#:  EQU  H#A  ;  D(0E)

;  DESTINATION [4]
;
NY:  EQU  DB0  ;  Y BUS
NYA:  EQU  DB1  ;  ACC
NYS:  EQU  DB4  ;  STATUS
NYAS:  EQU  DB5  ;  ACC,STATUS
;
;  *****************************************************
;  SINGLE OPERAND NON-RAM
;  ******************************************************************************
;
TORRAA:  EQU  H#0  ;  RAM ACC ACC
TORI:  EQU  H#2  ;  RAM I  ACC
TORIA:  EQU  H#3  ;  D RAM ACC
TORIY:  EQU  H#4  ;  RAM I  Y BUS
TORIR:  EQU  H#6  ;  D RAM Y BUS
TORIR:  EQU  H#C  ;  RAM ACC RAM
TORIR:  EQU  H#E  ;  RAM I  RAM
TORIR:  EQU  H#F  ;  D RAM RAM

;  *****************************************************
;  TOlll:  DEF IV,  B#00,4V&D#,  4V&D#,  5V&D#,64X ;  TWO OPERAND RAM [1]
;  ******************************************************************************
;  ******************************************************************************
SOURCE-DESTINATION [7]  R  S  DEST

TODA:  EQU  H#1 ; D  ACC  RAM
TODAI: EQU  H#2 ; ACC  I  RAM
TODIR: EQU  H#5 ; D  I  RAM

TODR: DEF IV, 8#10,4V#D$, 4V#D$, 5V#D$, 64X ; TWO OPERAND RAM (2)

SOURCE [8]  R  S

TODA:  EQU  H#1 ; D  ACC
TODAI: EQU  H#2 ; ACC  I
TODIR: EQU  H#5 ; D  I

TODR: DEF IV, 8#11,4V#D$, 4V#D$, 5V#D$, 64X ; TWO OPERAND NON-RAM

MODE, QUAD, SOURCE-DST, OPCODE, REGISTER

SOURCE [9]  R  S

TODA:  EQU  H#1 ; D  ACC
TODAI: EQU  H#2 ; ACC  I
TODIR: EQU  H#5 ; D  I

TODR: DEF IV, 8#11,4V#D$, 4V#D$, 5V#D$, 64X ; TWO OPERAND NON-RAM

MODE, QUAD, SOURCE, OPCODE, DESTINATION

SHIFT INSTRUCTIONS

DIRECTION AND INPUT [9]

SH-Up:  EQU  H#0 ; UP 0
SH-UPL: EQU  H#1 ; UP 1
SH-UPL: EQU  H#2 ; UP QLINK
SH-On:  EQU  H#4 ; DOWN 0
SH-ONL: EQU  H#5 ; DOWN 1
SH-ONL: EQU  H#6 ; DOWN QLINK
SH-ONC: EQU  H#7 ; DOWN QC
SH-ONOV: EQU  H#8 ; DOWN QC QN

SOURCE [10]

SHRR:  EQU  H#6 ; RAM  RAM
SHRD:  EQU  H#7 ; D  RAM

SHIFT: DEF IV, 8#10,4V#D$, 4V#D$, 5V#D$, 64X ; SHIFT RAM

MODE, QUAD, SOURCE, DIRECT-INPUT, REGISTER

SOURCE [11]

SHA:  EQU  H#6 ; ACC
SHD:  EQU  H#7 ; D

SHIFT: DEF IV, 8#11,4V#D$, 4V#D$, 5V#D$, 64X ; SHIFT NON-RAM

MODE, QUAD, SOURCE, DIRECT-INPUT, DESTINATION

; ROTATE INSTRUCTIONS
; *********************************************
; SOURCE-DESTINATION [12]
;
RTRA:  EQU  H$C  ; RAM  ACC
RTRY:  EQU  H$E  ; RAM  Y BUS
RTTR:  EQU  H$F  ; RAM  RAM

; *********************************************
RTAR1:  DEF  IV,  B$00,4V$D$,4V$D$,  5V$D$,64X  ; ROTATE RAM (1)
;   MODE,QUAD,N,SOURCE-DEST,REGISTER
   [M] [N] [12] [R]
; *********************************************
RTAR:  EQU  H$0  ; ACC  RAM
RTDR:  EQU  H$1  ; D  RAM

; *********************************************
RTAR2:  DEF  IV,  B$01,4V$D$,4V$D$,  5V$D$,64X  ; ROTATE RAM (2)
;   MODE,QUAD,N,SOURCE-DEST,REGISTER
   [M] [N] [13] [R]
; *********************************************
RTDY:  EQU  D$24  ; D  Y BUS
RTDA:  EQU  D$25  ; D  ACC
RTAY:  EQU  D$28  ; ACC  Y BUS
RTAA:  EQU  D$29  ; ACC  ACC

; *********************************************
ROTR:  DEF  IV,  B$11,4V$D$,H$C,  5V$D$,64X  ; ROTATE NON-RAM
;   MODE,QUAD,N,FIXED CODE,DESTINATION
   [M] [N] [14]
; *********************************************
; ***************************************************
; BIT ORIENTED INSTRUCTIONS
; ***************************************************
;
; OPCODES [15]
;
; SETNR:   EQU H#D   ; SET RAM, BIT N
RSTNR:   EQU H#E   ; RESET RAM, BIT N
TSTNR:   EQU H#F   ; TEST RAM, BIT N
;
; ***************************************************
; BOR1: DEF IV, B$11,4VAD$,4VAD$, 5VAD$,64X ; BIT ORIENTED RAM (1)
;
; MODE,QUAD,N,OPCODE,REGISTER
; [M] [N] [15] [R]
;
; ***************************************************
;
; OPCODES [16]
;
; LD2NR:   EQU H#C   ; 2"N --- RAM
LDC2NR:   EQU H#D   ; 2"N --- RAM
A2NR:     EQU H#E   ; RAM + 2"N - RAM
S2NR:     EQU H#F   ; RAM - 2"N - RAM
;
; ***************************************************
;
; BOR2: DEF IV, B$10,4VAD$,4VAD$, 5VAD$,64X ; BIT ORIENTED RAM (2)
;
; MODE,QUAD,N,OPCODE,REGISTER
; [M] [N] [16] [R]
;
; ***************************************************
;
; OPCODES [17]
;
; TSTNA:   EQU D#0   ; TEST ACC, BIT N
RSTNA:   EQU D#1   ; RESET ACC, BIT N
SETNA:   EQU D#2   ; SET ACC, BIT N
A2NA:     EQU D#3   ; ACC + 2"N -- ACC
S2NA:     EQU D#5   ; ACC - 2"N -- ACC
LD2NA:   EQU H#6   ; 2"N -- ACC
LDC2NA:   EQU D#7   ; 2"N -- ACC
TSTND:   EQU D#16  ; TEST D, BIT N
RSTND:   EQU D#17  ; RESET D, BIT N
SETND:   EQU D#18  ; SET D, BIT N
A2NDY:    EQU D#20  ; D + 2"N -- Y BUS
S2NDY:    EQU D#21  ; D - 2"N -- Y BUS
LD2NY:    EQU D#22  ; 2"N -- Y BUS
LDC2NY:   EQU D#23  ; 2"N -- Y BUS
;
; ***************************************************
;
; BONR: DEF IV, B$11,4VAD$,B$1100, 5VAD$,64X ; BIT ORIENTED NON-RAM
;
; MODE,QUAD,N,FIXED CODE,OPCODE
; [M] [N] [17]
; ***************************************************
; **********************************************
; ROTATE AND MERGE
; **********************************************
; SOURCE-DEST SELECT [U,S,MASK-DEST] [18]
; 
; MDAR: EQU H#B ; D  ACC RAM
; MDRI: EQU H#A ; D  RAM  I
; MRA:  EQU H#E ; RAM  ACC I
; 
; ****************************************************
; ROTM: DEF IV, B#01,4V#D#, 4V#D#, 5V#D#,64X ; ROTATE AND MERGE
; [M] [N] [18] [R]
; *****************************************************
; **************************************************************************
; ROTATE AND COMPARE
; **************************************************************************
; Rotate and Source [19]
; 
; CDAI:  EQU H#2 ; D  ACC I
; CDRI:  EQU H#3 ; D  RAM I
; CDRA:  EQU H#4 ; D  RAM ACC
; CRI:   EQU H#5 ; RAM  ACC I
; 
; **************************************************************************
; ROTC: DEF IV, B#01,4V#D#, 4V#D#, 5V#D#,64X ; ROTATE AND COMPARE
; [M] [N] [19] [R]
; **************************************************************************
; PRIORITIZE
; **************************************************************************
; 
; SOURCE [20]
; 
; PRT1A: EQU H#7 ; ACC
; PRT1D: EQU H#9 ; D
; 
; DESTINATION [21]
; 
; PRA:  EQU H#8 ; ACC
; PRY:  EQU H#A ; Y BUS
; PRIR: EQU H#B ; RAM
; 
; **************************************************************************
; PRT1: DEF IV, B#10,4V#D#, 4V#D#, 5V#D#,64X ; RAM ADDR MASK[S]
; [M] [21] [20] [R]
; **************************************************************************
; 
; DESTINATION [23]
; 
; PRA:  EQU H#8 ; ACC
; PRA:  EQU H#A ; 0
; PRI:   EQU H#B ; I
; 
; **************************************************************************
; PRT2: DEF IV, B#10,4V#D#, 4V#D#, 5V#D#,64X ; PRIORITIZE RAM
; [M] [22] [23] [R]
; **************************************************************************

30
; SOURCE (R) [24]
; PR3B: EQU H$3 ; RAM
PR3A: EQU H$4 ; ACC
PR3D: EQU H$6 ; D
;
; ***********************************************
PR3R: DEF IV, B$10,4V#D$, 4V#D$, 5V#D$,64X ; PRIORITIZE RAM
; MODE,QUAD,MASK,SOURCE,REG-DEST
[ M ] [ 22 ] [ 24 ] [ R ]
; ***********************************************
; SOURCE (R) [25]
PRTA: EQU H$4 ; ACC
PRTD: EQU H$6 ; D
;
; ***********************************************
PRTNR: DEF IV, B$11,4V#D#, 4V#D#, 5V#D#,64X ; PRIORITIZE NON-RAM
; MODE,QUAD,MASK,SOURCE,DESTINATION
[ M ] [ 22 ] [ 25 ] [ 4 ] (NR,NRA ONLY)
; ***********************************************
; CYCLIC REDUNDANCY CHECK
; ***********************************************
; ***********************************************
CRCF: DEF B$11001100011,5V#D#,64X ; FORWARD
; ***********************************************
CRCR: DEF B$11001101001,5V#D#,64X ; REVERSE
; ***********************************************
NOOP
; ***********************************************
NOOP: DEF H$7146,64X ; NO OPERATION
; ***********************************************
; ***************************************************************
; STATUS
; ***************************************************************
; OPCODE [26]
;
SONZC: EQU 5D#3 ; SET OVR,N,C,Z
SL: EQU 5D#5 ; SET LINK
SF1: EQU 5D#6 ; SET FLAG 1
SF2: EQU 5D#9 ; SET FLAG 2
SF3: EQU 5D#10 ; SET FLAG 3
;
; ***************************************************************
SETST: DEF B#011,H#BA,5V#D$,64X ; SET STATUS
;
; OPCODE [26]
; ***************************************************************
;
; OPCODE [27]
;
RONCZ: EQU D#3 ; RESET OVR,N,C,Z
RL: EQU D#5 ; RESET LINK
RFI: EQU D#6 ; RESET FLAG 1
RF2: EQU D#9 ; RESET FLAG 2
RF3: EQU D#10 ; RESET FLAG 3
;
; ***************************************************************
RSTST: DEF B#011,H#AA,5V#D$,64X ; RESET STATUS
;
; OPCODE [27]
; ***************************************************************
;
; ***************************************************************
SVSTR: DEF 1V, B#10,H#7A, 5V#D$,64X ; SAVE STATUS-RAM
;
; \ MODE,QUAD,FIXED, RAM ADDRESS/DEST
; [M] [R]
; ***************************************************************
;
; ***************************************************************
SVSTNR: DEF 1V, B#11,H#7A, 5V#D$,64X ; SAVE STATUS NON-RAM
;
; \ MODE,QUAD,FIXED, DESTINATION
; [M] [4](NRY,NRA ONLY)
; ***************************************************************
TEST STATUS
 *****************************************************

OPCODE (CT)

TNOZ: EQU D#0 ; TEST (N OVR) + Z
TNO: EQU D#2 ; TEST N OVR
TZ: EQU D#4 ; TEST Z
TOVR: EQU D#6 ; TEST OVR
TLOW: EQU D#8 ; TEST LOW
TC: EQU D#10 ; TEST C
TNC: EQU D#12 ; TEST Z + C
TN: EQU D#14 ; TEST N
TFL: EQU D#16 ; TEST LINK
TF1: EQU D#18 ; TEST FLAG 1
TF2: EQU D#20 ; TEST FLAG 2
TF3: EQU D#22 ; TEST FLAG 3

****************************************************

TEST: DEF B#011, H#9A, 5V#D#, 64X ; TEST STATUS
FIXED, OPCODE [CT]

 ****************************************************

; added DEF and EQU statements

IMMEDIATE OPERAND

IMME: DEF 16V#D#, 64X

CT MULTIPLEXER CONTROL

CT: DEF 16X, 4V#D#, 60X
NOZ: EQU H#0
NO: EQU H#1
Z: EQU H#2
OVR: EQU H#3
LOW: EQU H#4
C: EQU H#5
ZC: EQU H#6
N: EQU H#7
L: EQU H#8
F1: EQU H#9
F2: EQU H#A
F3: EQU H#B

STATUS REGISTER ENABLE

SRE: DEF 20X, H#1, 59X
SORE: DEF 20X, H#0, 59X

OUTPUT ENABLE Y

OET: DEF 21X, H#0, 58X
NOEY: DEF 21X, H#1, 58X

INSTRUCTION ENABLE

IEN: DEF 22X, H#0, 57X
NOIEN: DEF 22X, H#1, 57X

D-1-LATCH ENABLE

DLE: DEF 23X, H#0, 56X
NODLE: DEF 23X, H#0, 56X
; Am2910 COMMANDS AND BRANCH ADDRESSES
; note use of DEF statements - overlay in SRC file

; JZ:      DEF 24X, H11, 10V5D#1023, 42X
JCS:     DEF 24X, H11, 10V5D#1023, 42X
JS:      DEF 24X, H11, 10V5D#1023, 6Q#36, 36X
JMAP:    DEF 24X, H12, 10V5D#1023, 42X
CJP:     DEF 24X, H13, 10V5D#1023, 42X
JP:      DEF 24X, H13, 10V5D#1023, 6Q#36, 36X
POSR:    DEF 24X, H14, 10V5D#1023, 42X
CJS:     DEF 24X, H14, 10V5D#1023, 42X
JS:      DEF 24X, H15, 10V5D#1023, 42X
JP:      DEF 24X, H15, 10V5D#1023, 6Q#36, 36X
UNCONDITIONAL JUMP TO SUBR.
CJVP:    DEF 24X, H16, 10V5D#1023, 42X
JVP:     DEF 24X, H17, 10V5D#1023, 42X
RPCT:    DEF 24X, H18, 10V5D#1023, 42X
CJPP:    DEF 24X, H19, 10V5D#1023, 42X
JP:      DEF 24X, H19, 10V5D#1023, 6Q#36, 36X
UNCONDITIONAL RETURN
CJPP:    DEF 24X, H20, 10V5D#1023, 42X
LDCT:    DEF 24X, H21, 10V5D#1023, 42X
CJSR:    DEF 24X, H22, 10V5D#1023, 42X
CONT:    DEF 24X, H23, 10V5D#1023, 42X
TWB:     DEF 24X, H24, 10V5D#1023, 42X

; NOTE: For proper assembly, a "$" must be used in any field which
; will be used to accept a symbolic address in the SRC file.

; Am2910 CONDITION CODE SELECTIONS
;
IF:      DEF 38X, 5V$D#, B#0, 36X
IFNOT:   DEF 38X, 5V$D#, B#1, 36X

AR20:    EQU 5Q#10: ; AM9520 ALIGNMENT ERROR FLAG
CT16:    EQU 5Q#11: ; AM29116 CONDITIONAL TEST FLAG
EP20:    EQU 5Q#12: ; AM9520 ERROR PATTERN FLAG
ER20:    EQU 5Q#13: ; AM9520 ERROR DETECTED FLAG
FAIL:    EQU 5Q#14: ; UNCONDITIONAL FAILURE OF "TEST"
RDY1:    EQU 5Q#15: ; NOT READY INPUT (DATA UNAVAILABLE FROM FIFO)
RDY0:    EQU 5Q#16: ; NOT READY OUTPUT (FIFO FULL)
SUCC:    EQU 5Q#17: ; UNCONDITIONAL SUCCESS OF "TEST"
ATTN:    EQU 5Q#20: ; ATTENTION
ACK:     EQU 5Q#21: ; BUS ACKNOWLEDGE
BUSY:    EQU 5Q#22: ; BUSY
INDEX:   EQU 5Q#23: ; INDEX
SAMD:    EQU 5Q#24: ; SECTOR / ADDRESS MARK DETECTED
PM2:     EQU 5Q#25: ; AM9520 PATTERN MATCH 2 FLAG
PM1:     EQU 5Q#26: ; AM9520 PATTERN MATCH 3 FLAG
PM4:     EQU 5Q#27; ; AM9520 PATTERN MATCH 4 FLAG
MISCELLANEOUS CONTROL SIGNALS

ADMC: DEF 44X, B$0, 35X ; ADDRESS MARK CONTROL
BFCB: DEF 45X, B$0, 34X ; MEMORY BUS FROM DRIVE CONTROL BUS
BFTP: DEF 46X, B$0, 33X ; MEMORY BUS FROM TRANSLATE FROM
BF03: DEF 47X, B$0, 32X ; MEMORY BUS FROM 9403AS
BF16: DEF 48X, B$0, 31X ; MEMORY BUS FROM AM29116
BFTP: DEF 49X, B$0, 30X ; MEMORY BUS FROM AM9520 - LOWER BYTE
BF2U: DEF 50X, B$0, 29X ; MEMORY BUS FROM AM9520 - UPPER BYTE
BOUT: DEF 51X, B$0, 28X ; (DISK) BUS DIRECTION OUT (FROM CONTROLLER)
BT03: DEF 52X, B$0, 27X ; MEMORY BUS TO 9403AS
BT16: DEF 53X, B$0, 26X ; MEMORY BUS TO AM29116
BT2L: DEF 54X, B$0, 25X ; MEMORY BUS TO AM9520 - LOWER BYTE
BT2U: DEF 55X, B$0, 24X ; MEMORY BUS TO AM9520 - UPPER BYTE
CE2L: DEF 56X, B$0, 23X ; MEMORY BUS TO AM9520 - CONTROL INFORMATION
CE2U: DEF 57X, B$0, 22X ; MEMORY BUS TO AM9520 - CONTROL INFORMATION
CP20: DEF 58X, B$0, 21X ; MEMORY BUS TO AM29116
CP2U: DEF 59X, B$0, 20X ; MEMORY BUS TO AM9520
CRQ: DEF 60X, B$0, 19X ; COMMAND REQUEST
INPT: DEF 61X, B$0, 18X ; INPUT SERIAL DATA TO 9403AS
JMPI: DEF 62X, B$0, 17X ; JUMP INDIRECT AM29116 REGISTER
NORMPI: DEF 63X, B$0, 16X ; NO INDIRECT JUMP
MADV: DEF 64X, B$0, 15X ; MEMORY ACCESS
MRAD: DEF 65X, B$0, 14X ; MEMORY ADDRESS
MMRT: DEF 66X, B$0, 13X ; MEMORY WRITE
OUPT: DEF 67X, B$0, 12X ; OUTPUT SERIAL DATA FROM 9403AS
PEAD: DEF 68X, B$0, 11X ; PARAMETER ENABLE
PFPM: DEF 69X, B$0, 10X ; SET 9520 P BITS FROM 9520 PM BITS
PF03: DEF 70X, B$0, 9X ; PARALLEL FETCH FROM 9403AS
PL03: DEF 71X, B$0, 8X ; PARALLEL LOAD INTO 9403AS
PREQ: DEF 72X, B$0, 7X ; PARAMETER REQUEST
RGA: DEF 73X, B$0, 6X ; READ GATE
RFF: DEF 74X, B$0, 5X ; RESET FIFO
SAST: DEF 75X, B$0, 4X ; SELECT / ATTENTION STROBE
WRGA: DEF 76X, B$0, 3X ; WRITE GATE

ASCBC: EQU QIO ; ASCII TO EBCDIC SUBSET PREFIX
BCDABC: EQU QI1 ; BCD TO EBCDIC SUBSET PREFIX
EBCS: EQU QI2 ; EBCDIC SUBSET TO ASCII PREFIX
EBDCBC: EQU QI3 ; EBCDIC SUBSET TO BCD PREFIX
XLAT: DEF 77X, 3VX $D ; TRANSLATE PREFIX

END

TOTAL PHASE 1 ERRORS = 0
; This SRC file was created for the AMD application note:
; "A High-Performance Intelligent Disk Controller"
; by Otis Tabler and Brad Kitson.
; Mnemonics and word format are defined in DISKCTRL.DEF
; Advanced Micro Devices reserves the right to make changes in its
; product without notice in order to improve design or performance
; characteristics. The company assumes no responsibility for the
; use of any circuits or programs described herein.
; Am29116 Mnemonics Copyright © 1982 Advanced Micro Devices

SECTOR READ / WRITE SUBROUTINE
******************************************************************************

INPUTS:

FUNCTION CODE IN R0:
0 TO READ SECTOR
+1 TO WRITE SECTOR
HEAD NUMBER IN MSB OF R1
MSB OF TRACK NUMBER IN LSB OF R1
LSB OF TRACK NUMBER IN MSB OF R2
SECTOR NUMBER IN LSB OF R2
START ADDRESS OF RAM SECTOR BUFFER IN R3
OUTPUT:

RO CONTAINS:

0 IF THE FUNCTION SPECIFIED WAS COMPLETED
   EITHER WITHOUT ERROR OR WITH A
   SUCCESSFULLY CORRECTED READ ERROR
+1 IF THE SECTOR'S HEADER IS BAD
+2 IF AN UNCORRECTABLE ERROR WAS DETECTED IN
   READING THE SECTOR'S DATA SEGMENT

ADDITIONAL MNEMONICS

C001 CRCMSK: EQU 16H#C001 ; CRCF POLYNOMIAL MASK
0010 CRCNIT: EQU 16 ; CRCF NUMBER OF ITERATIONS (D#16 <-- default bas.
NSPASSE: EQU 64 ; NUMBER OF SECTOR PASSES (SET THIS EQUAL
0040 ;
RDITCT: EQU 65 ; READ ITERATION COUNT, EQUAL TO THE NUMBER
; OF 16-BIT WORDS (DATA PLUS MODIFIED FIRE
0041 ;
0016 PF1:   EQU 22 ; PERIOD FACTOR ONE
000D PF2:   EQU 13 ; PERIOD FACTOR TWO
0059 PF3:   EQU 89 ; PERIOD FACTOR THREE
004: EQU 23 ; PERIOD FACTOR FOUR
0017 ;
8723 A1LSW: EQU H#8723 ; A1 CONSTANT(LEAST SIG. WORD)
0006 A1MSW: EQU 6 ; A1 CONS.(MOST SIG. WORD)
BF8 A2LSW: EQU H#BF8 ; A2 CONS.(LEAST SIG. WORD)
D530 A2LSW: EQU H#D530 ; A3' CONS.(LEAST SIG. WORD)
4A5LSW: EQU H#A928 ; A4' CONS.(LEAST SIG. WORD)
A928 ;
7100 KL128: EQU H#7100 ; K(LEAST SIG. WORD) SHIFTED UP
0477 KM128: EQU H#0477 ; K(MOST SIG. WORD) SHIFTED UP
0EE2 KLSW: EQU H#EE2 ; K(LEAST SIG. WORD)
KMSW: EQU 8 ; K(MOST SIG. WORD)
0008 ;
IF THE FUNCTION CODE IN R0 EQUALS +1 (WRITE SECTOR), PRECALCULATE
THE MODIFIED FIRE CODE'S PARTIAL CHECKSUM FOR THE FIRST HALF OF
THE DATA SEGMENT TO BE WRITTEN.

0000 SECTIO: BOR2 W0.82MR,RO
         &NODE LENIEN @NOSE5E ;--- note use of overlayed
         &CT 2 @NOJMI &CPFCD8 ; signified by "&"

RESET THE AM9520 AND THEN PLACE IT IN COMPUTE CHECK BITS MODE.
INITIALIZE COUNTER FOR CHECK BITS PRECALCULATION LOOP.

0001 SORR W,MOVE,SOI,NRV
         &NODE LENIEN @NOSE5E &NOJMI &CONT

0002 IMME H#0000
         &NODE LENIEN @NOSE5E @B20 &NOJMI &CONT

0003 SORR W,MOVE,SOI,NRV
         &NODE LENIEN @NOSE5E &NOJMI &CONT

0004 IMME H#0010
         &NODE LENIEN @NOSE5E @B20 &NOJMI &LCT 127

(R3) TO R4.

0005 SORR W,MOVE,SORY,R3
         &DLE LENIEN @NOSE5E @NOJMI &CONT

0006 SORR W,MOVE,SORY,R4
         &NODE LENIEN @NOSE5E @NOJMI &CONT

BEGIN CHECK BITS PRECALCULATION LOOP.
(R4) TO THE MAR.

0007 PCPREL: SORR W,MOVE,SORY,R4
         &NODE LENIEN @NOSE5E &NOJMI &CONT

CLOCK THE LESS SIGNIFICANT BYTE OF ((MAR)) INTO THE AM9520.

NOOP
         &NODE LENIEN @NOSE5E @B20 @NOJMI &CONT

NOOP
         &NODE LENIEN @NOSE5E @B20 @NOJMI &CONT

NOOP FOR TIMING PURPOSES.

NOOP
         &NODE LENIEN @NOSE5E &NOJMI &CONT

CLOCK THE MORE SIGNIFICANT BYTE OF ((MAR)) INTO THE AM9520.

NOOP
         &NODE LENIEN @NOSE5E @B20 @NOJMI &CONT

INCREMENT (R4).
END CHECK BITS PRECALCULATION LOOP.

000C SORR W,INC,SORY,R4
         &NODE LENIEN @NOSE5E @CP20 @NOJMI &REPC8 PCPREL
PLACE THE AM9520 IN WRITE CHECK BITS MODE.
INITIALIZE COUNTER FOR STORE CHECK BITS IN BUFFER LOOP.

000D
SORR W,MOVE,SOL,NRY
&NOJMPI &NOEY &NOSRE
&CONT

000E
IMME #0011
&NOJMPI &NOEY &NOSRE
&BT20 &NOJMPI
&LCT 2

BEGIN STORE CHECK BITS IN BUFFER LOOP.
(R4) TO THE MAR.
CLOCK OUT NEXT MODIFIED FIRE CODE BYTE TO THE
LESS-SIGNIFICANT MEMORY BUS INTERFACE REGISTER.

000F
SCBIBL SOR W,MOVE,SORY,R4
&NOJMPI &NOEY &NOSRE
&CP20 &NADR &NOJMPI
&CONT

NOOP
&NOJMPI &NOEY &NOSRE
&CE2L &NOJMPI
&CONT

NOOP FOR TIMING PURPOSES

0010
&NOJMPI &NOEY &NOSRE
&CONT

CLOCK OUT NEXT MODIFIED FIRE CODE BYTE TO THE
MORE-SIGNIFICANT MEMORY BUS INTERFACE REGISTER.

0012
&NOJMPI &NOEY &NOSRE
&CONT

NOOP FOR TIMING PURPOSES

0013
&NOJMPI &NOEY &NOSRE
&CONT

(BUS INTERFACE REGISTER PAIR) TO (MAR).
INCREMENT (R4).
END STORE CHECK BITS IN BUFFER LOOP.

0014
SOR W,INC,SORR,R4
&NOJMPI &NOSRE
&BF2L &BF2U &MWRT &NOJMPI
&RPCT SCBIBL

ZERO THE UPPER BYTE OF (R5) AND THEN CLOCK THE 7TH AND LAST
BYTE OF THE MODIFIED FIRE CODE INTO ITS LOWER BYTE.

0015
SOR W,MOVE,SORR,R5
&NOJMPI &NOSRE
&CONT

NOOP
&NOJMPI &NOEY &NOSRE
&CE2L &NOJMPI
&CONT

0017
SOR B,MOVE,SODR,R5
&NOJMPI &NOSRE
&BF2L &BT16 &NOJMPI
&CONT

(R4) TO MAR.

0018
SOR W,MOVE,SORR,R5
&NOJMPI &NOEY &NOSRE
&NADR &NOJMPI
&CONT

(R5) TO (MAR).

0019
SOR W,MOVE,SORR,R5
&NOJMPI &NOEY &NOSRE
&BF16 &MWRT &NOJMPI
&CONT
CONVERT THE FUNCTION CODE IN RO TO A MICROCODE BRANCH ADDRESS.

001A
\textbf{CFCODE: SONR} W, MOVE, SOI, NRA
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

001B
\textbf{IMME BRTAB1}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

001C
\textbf{TOR1 W, ADD, TORAA, RO}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

CRC POLYNOMIAL MASK TO ACC.

001D
\textbf{SONR} W, MOVE, SOI, NRA
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

001E
\textbf{IMME CRCMSK}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

CLEAR REGISTER USED TO ACCUMULATE CRCF.

001F
\textbf{SOR} W, MOVE, SOR, R4
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

COPY HEAD BYTE AND TRACK BYTE 1 TO R5.
SET CRCF LOOP COUNTER.

0020
\textbf{SOR} W, MOVE, SOR, R1
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &LDCT CRCNIT
\end{verbatim}

0021
\textbf{SOR} W, MOVE, SODR, R5
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

SHIFT R5 AND SET QLINK.

0022
\textbf{CRCFL1: SHFTR W, SHUI, SRRR, R5}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

ACCUMULATE CRCF.

0023
\textbf{CRCF R4}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &RPCT CRCFL1
\end{verbatim}

COPY TRACK BYTE 2 AND SECTOR BYTE TO R5.
SET CRCF LOOP COUNTER.

0024
\textbf{SOR} W, MOVE, SOR, R2
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &LDCT CRCNIT
\end{verbatim}

0025
\textbf{SOR} W, MOVE, SODR, R5
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

SHIFT R5 AND SET QLINK.

0026
\textbf{CRCFL2: SHFTR W, SHUI, SRRR, R5}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &CONT
\end{verbatim}

ACCUMULATE CRCF.

0027
\textbf{CRCF R4}
\begin{verbatim}
&NODLE &IEN &NOOEY &NOSRE
&NOMPI &RPCT CRCFL2
\end{verbatim}
INITIALIZE SECTOR PASS LOOP COUNTER.
ENTER INPUT MODE.
TURN ON READ GATE.

BEGIN SECTOR PASS LOOP.
TURN ON ADDRESS MARK CONTROL.
RESET BYTE SYNC ACQUISITION CIRCUITRY AND FIFO ARRAY.

SECTL1: NOP

PASS WHEN ADDRESS MARK DETECTED.

NOP

PASS WHEN INPUT AVAILABLE FROM FIFO ARRAY.

INPUT RECORDED HEAD NUMBER AND MSB OF RECORDED TRACK NUMBER TO R5 AND D-LATCH.
PASS WHEN INPUT AVAILABLE FROM FIFO ARRAY.

COMPARE THE CONTENTS OF R1 AND R5.
IF THEY DISAGREE, EXAMINE THE NEXT SECTOR.

INPUT LSB OF RECORDED TRACK NUMBER AND RECORDED SECTOR NUMBER TO R5 AND D-LATCH.
PASS WHEN INPUT AVAILABLE FROM FIFO ARRAY.

NOOP

NOOP

NOP

NOP

POP
COMPARE THE CONTENTS OF R2 AND R5;
IF THEY DISAGREE, EXAMINE THE NEXT SECTOR.

0030 TOR1 W.EXOR,TODRR.R2
&NODLE &NOEN &NOOEY &SRE &CT Z
&PF03 &INPT &NOJMPI &RDGA
&IF CT16 &CJP SECTL2

INPUT RECORDED CRCF BYTES 1 AND 0 TO R5 AND D-LATCH.

0031 SOR W,MOVE,SODR.R5
&DLE &IEN &NOOEY &NSRE
&BF03 &BT16 &INPT &NOJMPI &RDGA
&CONT

COMPARE THE TWO CRCFS.
IF THEY AGREE, PROCEED TO READ OR WRITE AS SPECIFIED BY RO.
OTHERWISE, ASSUME BAD HEADER, TRUE ID UNKNOWN,
AND CONTINUE LOOP.

0032 TOR1 W.EXOR,TODRR.R4
&NODLE &NOEN &NOOEY &SRE &CT Z
&INPT &NOJMPI &RDGA
&IF CT16 &CJP MATCH1

TURN OFF READ GATE.
LEAVE INPUT MODE.
END SECTOR PASS LOOP.
NOTICE WE HAVE THREE MICROINSTRUCTION CLOCKS LEFT BEFORE
IT IS TIME TO BEGIN WRITING OR RE-SYNC AND BEGIN READING.

0033 SECTL2: SOR W,MOVE,SORY,RO
&NODLE &NOEN &NOOEY &NSRE
&RPCT SECTL1

IF SECTOR SEARCH COUNT EXHAUSTED, LOAD +1 INTO RO AND RETURN

0034 BOR2 W,LD2NR,0,RO
&NODLE &IEN &NOOEY &NSRE
&JMPI &RTN

SUCCESSFUL MATCH.

0035 MATCH1: SOR W,MOVE,SORY,RO
&NODLE &NOEN &NOOEY &NSRE
&JMPI &CONT

0036 BRTABL: SOR W,NEG,SORA,R3
&NODLE &IEN &NOOEY &NSRE
&NOJMPI &RFIF &JP RDSEC1

0037 SOR W,NEG,SORA,R3
&NODLE &IEN &NOOEY &NSRE
&NOJMPI &RFIF &JP WRSEC1
READ SECTOR

(R3) TO R4 AND MAR.
ENTER INPUT MODE AGAIN.
TURN READ GATE BACK ON.
INITIALIZE COUNTER FOR READ DATA SEGMENT LOOP.
DURING THAT LOOP, AM9520 READ HIGH SPEED IS PERFORMED ON THE
FIRST HALF OF THE SEGMENT BUFFER. THE SECOND HALF OF THE BUFFER
IS PROCESSED BY THE AM9520 IN THE READ HIGH SPEED COMPLETION LOOP.

0030 READC1: SOR W,NEG,SOAR,R4
&NODE &IEN &OEY &NOSRE
&INPT &NOJMPI &MADR &RPGA &RPIF
&LDCT &DICTT

(R3) = 1 TO R5.
PASS WHEN INPUT AVAILABLE FROM FIFO ARRAY.

0034 SOR W,CMP,SOAR,R5
&NODE &IEN &NOOEY &NOSRE
&INPT &NOJMPI &RDGA
&IFNOT RDY &CJP S
RESET THE AM9520 AND THEN PLACE IT IN READ HIGH SPEED MODE.

0036 SONR W,MOVE,SOI,NRY
&NODE &NOIEN &OEY &NOSRE
&INPT &NOJMPI &RDGA
&CONT

0038 IMME #0003
&NODE &NOIEN &OEY &NOSRE
&INPT &NOJMPI &RDGA
&CONT

003C SONR W,MOVE,SOI,NRY
&NODE &NOIEN &OEY &NOSRE
&INPT &NOJMPI &RDGA
&CONT

003D IMME #0013
&NODE &NOIEN &OEY &NOSRE
&BT20 &INPT &NOJMPI &RDGA
&CONT

003E NOOP
&NODE &NOIEN &NOOEY &NOSRE
&INPT &NOJMPI &PF03 &RDGA
&CONT
BEGIN READ DATA SEGMENT LOOP.
TRANSFER NEXT WORD FROM FIFO ARRAY TO (MAR).
PASS WHEN FIFO INPUT AGAIN BECOMES AVAILABLE.

RDSEC2: NOOP

003F
&NODE &NOEI &NOOY &NOSE
&BF03 &INPT &NOJMPI &MWR &RDGA
&FNOD &RDY &CJP $ 
;
INCREMENT (R5) AND TRANSFER THIS TO THE MAR.

0040 SOR W,INC,SORR,R5
&NODE &IE &OEY &NOSRE
&INPT &NOJMPI &MADR &RDGA
&CONT
;
CLOCK LESS SIGNIFICANT BYTE OF (MAR) INTO THE AM9520.
INCREMENT (R4) AND TRANSFER THIS TO THE MAR.

0041 SOR W,INC,SORR,R4
&NODE &IE &OEY &NOSRE
&BT2L &CP20 &INPT &NOJMPI &MADR &MR8A &PF03 &RDGA
&CONT
;
TRANSFER NEXT WORD FROM FIFO ARRAY TO (MAR).
PASS WHEN FIFO INPUT AGAIN BECOMES AVAILABLE.

NOOP

0042 &NODE &NOEI &NOOY &NOSE
&BF03 &INPT &NOJMPI &MWR &RDGA
&FNOD &RDY &CJP $ 
;
TRANSFER (R5) TO THE MAR AGAIN.

0043 SOR W,MOVE,SORR,R5
&NODE &IE &OEY &NOSRE
&INPT &NOJMPI &MADR &RDGA
&CONT
;
THIS TIME, CLOCK THE MORE SIGNIFICANT BYTE OF ((MAR))
INTO THE AM9520.
END READ DATA SEGMENT LOOP.

0044 SOR W,INC,SORR,R4
&NODE &IE &OEY &NOSRE
&BT2U &CP20 &INPT &NOJMPI &MADR &MREA &PF03 &RDGA
&RPCT RDSEC2
;
INITIALIZE COUNTER FOR READ HIGH SPEED COMPLETION LOOP.

NOOP

0045 &NODE &NOEI &NOOY &NOSRE
&NOJMPI
&LPCT RDITCT
BEGIN READ HIGH SPEED COMPLETION LOOP.
INCREMENT (R5) AND TRANSFER THIS TO THE MAR.

RDSEC3: SOR W,INC,SORR,R5
// &NODELE &IDEN &GEOY &NOSRE
// &NOJMPI &MADR &CONT

NOOP FOR TIMING PURPOSES

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &NOJMPI &CONT

CLOCK LESS SIGNIFICANT BYTE OF ((MAR)) INTO THE AM9520.

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &BT2L &CP20 &NOJMPI &MREA
// &CONT

NOOP FOR TIMING PURPOSES

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &NOJMPI &CONT

NOOP FOR TIMING PURPOSES

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &NOJMPI &CONT

CLOCK MORE SIGNIFICANT BYTE OF ((MAR)) INTO THE AM9520.

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &FCT RDSEC3
// &CONT

RDSEC3 WAS AN ERROR DETECTED BY THE AM9520?

NOOP
// &NODELE &NOIEN &NOOXY &NOSRE
// &FIF ER20 &CJP RDSEC4

NO; LOAD 0 INTO RO AND RETURN.

SOR W,MOVE,SIZA,RO
// &NODELE &IDEN &NOOXY &NOSRE
// &NOJMPI &KTH

YES; IF THE ERROR IS A CORRECTABLE ONE, LOCATE AND CORRECT IT.
THE ERROR IS LOCATED USING THE CORRECT HIGH SPEED EQUATION:

L = NK - (M1A1 + M2A2 + M3A3 + M4A4)

WHERE K,A1,A2,A3,A4 ARE CONSTANTS
AND M1,M2,M3,M4 MUST BE CALCULATED.
THE ERROR IS CORRECTED BY PERFORMING AN EXOR FUNCTION ON THE BURST ERROR IN MEMORY WITH AN ERROR PATTERN (EP) PROVIDED BY THE BEP(9520).

INITIALIZE CORRECT HIGH SPEED, SET P0=1, & REP=1
0044 RDSEC4: SONR W,MOVE,SOI,NSY
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &CONT
;
0045 IMME H#001F
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &BP10 &FP16
/ &CONT
;
CLEAR R8(M1).
;
0050 SONR W,MOVE,SOZR,R8
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &CONT
;
PERIOD FACTOR 1(PF1) TO ACC.
;
0051 SONR W,MOVE,SOI,NRA
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &CONT
;
0052 IMME PF1
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &CONT
;
TEST FOR ERROR PATTERN PRESENT.
;
0053 NOOP
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI
/ &IF EP20 &CJP M2341
;
EP NOT PRESENT,
DECREMENT ACC.
TEST FOR ALIGNMENT EXCEPTION(AE).
;
0054 BONR W,0,SZNA
/ &NODE &NONIY &OEY &SRE
/ &NOJMPI &CP20
/ &IF AE20 &CJP AE
;
AE NOT PRESENT,
ADD 8 TO R8.
;
0055 BOR2 W,3,AZNR,R8
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI
/ &JP LINK
;
AE PRESENT;
INC R8.
;
0056 AE: BOR2 W,0,AZNR,R8
/ &NODE &NONIY &OEY &NOSRE
/ &NOJMPI &CONT
TEST FOR PERIOD FACTOR EXCEEDED (UNCORRECTABLE ERROR).

LINK: NOOP &NODEL &NOISE &NOSRE &CT N &NOJMPI &IF NOT CT16 &CJP M1

PERIOD FACTOR EXCEEDED (UNCORRECTABLE ERROR); SET RO = 2.
RETURN.

0058 ERR: BOR2 W,1,LD2NR,RO
&NODEL &IEN &OEY &NOSRE &NOJMPI &BT20 &BF16 &BN

EP PRESENT, CALCULATE M2,M3,M4.
PERIOD FACTOR 2(PF2) TO R9(M2).
PERIOD FACTOR 3(PF3) TO R10(M3).
PERIOD FACTOR 4(PF4) TO R11(M4).

0059 M2345: SOR W,MOVE,SOI,R9
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

005A IMME PF2
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

005B SOR W,MOVE,SOI,R10
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

005C IMME PF3
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

005D SOR W,MOVE,SOI,R11
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

005E IMME PF4
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

JUMP TABLE ADDRESS(TAB1) TO R12.

005F SOR W,MOVE,SOI,R12
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

0060 IMME TAB1
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &CONT

JUMP INDIRECT TO TAB1 VIA PM2-4.

0061 ROTM W,15,MDRI,R12
&NODEL &IEN &NOOEY &NOSRE &NOJMPI &PFP &BF2U &BT16 &CONT

0062 IMME H#0007
&NODEL &IEN &OEY &NOSRE &JMPI &PFP &BF2U &BT16 &JP &
R9 = PF2 - R9.
0063 MFIX:  TOR1 W,TORIR,SUBR,R9
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
0064  IMME PF2
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
     R10 = PF3 - R10.
0065  TOR1 W,TORIR,SUBR,R10
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
0066  IMME PF3
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
     R11 = PF4 - R11.
0067  TOR1 W,TORIR,SUBR,R11
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
0068  IMME PF4
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
     0 TO R7 (LOCATION ACC MSW,LAC).
0069 M1A1:  SOR W,MOVE,SOIR,R7
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
     A1 TO R12(LSW), R13(MSW).
006A  SOR W,MOVE,SOI,R12
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
006B  IMME A1LSW
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
006C  SOR W,MOVE,SOI,R13
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &CONT
006D  IMME A1MSW
     &NODLE &IEN &NOOEEY &NOSRE
     &NOJMPI &LDCT 4
     R8 TO D.
     LAC = M1A1, MULTIPLY M1A1.
006E  SOR W,MOVE,SOIR,R8
     &NODLE &NOIEN &OEY &NOSRE
     &NOJMPI &JE MUL
A2 TO R12, R13.

006F M2A2:
IMME A2LSW
&NODLE &IEN &NOOEU &NOSRE
&NOJMPI
&CONT
;

0070 R9 TO D.
LAC = LAC + M2A2.

0071 SOR W, MOVE, SORY, R9
&DLE &NOIEN &OEY &NOSRE
&NOJMPI
&JS MUL

A3'(A3 - 4K) TO R12, R13.

0072 M3A3:
IMME A3LSW
&NODLE &IEN &NOOEU &NOSRE
&NOJMPI
&CONT
;

0073 R10 TO D.
LAC = LAC + M3A3.

0074 SOR W, MOVE, SORY, R10
&DLE &NOIEN &OEY &NOSRE
&NOJMPI
&JS MUL

A4'(A4 - 4K) TO R12, R13.

0075 M4A4:
IMME A4LSW
&NODLE &IEN &NOOEU &NOSRE
&NOJMPI
&CONT
;

0076 R11 TO D.
LAC = LAC + M4A4.

0077 SOR W, MOVE, SORY, R11
&DLE &NOIEN &OEY &NOSRE
&NOJMPI
&JS MUL

PRESHIFTED DIVISOR(K) TO R12, R13, D.
LAC = REM(M1A1 + M2A2 + M3A3 + M4A4) / K.
LAC = -L + K.

0078 IMME KL128
&NODLE &IEN &NOOEU &NOSRE
&NOJMPI
&CONT
;

0079 SOR W, MOVE, SORI, R13
&DLE &IEN &OEY &NOSRE
&NOJMPI
&LDCT 6

007A IMME KM128
&DLE &IEN &OEY &NOSRE
&NOJMPI
&JS DIV
\( L = L_{AC} - K - L_{AC} \)

007B SUBK:
\[
\text{IM} \quad \text{KLSW} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\
\]

007C	\text{TOR1 \ W,TORIR, SUBRC,R7} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

007D	\text{IMME \ R5SW} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

0 TO R6.

007E XORMEM:
\[
\text{SCR \ W,MOVE, SOZR, R8} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\
\]

0 TO R9.

007F	\text{SOR \ W, MOVE, SOZR, R9} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

0 TO ACC.

0080	\text{SOR \ W, MOVE, SCRA, R9} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

\text{ROTATE \ R6 \ DOWN \ by \ four \ to \ obtain \ word \ address} \\
\text{and \ store \ in \ ACC.}

0081	\text{BOTH \ W, 12, MRAI, R6} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

0082	\text{IMME \ #OFFF} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

\text{MASK \ upper \ 12 \ bits \ of \ R6 \ to \ obtain \ first \ bit \ of} \\
\text{burst \ error \ and \ store \ in \ R6.}

0083	\text{TOR1 \ W, TORIR, AND, R6} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

0084	\text{IMME \ #000F} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

\text{Jump \ indirect \ to \ TAB2 \ via \ R6.}

0085	\text{TOR1 \ W, TORIR, ADD, R6} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOOYE} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT} \\

0086	\text{IMME \ TAB2} \\
\quad \text{&NODLE} \quad \text{&IEN} \quad \text{&NOEY} \quad \text{&NOSRE} \\
\quad \text{&NJMPI} \\
\quad \text{&CONT}
MADR = ACC = R4 - ACC.

0087 XOR: TOR1 W, TORAA, SUBS, R4
 & NODLE & IEN & NOOEY & NOSRE
 & CONT

0088 SORR W, R2N & NODLE & IEN & NOOEY & NOSRE
 & CONT & NOJMPl
 & CONT

R8 = R8 XOR MEM.

0089 TOR1 W, TODRR, EXOR, R8
 & DLE & IEN & NOOEY & NOSRE
 & NOJMPl & BT16
 & CONT

R8 TO MEMORY.

008A SOR W, MOVE, SORY, R8
 & NODLE & NOIEN & OY & NOSRE
 & NOJMPl & MWRT & BF16
 & CONT

MADR = ACC + 1.

008B SORR W, INC, SOA, NY
 & NODLE & NOIEN & OY & NOSRE
 & NOJMPl & MADR
 & CONT

R9 = R9 XOR MEM.

008C TOR1 W, TODRR, EXOR, R9
 & DLE & IEN & NOOEY & NOSRE
 & NOJMPl & BT16
 & CONT

R9 TO MEMOY.

008D SOR W, MOVE, SORY, R9
 & NODLE & NOIEN & OY & NOSRE
 & NOJMPl & MWRT & BF16
 & CONT

0 TO R0 (ERROR CORRECTED FLAG).
 RETURN.

008E SOR W, MOVE, SOZ, R0
 & NODLE & IEN & GOEY & NOSRE
 & NOJMPl & BT20 & BF16
 & RTN
TABLE 1 IS USED TO CALCULATE M2, M3, M4 AND DETECT UNCORRECTABLE ERRORS.
EACH MICROINSTRUCTION PATH DECREMENTS THE APPROPRIATE REGISTER(S) AND CHECKS FOR VALUES
EXCEEDING THE 56-BIT POLYNOMIAL PERIOD FACTOR LIMITS.

ALIGN 8

0090 TAB1: BOR2 W,0,S2NR,R9
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP PM34

0091 BOR2 W,0,S2NR,R11
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM34

0092 BOR2 W,0,S2NR,R9
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM24

0093 BOR2 W,0,S2NR,R11
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM1

0094 BOR2 W,0,S2NR,R10
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM23

0095 BOR2 W,0,S2NR,R10
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM1

0096 BOR2 W,0,S2NR,R9
&NODLE &IEN &NOOEY &SRE
&NJMPI &CP20 &FPFM
&JP TM1

0097 SOR W,MOVE,BOIR,R6
&NODLE &IEN &NOOEY &SRE
&NJMPI &FPFM
&JP MFIX

TM34: NOOP

0098 &NODLE &NOISN &NOOEY &NOSRE
&NJMPI &CP20 &FPFM
&JP PM34

TM24: NOOP

0099 &NODLE &NOISN &NOOEY &NOSRE
&NJMPI &CP20 &FPFM
&JP PM24

TM23: NOOP

009A &NODLE &NOISN &NOOEY &NOSRE
&NJMPI &CP20 &FPFM
&JP PM23

TM1: NOOP

009B &NODLE &NOISN &NOOEY &NOSRE
&NJMPI &CP20 &FPFM
&CONT

NOOP

009C &NODLE &NOISN &NOOEY &NOSRE
&NJMPI &CP20 &FPFM
&JP PM1

009D PM34: BOR2 W,0,S2NR,R11
&NODLE &IEN &NOOEY &SRE &CT N
&NJMPI &CP20 &FPFM
&IF CT16 &CP ERR

009E PM34: BOR2 W,0,S2NR,R10
&NODLE &IEN &NOOEY &SRE &CT N
&NJMPI &CP20 &FPFM
&IF CT16 &CP ERR

009F PM1: ROTM W,15,MDR1,R12
&DLE &IEN &NOOEY &NOSRE &CT N
&NJMPI &FPFM &BT16 &BF2U
&IF CT16 &BP LINK

00A0 IMME W,0007
&NODLE &IEN &NOOEY &NOSRE
&NJMPI &FPFM &BT16 &BF2U
&JP 5
TABLE 2 is used for rotating memory word(s) via rotate and merge instruction(s) so that burst errors can be aligned with the error pattern provided by the BEP(9520).

00A5 TAB2: ROTM W,9,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP1

00A6 ROTM W,10,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP2

00A7 ROTM W,11,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP3

00A8 ROTM W,12,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP4

00A9 ROTM W,13,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP5

00AA ROTM W,14,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP6

00AB ROTM W,15,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP7

00AC ROTM W,0,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP8

00AD ROTM W,1,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP9

00AE ROTM W,2,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP10

00AF ROTM W,3,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP11

00B0 ROTM W,4,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP12

00B1 ROTM W,5,MDRI,RS
// &DLE &IEN &NODLE &NOSRE
// &NOJMPI &BT16 &BF2U &BF2L
// &JMP REP13
00B2  ROTM  W,6,MDR1,R8
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  REP14
;
00B3  ROTM  W,7,MDR1,R8
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  REP15
;
00B4  ROTM  W,8,MDR1,R8
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  REP16
;
00B5  REP1:  1MME  H00001
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM1
;
00B6  REP2:  1MME  H00002
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM2
;
00B7  REP3:  1MME  H00003
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM3
;
00B8  REP4:  1MME  H00004
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM4
;
00B9  REP5:  1MME  H00005
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM5
;
00BA  REP6:  1MME  H00006
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM6
;
00BB  REP7:  1MME  H00007
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM7
;
00BC  REP8:  1MME  H00008
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM8
;
00BD  REP9:  1MME  H00009
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM9
;
00BE  REP10:  1MME  H00010
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM10
;
00BF  REP11:  1MME  H00011
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  RM11
;
00C0  REP12:  1MME  H00012
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  XOR
;
00C1  REP13:  1MME  H00013
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  XOR
;
00C2  REP14:  1MME  H00014
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  XOR
;
00C3  REP15:  1MME  H00015
/  &DLE  61EN  &NOSRE  &NOJMP  6BF2U  6BF2L
/  &JM  XOR
;
54
SUBROUTINE MULTIPLY


MUL: SHFTR W,SHDR,SHDNZ,R8
&NODEL &IE:N &NOOEY &NOSRE
&NOJMPI
&CONT

SOR W,MOVE,SORY,R13
&NODEL &NOIE: &NOEY &NOSRE
&NOJMPI
&CONT

CYC: SOR W,MOVE,SORR,R12
&NODEL &IEIEN &NOEY &NOSRE &CT L
&NOJMPI
&IFNOT CT16 &CJP NOTQ

TORI W,TORAR,ADD,R6
&NODEL &NOIE: &NOEY &NRE
&NOJMPI
&CONT

TORI W,TORAR,ADD,R7
&NODEL &IEIEN &NOEY &NOSRE
&NOJMPI
&CONT
SUBROUTINE DIVIDE

THIS SUBROUTINE DIVIDES A DOUBLE PRECISION NUMBER BY A DOUBLE PRECISION NUMBER LEAVING ONLY A REMAINDER. THE DIVISOR IS IN R12, R13, D AND THE DIVIDEND/REMAINDER APPEARS IN R6, R7. NOTE, UPON RETURN AN IMMEDIATE SUBTRACT IS INITIATED.

DIV: SOR W, MOVE, SORA, R12

CYC1: TORI W, TORAR, SUBS, R6

CYC2: TORI W, TODRR, SUBSC, R7

NOOP

IFNOT CT16 &CJP POS

ADD: TORI W, TORAR, ADD, R6

ADDC: TORI W, TODRR, ADDC, R7
WRITE SECTOR

(R3) TO R4 AND MAR.
ENTER OUTPUT MODE. (TURNS ON WRITE CLOCK.)
INITIALIZE COUNTER FOR OUTPUT DATA PREAMBLE LOOP.

WRSEC1: SOR W,NEG,SOAR,R4
&NODLE &IEN &NOOEY &NSRE
&MADE &NOMPI &OUPT
&LCNT 5
BEGIN WRITE DATA PREAMBLE LOOP.

WRITE SEC2: NOOP

WRSEC2: NOOP

WRSEC3: NOOP

OUTPUT H#D000 TO FIFO ARRAY.

WRSEC3: NOOP

WRITE LAST DATA PREAMBLE BYTE AND THE H#FE DATA SYNC BYTE.
INITIALIZE COUNTER FOR WRITE DATA SEGMENT LOOP.
PASS WHEN FIFO ARRAY AGAIN READY FOR OUTPUT.

WRSEC1: SOR W,MOVE,SOI,NRY
&NODLE &NOIEN &NOOEY &NSRE
&MADE &NOMPI &OUPT &WRGA
&LDCT 131
IMME HFF00
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&IFNOT RDYO &CJP WRSEC3

END WRITE DATA PREAMBLE LOOP.

WRSEC1: SOR W,MOVE,SOI,NRY
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&LDCT  131
SONR W,MOVE,SOZ,NRY
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&CONT
BEGIN WRITE DATA PREAMBLE LOOP.

TURN ON WRITE GATE.
PASS WHEN FIFO ARRAY READY FOR OUTPUT.

WRSEC3: NOOP

END WRITE DATA PREAMBLE LOOP.

WRSEC3: NOOP

WRITE LAST DATA PREAMBLE BYTE AND THE H#FE DATA SYNC BYTE.
INITIALIZE COUNTER FOR WRITE DATA SEGMENT LOOP.
PASS WHEN FIFO ARRAY AGAIN READY FOR OUTPUT.

WRSEC3: NOOP

WRITE LAST DATA PREAMBLE BYTE AND THE H#FE DATA SYNC BYTE.
INITIALIZE COUNTER FOR WRITE DATA SEGMENT LOOP.
PASS WHEN FIFO ARRAY AGAIN READY FOR OUTPUT.

WRSEC1: SOR W,MOVE,SOI,NRY
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&LDCT  131
IMME HFF00
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&IFNOT RDYO &CJP WRSEC3

END WRITE DATA PREAMBLE LOOP.

WRSEC3: NOOP

WRITE LAST DATA PREAMBLE BYTE AND THE H#FE DATA SYNC BYTE.
INITIALIZE COUNTER FOR WRITE DATA SEGMENT LOOP.
PASS WHEN FIFO ARRAY AGAIN READY FOR OUTPUT.

WRSEC1: SOR W,MOVE,SOI,NRY
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&LDCT  131
IMME HFF00
&NODLE &NOIEN &NOOEY &NSRE
&NOMPI &OUPT &WRGA
&IFNOT RDYO &CJP WRSEC3

END WRITE DATA PREAMBLE LOOP.
BEGIN WRITE DATA SEGMENT LOOP.
(R4) TO MAR.

00F4 WRSEC4: SOR W, MOVE, SORY, R4

INCREMENT (R4).

00F5 SOR W, INC, SORR, R4

((MAR)) TO FIFO ARRAY.

END WRITE DATA SEGMENT LOOP.

NOOP

00F6 / &NODLE &NODIE &NOOEY &NOSRE
&BT03 &MREA &NOJMPI &OUPT &PL03 &WRGA
&RPCT WRSEC4

CLEAR (R4).

INITIALIZE COUNTER FOR WRITE DATA POSTAMBLE LOOP.

00F7 SOR W, MOVE, SOZR, R4

BEGIN WRITE DATA POSTAMBLE LOOP.

OUTPUT H#0000 TO FIFO ARRAY.

00F8 WRSEC5: SOR W, MOVE, SORY, R4

&NODLE &NOIEN &NOEY &NOSRE
&NOJMPI &OUPT &WRGA
&IFNOT NODYO &CUP S

00F9 SOR W, MOVE, SORY, R4

&NODLE &NOIEN &NOEY &NOSRE
&BF16 &BT03 &NOJMPI &OUPT &PL03 &WRGA
&CONT

(NOOP FOR TIMING PURPOSES)

END WRITE DATA POSTAMBLE LOOP.

NOOP

00FA / &NODLE &NOIEN &NOOEY &NOSRE
&NOJMPI &OUPT &WRGA
&RPCT WRSEC5

TURN OFF WRITE GATE.

NOOP

00FB / &NODLE &NOIEN &NOOEY &NOSRE
&NOJMPI &OUPT
&CONT

THEN LEAVE OUTPUT MODE. (TURNS OFF WRITE CLOCK.)
LOAD O INTO RO.
RETURN.

00FC SOR W, MOVE, SOZR, RO

&NODLE &LEON &NOOEY &NOSRE
&NOJMPI
&RTN

END
<table>
<thead>
<tr>
<th>SYMBOLS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1LSW</td>
<td>E723</td>
</tr>
<tr>
<td>A1MSW</td>
<td>0006</td>
</tr>
<tr>
<td>A2LSW</td>
<td>BFPA8</td>
</tr>
<tr>
<td>A2NA</td>
<td>0004</td>
</tr>
<tr>
<td>A2NDY</td>
<td>0014</td>
</tr>
<tr>
<td>A2NR</td>
<td>000E</td>
</tr>
<tr>
<td>A3LSW</td>
<td>D530</td>
</tr>
<tr>
<td>A4LSW</td>
<td>A928</td>
</tr>
<tr>
<td>ADD</td>
<td>0004</td>
</tr>
<tr>
<td>ADDC</td>
<td>0005</td>
</tr>
<tr>
<td>AE</td>
<td>0056</td>
</tr>
<tr>
<td>AE20</td>
<td>0008</td>
</tr>
<tr>
<td>AND</td>
<td>0006</td>
</tr>
<tr>
<td>ASCII</td>
<td>0000</td>
</tr>
<tr>
<td>ATTN</td>
<td>0010</td>
</tr>
<tr>
<td>B</td>
<td>0000</td>
</tr>
<tr>
<td>BACK</td>
<td>0011</td>
</tr>
<tr>
<td>BCDEBC</td>
<td>0001</td>
</tr>
<tr>
<td>BRTABL</td>
<td>0036</td>
</tr>
<tr>
<td>BUSY</td>
<td>0012</td>
</tr>
<tr>
<td>C</td>
<td>0005</td>
</tr>
<tr>
<td>CDIAI</td>
<td>0002</td>
</tr>
<tr>
<td>CDRAI</td>
<td>0004</td>
</tr>
<tr>
<td>CDRI</td>
<td>0003</td>
</tr>
<tr>
<td>CFCODE</td>
<td>001A</td>
</tr>
<tr>
<td>COMP</td>
<td>000D</td>
</tr>
<tr>
<td>CRAI</td>
<td>0005</td>
</tr>
<tr>
<td>CRCFL1</td>
<td>0022</td>
</tr>
<tr>
<td>CRCFL2</td>
<td>0026</td>
</tr>
<tr>
<td>CRCMSK</td>
<td>C001</td>
</tr>
<tr>
<td>CRCNIT</td>
<td>0010</td>
</tr>
<tr>
<td>CT16</td>
<td>0009</td>
</tr>
<tr>
<td>CYC</td>
<td>000D</td>
</tr>
<tr>
<td>CYCL</td>
<td>00E5</td>
</tr>
<tr>
<td>DIV</td>
<td>00E4</td>
</tr>
<tr>
<td>EBCASC</td>
<td>0002</td>
</tr>
<tr>
<td>EBCBCD</td>
<td>0003</td>
</tr>
<tr>
<td>EP20</td>
<td>000A</td>
</tr>
<tr>
<td>ER20</td>
<td>000B</td>
</tr>
<tr>
<td>ERR</td>
<td>0058</td>
</tr>
<tr>
<td>EXNOR</td>
<td>000B</td>
</tr>
<tr>
<td>EXOR</td>
<td>0008</td>
</tr>
<tr>
<td>F1</td>
<td>0009</td>
</tr>
<tr>
<td>F2</td>
<td>000A</td>
</tr>
<tr>
<td>F3</td>
<td>000B</td>
</tr>
<tr>
<td>FAIL</td>
<td>000C</td>
</tr>
<tr>
<td>INC</td>
<td>000E</td>
</tr>
<tr>
<td>INDEX</td>
<td>0013</td>
</tr>
<tr>
<td>K1128</td>
<td>7100</td>
</tr>
<tr>
<td>KLSW</td>
<td>EE2</td>
</tr>
<tr>
<td>KM128</td>
<td>0477</td>
</tr>
<tr>
<td>KMSW</td>
<td>0008</td>
</tr>
<tr>
<td>L</td>
<td>L2NA</td>
</tr>
<tr>
<td>LD2NA</td>
<td>0006</td>
</tr>
<tr>
<td>LD2NY</td>
<td>0016</td>
</tr>
<tr>
<td>LD2NR</td>
<td>000C</td>
</tr>
<tr>
<td>LDC2NA</td>
<td>0007</td>
</tr>
<tr>
<td>LDC2NY</td>
<td>0017</td>
</tr>
<tr>
<td>LINK</td>
<td>0057</td>
</tr>
<tr>
<td>LOW</td>
<td>0004</td>
</tr>
<tr>
<td>M1</td>
<td>0053</td>
</tr>
<tr>
<td>M1A1</td>
<td>0069</td>
</tr>
<tr>
<td>M234I</td>
<td>0059</td>
</tr>
<tr>
<td>M2A2</td>
<td>006F</td>
</tr>
<tr>
<td>M3A3</td>
<td>0072</td>
</tr>
<tr>
<td>M4A4</td>
<td>0075</td>
</tr>
<tr>
<td>MARI</td>
<td>000C</td>
</tr>
<tr>
<td>MATCH1</td>
<td>0035</td>
</tr>
<tr>
<td>MDAI</td>
<td>0007</td>
</tr>
<tr>
<td>MDAR</td>
<td>0008</td>
</tr>
<tr>
<td>MDRA</td>
<td>000A</td>
</tr>
<tr>
<td>MDRI</td>
<td>0009</td>
</tr>
<tr>
<td>MFIX</td>
<td>0063</td>
</tr>
<tr>
<td>MOVE</td>
<td>000C</td>
</tr>
<tr>
<td>MRAI</td>
<td>000E</td>
</tr>
<tr>
<td>MUL</td>
<td>000D</td>
</tr>
<tr>
<td>N</td>
<td>0007</td>
</tr>
<tr>
<td>NO</td>
<td>0000</td>
</tr>
<tr>
<td>N1</td>
<td>0001</td>
</tr>
<tr>
<td>N2</td>
<td>0002</td>
</tr>
<tr>
<td>N3</td>
<td>0003</td>
</tr>
<tr>
<td>N4</td>
<td>0004</td>
</tr>
<tr>
<td>N5</td>
<td>0005</td>
</tr>
<tr>
<td>N6</td>
<td>0006</td>
</tr>
<tr>
<td>N7</td>
<td>0007</td>
</tr>
<tr>
<td>N8</td>
<td>0008</td>
</tr>
<tr>
<td>N9</td>
<td>0009</td>
</tr>
<tr>
<td>NA</td>
<td>000A</td>
</tr>
<tr>
<td>NAND</td>
<td>0007</td>
</tr>
<tr>
<td>NB</td>
<td>000B</td>
</tr>
<tr>
<td>NC</td>
<td>000C</td>
</tr>
<tr>
<td>ND</td>
<td>000D</td>
</tr>
<tr>
<td>NE</td>
<td>000E</td>
</tr>
<tr>
<td>NEG</td>
<td>000F</td>
</tr>
<tr>
<td>NF</td>
<td>000F</td>
</tr>
<tr>
<td>NO</td>
<td>0001</td>
</tr>
<tr>
<td>NOR</td>
<td>0009</td>
</tr>
<tr>
<td>NOTQ</td>
<td>000E</td>
</tr>
<tr>
<td>NOZ</td>
<td>0000</td>
</tr>
<tr>
<td>NRA</td>
<td>0001</td>
</tr>
<tr>
<td>NRAS</td>
<td>0005</td>
</tr>
<tr>
<td>NRS</td>
<td>0004</td>
</tr>
<tr>
<td>NRY</td>
<td>0000</td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>S2NA</td>
<td>0005</td>
</tr>
<tr>
<td>S2NDY</td>
<td>0015</td>
</tr>
<tr>
<td>S2NR</td>
<td>000F</td>
</tr>
<tr>
<td>SAMD</td>
<td>0014</td>
</tr>
<tr>
<td>SCBIBL</td>
<td>000F</td>
</tr>
<tr>
<td>SECT10</td>
<td>0000</td>
</tr>
<tr>
<td>SECTL1</td>
<td>0029</td>
</tr>
<tr>
<td>SECTL2</td>
<td>0033</td>
</tr>
<tr>
<td>SETNA</td>
<td>0002</td>
</tr>
<tr>
<td>SETND</td>
<td>0012</td>
</tr>
<tr>
<td>SETNR</td>
<td>000D</td>
</tr>
<tr>
<td>SF1</td>
<td>0006</td>
</tr>
<tr>
<td>SF2</td>
<td>0009</td>
</tr>
<tr>
<td>SF3</td>
<td>000A</td>
</tr>
<tr>
<td>SHA</td>
<td>0006</td>
</tr>
<tr>
<td>SHD</td>
<td>0007</td>
</tr>
<tr>
<td>SHDN1</td>
<td>0005</td>
</tr>
<tr>
<td>SHDNC</td>
<td>0007</td>
</tr>
<tr>
<td>SHDNL</td>
<td>0006</td>
</tr>
<tr>
<td>SHDNOV</td>
<td>0008</td>
</tr>
<tr>
<td>SHDNZ</td>
<td>0004</td>
</tr>
<tr>
<td>SHDR</td>
<td>0007</td>
</tr>
<tr>
<td>SHR</td>
<td>0006</td>
</tr>
<tr>
<td>SHUP1</td>
<td>0001</td>
</tr>
<tr>
<td>SHUP2</td>
<td>0000</td>
</tr>
<tr>
<td>SL</td>
<td>0005</td>
</tr>
<tr>
<td>SOA</td>
<td>0004</td>
</tr>
<tr>
<td>SOAR</td>
<td>0004</td>
</tr>
<tr>
<td>SOD</td>
<td>0006</td>
</tr>
<tr>
<td>SOAR</td>
<td>0004</td>
</tr>
<tr>
<td>SODR</td>
<td>0006</td>
</tr>
<tr>
<td>SOI</td>
<td>0007</td>
</tr>
<tr>
<td>SOIR</td>
<td>0007</td>
</tr>
<tr>
<td>SON2C</td>
<td>0003</td>
</tr>
<tr>
<td>SORA</td>
<td>0000</td>
</tr>
<tr>
<td>SOR</td>
<td>0008</td>
</tr>
<tr>
<td>SORA</td>
<td>0000</td>
</tr>
<tr>
<td>SORR</td>
<td>0008</td>
</tr>
<tr>
<td>SORS</td>
<td>0003</td>
</tr>
<tr>
<td>SORY</td>
<td>0002</td>
</tr>
<tr>
<td>SOSE</td>
<td>000A</td>
</tr>
<tr>
<td>SOSER</td>
<td>000A</td>
</tr>
<tr>
<td>SOZ</td>
<td>0008</td>
</tr>
<tr>
<td>SOZ2</td>
<td>0009</td>
</tr>
<tr>
<td>SOZER</td>
<td>0009</td>
</tr>
<tr>
<td>SOZR</td>
<td>0008</td>
</tr>
<tr>
<td>SUBK</td>
<td>0078</td>
</tr>
<tr>
<td>SUBR</td>
<td>0000</td>
</tr>
<tr>
<td>SUBRC</td>
<td>0001</td>
</tr>
<tr>
<td>SUBS</td>
<td>0002</td>
</tr>
<tr>
<td>SUBSC</td>
<td>0003</td>
</tr>
</tbody>
</table>

**TOTAL PHASE 2 ERRORS = 0**
The International Standard of Quality guarantees these electrical AQLs on all parameters over the operating temperature range: 0.1% on MOS RAMs & ROMs; 0.2% on Bipolar Logic & Interface; 0.3% on Linear, LSI Logic & other memories.